GT 4.2.0 CAS User's Guide

DRAFT

GT 4.2.0 CAS User's Guide

Introduction

The CAS User's Guide provides general end user-oriented information.

Building on the Globus Toolkit® Grid Security Infrastructure (GSI), CAS allows resource providers to specify course-grained access control policies in terms of communities as a whole, delegating fine-grained access control policy management to the community itself. Resource providers maintain ultimate authority over their resources but are spared day-to-day policy administration tasks (e.g. adding and deleting users, modifying user privileges).

DRAFT

Table of Contents

1. Using CAS 1
    1. How it works 1
    2. Basic procedure for using CAS 1
    3. Using Authorization Service Interface 2
I. CAS User Commands 3
    cas-proxy-init 4
    cas-wrap 7
II. CAS Admin Commands 10
    cas-enroll 11
    cas-remove 15
    cas-action 18
    cas-group-admin 20
    cas-group-add-entry 24
    cas-group-remove-entry 27
    cas-rights-admin 30
2. Troubleshooting 33
    1. Credential Troubleshooting 33
    2. Error Messages 36
3. Information Regarding CAS Licensing 37
Glossary 39

DRAFT

List of Tables

2.1. Credential Errors 34
2.2. WS A&A Authorization Framework Error Messages 36

DRAFT

Chapter 1. Using CAS

1. How it works

1. A CAS server is initiated for a community: a community representative acquires a GSI credential to represent that community as a whole, and then runs a CAS server using that community identity.

2. Resource providers grant privileges to the community. Each resource provider verifies that the holder of the community credential represents that community and that the community's policies are compatible with the resource provider's own policies. Once a trust relationship has been established, the resource provider then grants rights to the community identity, using normal local mechanisms (e.g. grid map files and disk quotas, file system permissions, etc).

3. Community representatives use the CAS to manage the community's trust relationships (e.g., to enroll users and resource providers into the community according to the community's standards) and grant fine-grained access control to resources. The CAS server is also used to manage its own access control policies; for example, community members who have the appropriate privileges may authorize additional community members to manage groups, grant permissions on some or all of the community's resources, etc.

4. When a user wants to access resources served by the CAS, that user makes a request to the CAS server. If the CAS server's database indicates that the user has the appropriate privileges, the CAS issues the user a GSI restricted *proxy credential* with an embedded policy giving the user the right to perform the requested actions.

5. The user then uses the credentials from the CAS to connect to the resource with any normal Globus tool (e.g. GridFTP). The resource then applies its local policy to determine the amount of access granted to the community, and further restricts that access based on the policy in the CAS credentials, This serves to limit the user's privileges to the intersection of those granted by the CAS to the user and those granted by the resource provider to the community.

2. Basic procedure for using CAS

A typical CAS user will use only two CAS-specific commands: **cas-proxy-init**, which contacts a CAS server and obtains a credential, and **cas-wrap**, which wraps a grid-enabled client program, causing that client program to use the credential that was previously generated using **cas-proxy-init**. For example, a day in the life of a CAS user might look something like this:

1. In the morning, the user runs:

   ```
   % grid-proxy-init
   % cas-proxy-init -t tag
   ```

   The first line generates a Globus proxy credential; the second uses that credential to contact the CAS server and retrieve a CAS credential that includes all the permissions granted to the user by the community. The *`tag`* argument can be any string and is used to assign a name for that credential (**cas-wrap** uses this name to locate the credential).

2. Several times throughout the day, the user runs commands that look like:

   ```
   % cas-wrap -t tag grid-enabled-program args
   ```

   This runs the program **grid-enabled-program** with arguments `args`, using the CAS credential that had been created by and assigned the name *`tag`*.

For example:

```
% cas-wrap -t my-community gsincftp myhost.edu
```

looks for a CAS credential with the name "my-community" (e.g., a credential that had been created using "cas-proxy-init -t my-community") and then runs the command "gsincftp myhost.edu", causing the gsincftp program to use that CAS credential to authenticate.

3. At the end of the day, the user runs:

   % cas-wrap -t *tag* grid-proxy-destroy

   to destroy the CAS credential, and:

   ```
   % grid-proxy-destroy
   ```

   to destroy the Globus proxy credential. Or the user might simply let both credentials expire.

3. Using Authorization Service Interface

The user may access resources that are configured to contact a CAS server using the OGSA-AuthZ service interface. This interface allows the resource to pull down the user's rights from the CAS service and enforce the rights. Refer to Section 7, “OGSA-AuthZ Service Interface” for more details.

DRAFT

CAS User Commands

DRAFT

Name

cas-proxy-init -- Generate a CAS proxy

`cas-proxy-init` [common options] [-p *proxyfile* | -t *tag*]

Tool description

The **cas-proxy-init** command contacts a CAS server, obtains an assertion for the user, and embeds it in a credential. This credential can be used to access CAS-enabled services.

Options

Command-specific options

-b *policyFileName*	Generate a CAS credential that includes only those permissions specified in file *policyFileName* (the default is to generate a credential with all the user's permissions). Details about the template of the file is provided here.
-u *tag*	Choose a filename in which to store the CAS credential based on the value *tag*. Cannot be used with the `-p` option.
-w *generatedCredFile*	Specify the file in which to store the CAS credential. Cannot be used with the `-t` option.

Common Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate <*file*>	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor <*file*>	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime <*value*>	Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.

-m, --securityMech *<type>*

Specifies the authentication mechanism. The value *type* can be:

- `msg` for GSI Secure Message, or
- `conv` for GSI Secure Conversation.

-p, --protection *<type>*

Specifies the protection level. *type* can be:

- `sig` for signature, or
- `enc` for encryption.

-s *cas-url*

Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

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

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

Requesting specific permissions from the CAS server

It is possible to request specific permissions from the CAS server using the `-f` option. This option causes **cas-proxy-init** to read a set of requested rights from a file.

This file should contain one or more resource identifiers:

`Resource: ResourceNamespace|ResourceName`

For each resource, there should be one or more action identifiers:

`serviceType action`

For example, if the client needed assertions for "file/read" service/action (permission) on two resources ("ftp://sample1.org" and "ftp://sample3.org", both in "FTPNamespace") but "directory/read" and "directory/write" permissions on the former resource only, the policy file should have the following entries:

```
Resource: FTPNamespace|ftp://sample1.org

      file read

      directory read

      directory write

      Resource: FTPNamespace|ftp://sample3.org

      file read
```

To indicate any resource, the following wildcard notation should be used:

`uri:samlResourceWildcard`

To indicate any action, the following wildcard notation for *`serviceType`* and *`action`* should be used. Note that this should be the first (and clearly the only action) in the list of actions specified. All other actions in the list are ignored and if it is not the first, it is not treated as a wildcard.

`uri:samlActionNSWildcard uri:samlActionWildcard`

For example, if the client needs assertions for all resources and all actions, the policy file should look like:

```
Resource: uri:samlResourceWildcard

        uri:samlActionNSWildcard uri:samlActionWildcard
```

If the client needs assertions for all actions on resource "FTPNamespace|ftp://sample1.org", the policy file should be as follows:

```
Resource: FTPNamespace|ftp://sample1.org

        uri:samlActionNSWildcard uri:samlActionWildcard
```

Name

cas-wrap -- Runs program with CAS credentials

`cas-wrap` [`common options`] [-p *`proxyfile`* | -t *`tag`*]

Tool description

The **cas-wrap** command runs a grid-enabled program, causing it to use previously-generated CAS credentials.

This command invokes the given command with the given argument using the specified previously-generated CAS credential. For example:

```
casAdmin$ cas-wrap -t my-community gsincftp myhost.edu
```

will look for a credential generated by a previous execution of:

```
casAdmin$ cas-proxy-init -t my-community
```

and then set the environment to use that credential while running the command:

```
casAdmin$ gsincftp myhost.edu
```

The second form should be used if **cas-proxy-init** was run with the `-p` option. For example:

```
casAdmin$ cas-wrap -p /path/to/my/cas/credential gsincftp myhost.edu
```

will look for a credential generated by a previous execution of:

```
casAdmin$ cas-proxy-init -p /path/to/my/cas/credential
```

and then set the environment to use that credential while running the command:

```
casAdmin$ gsincftp myhost.edu
```

Options

Command-specific Options

-p *`proxy-file`* Specify the file in which to store the CAS credential. Cannot be used with the `-t` option.

-t *`tag`* Choose a filename in which to store the CAS credential based on the value *`tag`*. Cannot be used with the `-p` option.

Common Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.

-c, --serverCertificate *<file>*

Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.

-debug

Runs the client with debug message traces and error stack traces.

-f, --descriptor *<file>*

Specifies a client security descriptor. Overrides all other security settings.

-help

Prints the usage message for the client.

-l, --contextLifetime *<value>*

Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.

-m, --securityMech *<type>*

Specifies the authentication mechanism. The value *type* can be:

- `msg` for GSI Secure Message, or
- `conv` for GSI Secure Conversation.

-p, --protection *<type>*

Specifies the protection level. *type* can be:

- `sig` for signature, or
- `enc` for encryption.

-s *cas-url*

Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

DRAFT

Usage

Example of using cas-wrap to transfer a file.

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

DRAFT

CAS Admin Commands

DRAFT

Name

cas-enroll -- Enroll a CAS Object

`cas-enroll` [`common options`] trustAnchor *`userGpName nickname authMethod authData`* `cas-enroll` [`common options`] namespace *`userGpName nickname basename comparisonAlg`* `cas-enroll` [`common options`] object *`userGpName objectName namespaceNick`* `cas-enroll` [`common options`] serviceType *`userGpName serviceTypeName`*

Tool description

This command line client is used to enroll a CAS Object, which includes trust anchors, namespaces, objects and service types.

Enrolling Trust Anchors

To enroll a trust anchor, the user must have cas/enroll_trustAnchor permission on that CAS server object (that is, the user must have permission to perform the enroll_trustAnchor action on the CAS service type).

The enroll operation allows the user to choose a user group to which cas/grantAll permission on the enrolled object should be granted. The nickname should be unique across the CAS database and is used to refer to this trust anchor.

To enroll trust anchors:

```
casAdmin$ cas-enroll [common options] trustAnchor userGpName nickname authMethod authData
```

where:

`userGpName` Indicates the user group to which cas/grantAll permission should be granted on this trust anchor entity.

`nickname` Indicates the trust anchor nickname.

`authMethod` Indicates the authentication method used by the trust anchor.

`authData` Indicates the data used for authentication, typically the DN.

Enrolling Namespaces

To enroll a namespace, the user must have cas/enroll_namespace permission (that is, the user must have permission to perform the enroll_namespace action on the cas service type).

The enroll operation allows the user to choose a userGroup to have cas/grantAll permission on the enrolled object. The comparison algorithm specified should be the name of the Comparison class that needs to be used to compare objects that belong to this namespace. The nickname should be unique across the CAS database and is used to refer to this user.

Also, two namespaces are added to the CAS database at boot up time, other than the inherent CAS Namespace:

- `FTPDirectoryTree` uses the WildCardComparison Algorithm and has the base URL set to the current directory.
- `FTPExact` uses the ExactComparison Algorithm and has the base URL set to the current directory.

To enroll namespaces:

```
casAdmin$ cas-enroll [common options] namespace userGpName nickname basename comparisonAlg
```

where:

userGpName Indicates the user group to which cas/grantAll permission should be granted on this trust anchor entity.

nickname Indicates the nickname of the namespace to be unenrolled.

If the trust anchor nickname specified does not exist, an error is *not* thrown. If the unenroll operation is successful, all policy data on that trust anchor is purged.

basename Indicates the base URL for the namespace.

comparisonAlg Indicates the comparison algorithm to be used. Unless the standard comparison algorithms described below are used, the fully qualified name of the class that needs to be used should be given. The class needs to extend from the abstract class `org.globus.cas.impl.service.ObjectComparison`.

The two comparison classes provided as a part of the distribution are:

- `ExactComparison`: This class does a case-sensitive exact comparison of the object names. If *comparisonAlg* in the above method is set to `ExactComparison`, the class in the distribution is loaded and used.
- `WildcardComparison`: This class does wild card matching as described in CAS Simple Policy Language[1]. It assumes that the wild card character is "*" and that the file separator is "/". If *comparisonAlg* in the above method is set to `WildCardComparison`, the class in the distribution is loaded and used.

Enrolling Objects

To enroll an object, the user must have cas/enroll_object permission (that is, the use must have permission to perform the enroll_object action on the cas service type).

The enroll operation allows the user to choose a userGroup to have cas/grantAll permission on the enrolled object. The name of the object and the namespace this object belongs to identify an object in the database and should be unique across the CAS database.

To enroll objects:

casAdmin$ **cas-enroll [*common options*] object *userGpName objectName namespaceNick***

where:

userGpName Indicates the user group to which cas/grantAll permission should be granted on this trust anchor entity.

objectName Indicates the name of the object.

namespaceNick Indicates the nickname of the namespace to which this object belongs.

[1] http://www.globus.org/toolkit/docs/3.2/cas/CAS_policy_language_0.2.pdf

Enrolling Service Types

To enroll a service type, the user must have cas/enroll_serviceType permission (that is, the user must have permission to perform the enroll_serviceType action on the cas service type).

The enroll operation allows the user to choose a userGroup to have cas/grantAll permission on the enrolled service type. The service type name should be unique across the CAS database.

To enroll service types:

```
casAdmin$ cas-enroll [common options] serviceType userGpName serviceTypeName
```

where:

userGpName	Indicates the user group to which cas/grantAll permission should be granted on this trust anchor entity.
serviceTypeName	Indicates the service type name.

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate <*file*>	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor <*file*>	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime <*value*>	Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech <*type*>	Specifies the authentication mechanism. The value *type* can be: • `msg` for GSI Secure Message, or • `conv` for GSI Secure Conversation.
-p, --protection <*type*>	Specifies the protection level. *type* can be: • `sig` for signature, or • `enc` for encryption.
-s *cas-url*	Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *`Host`* and *`Port`* are the host and port where the container with the CAS service is running.

-v — Prints the version number.

-x, --proxyFilename *`<value>`* — Sets the proxy file to use as client credential.

-z *`authorization`* — Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *`<fqdn>`* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

For detailed examples of using this command, see Chapter 6, Example of CAS Server Administration .

DRAFT

Name

cas-remove -- Remove a CAS object from the database

`cas-remove` [`common options`] trustAnchor *`nickname`* `cas-remove` [`common options`] namespace *`nickname`* `cas-remove` [`common options`] object *`objName namespaceNick`* `cas-remove` [`common options`] serviceType *`serviceTypeName`*

Tool description

Removing Trust Anchors

To remove a trust anchor, the user must have cas/remove permission on that trust anchor. The trust anchor must also be unused (that is, there may not be any users in the database that have this trust anchor or it may not be a part of any object group).

To remove trust anchors:

```
casAdmin$ cas-remove [options] trustAnchor nickname
```

where:

`nickname` Indicates the nickname of the trust anchor to be unenrolled.

If the trust anchor nickname specified does not exist, an error is *not* thrown. If the unenroll operation is successful, all policy data on that trust anchor is purged.

Removing Namespaces

To remove a namespace, the user must have cas/remove permission on that namespace. The namespace must also be unused — that is, there may not be any object in the database that belongs to this namespace.

```
casAdmin$ cas-remove [options] namespace nickname
```

where:

`nickname` Indicates the nickname of the namespace to be unenrolled.

If the namespace nickname specified does not exist, an error is *not* thrown. If the remove operation is successful, all policy data on that trust anchor is purged.

Removing Objects

To remove an object the user must have cas/remove permission on that object. The object must also be unused — that is, there may not be any object group in the database that this object belongs to.

```
casAdmin$ cas-remove [options] object objName namespaceNick
```

where:

`objName` Indicates the name of the object to be removed.

`namespaceNick` Indicates the nickname of the namespace to which this object belongs.

DRAFT

If the object specified does not exist, an error is *not* thrown. If the remove operation is successful, all policy data on that object is purged.

Removing Service Types

To remove a service type the user must have cas/remove permission on that service type. The service type must also be unused — that is, there may not be any service type to action mapping.

casAdmin$ **cas-remove [*options*] serviceType *serviceTypeName***

where:

serviceTypeName Indicates the service type name.

If the service type specified does not exist, an error is *not* thrown. If the remove operation is successful, all policy data on that service type is purged.

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous
: Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.

-c, --serverCertificate <*file*>
: Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.

-debug
: Runs the client with debug message traces and error stack traces.

-f, --descriptor <*file*>
: Specifies a client security descriptor. Overrides all other security settings.

-help
: Prints the usage message for the client.

-l, --contextLifetime <*value*>
: Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.

-m, --securityMech <*type*>
: Specifies the authentication mechanism. The value *type* can be:

- msg for GSI Secure Message, or
- conv for GSI Secure Conversation.

-p, --protection <*type*>
: Specifies the protection level. *type* can be:

- sig for signature, or
- enc for encryption.

-s *cas-url*
: Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like http://Host:Port/wsrf/services/CASService, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/`*<fqdn>*, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

DRAFT

Name

cas-action -- Maintains service types

`cas-action` [`common options`] [add | remove] *`serviceTypeName`* *`actionName`*

Tool description

Use the **cas-action** command to add an action mapping to a service type or remove an action mapping from a service type.

To add an action mapping to a service type, the user must have cas/create_group_entry permission on the service type.

To remove a service type action mapping, the user must have cas/delete_group_entry permission on the service type.

If the group member being removed does not exist, an error is *not* thrown.

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate *<file>*	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor *<file>*	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime *<value>*	Sets the lifetime of the client security context. *`value`* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech *<type>*	Specifies the authentication mechanism. The value *`type`* can be: • `msg` for GSI Secure Message, or • `conv` for GSI Secure Conversation.
-p, --protection *<type>*	Specifies the protection level. *`type`* can be: • `sig` for signature, or • `enc` for encryption.
-s *`cas-url`*	Sets the CAS Service instance, where *`cas-url`* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *`Host`* and *`Port`* are the host and port where the container with the CAS service is running.

-v — Prints the version number.

-x, --proxyFilename *`<value>`* — Sets the proxy file to use as client credential.

-z *`authorization`* — Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *`<fqdn>`* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

For an example of using this command, see Section 9, “Adding action mappings”.

DRAFT

Name

cas-group-admin -- Maintains user groups, object groups, or serviceAction groups

`cas-group-admin` [`common options`] [user | object | serviceAction] create *`userGpName`* *`groupName`* `cas-group-admin` [`common options`] [user | object | serviceAction] delete *`groupName`*

Tool description

Use **cas-group-admin** to create or delete user groups, object groups, or serviceAction groups. Note: to add or delete entries to these groups, see [olink to other clients].

Adding user groups

To create a new user group the user must have cas/create_user_group permission (that is, the user must have permission to perform the create_user_group action on the cas service type). The user group name should be unique across the CAS database. The create operation allows the user to choose a user group to have cas/grantAll permission on the created user group. If the user group that is chosen to have cas/grantAll permission is the new group created, then the user making this request is added to the new group.

To add a user group:

```
casAdmin$ cas-group-admin [common options] user create userGpName groupName
```

where:

`userGpName` Indicates the user group to which cas/grantAll permission should be granted on this trust anchor entity.

`groupName` Indicates the name of the user group being created.

Deleting user groups

To delete a user group, the user must have cas/delete_user_group entry permission on that user group. The group must be empty and also must not be referenced from other entities in the database (for example, it should not be a member of some object group).

If the user group specified does not exist, an error is *not* thrown. If the delete operation is successful, all policy data on that user group is purged.

```
casAdmin$ cas-group-admin [common options] user delete groupName
```

where:

`groupName` Indicates the name of the user group to be deleted.

Creating An Object Group

To create a new object group, the user must have cas/create_object_group permission (that is, the user must have permission to perform the create_object_group action on the CAS service type). The object group name should be unique across the CAS database. The create operation allows the user to choose a user group to have cas/grantAll permission on the created object group.

```
casAdmin$ cas-group-admin [common options] object create userGpName groupName
```

where:

userGpName Indicates the user group to which cas/grantAll permission should be granted on this object group.

groupName Indicates the object group name.

Deleting An Object Group

To delete an object group, the user must have cas/delete_user_group entry permission on that object group. The group must be empty.

If the object group specified does not exist, an error is *not* thrown. If the delete operation is successful, all policy data on that object group is purged.

```
casAdmin$ cas-group-admin [common options] object delete groupName
```

where:

groupName The name of the object group to be deleted.

Creating A Service/Action Group

To create a new service/action group, the user must have cas/create_serviceAction_group permission (that is, the user must have permission to perform the create_serviceAction_group action on the CAS service type). The serviceAction group name should be unique across the CAS database. The create operation allows the user to choose a user group to have cas/grantAll permission on the created serviceAction group.

```
casAdmin$ cas-group-admin [common options] serviceAction create userGpName groupName
```

where:

userGp-Name Indicates the user group to which cas/grantAll permission should be granted on this service/action group.

groupName Indicates the name of the service/action group being created.

Deleting A Service/Action Group

To delete a service/action group, the user must have cas/delete_user_group entry permission on that service/action group. The group must be empty and also must not be referenced from any other entity in the database. For example, it should not be a member of some object group.

If the service/action group specified does not exist, an error is *not* thrown. If the delete operation is successful, all policy data on that service/action group is purged.

```
casAdmin$ cas-group-admin [common options] serviceAction delete groupName
```

where:

gp Indicates the name of the service/action group to be deleted.
Na

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate *<file>*	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor *<file>*	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime *<value>*	Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech *<type>*	Specifies the authentication mechanism. The value *type* can be: • `msg` for GSI Secure Message, or • `conv` for GSI Secure Conversation.
-p, --protection *<type>*	Specifies the protection level. *type* can be: • `sig` for signature, or • `enc` for encryption.
-s *cas-url*	Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here. The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.
-v	Prints the version number.
-x, --proxyFilename *<value>*	Sets the proxy file to use as client credential.
-z *authorization*	Specifies the type of authorization used, such as `self` or `host`. If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value. Alternatively, an environment variable can be set as shown here. If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

DRAFT

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

For examples of using this command, see Chapter 6, Example of CAS Server Administration .

DRAFT

Name

cas-group-add-entry -- Adds CAS objects to CAS groups

`cas-group-add-entry` [`common options`] user *`groupName nickname`* `cas-group-add-entry` [`common options`] object *`groupName objectSpecDesc objectSpec`* `cas-group-add-entry` [`common options`] serviceAction *`groupName serviceTypeName actionName`*

Tool description

Use **cas-group-add-entry** to add users to a user group, objects to an object group, or service/actions to service/action groups. Note: to add or delete groups, see [olink to other clients].

Adding Member To A User Group

To add a user to a user group, the user must have cas/add_group_entry permission on that particular user group. Only user nicknames that exist in the CAS database can be valid members.

```
casAdmin$ cas-group-add-entry [common options] user groupName nickname
```

where:

groupName Indicates the user group name to which the member needs to be added.

nickname Indicates the nickname of the user to be added to this group.

Adding Member To An Object Group

To add a member (an object group can have the following CasObjects as members: object, user, user group, service type, namespace or trust anchor) to an object group, the user must have cas/add_group_entry permission on that particular object group.

```
casAdmin$ cas-group-add-entry [common options] object groupName objectSpecDesc objectSpec
```

where:

groupName Indicates the object group name to which the member needs to be added.

objectSpecDesc Indicates the type of CasObject. Can be one of the following options:

- `trustAnchor`
- `user`
- `userGroup`
- `object`
- `namespace`
- `serviceType`

objectNameSpace Indicates the identifier for the CasObject the user is adding. Can be one of the following:

- `nickname` if adding a trustAnchor or user
- `groupName` if adding a userGroup
- `objectNamespace` objectName if adding an object
- `nickname` if adding a namespace
- `serviceTypeName` if adding a serviceType

Adding Service/Action To A Service/Action Group

To add a service/action to a serviceAction group, the user must have cas/add_group_entry permission on that particular serviceAction group (that is, the user must have permission to perform add_group_entry action on that service action group).

casAdmin$ **cas-group-add-entry [*common options*] serviceAction *groupName serviceTypeName act***

where:

groupNameAte Indicates the service/action group to which the service/action needs to be added.

serviceTypeNameAte Indicates the service type name part of the mapping to be added to the group.

actionNameAte Indicates the action name part of the mapping to be added to the group.

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate <*file*>	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor <*file*>	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime <*value*>	Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.
-m, --securityMech <*type*>	Specifies the authentication mechanism. The value *type* can be:

- `msg` for GSI Secure Message, or
- `conv` for GSI Secure Conversation.

-p, --protection *<type>*

Specifies the protection level. *type* can be:

- `sig` for signature, or
- `enc` for encryption.

-s *cas-url*

Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

For examples of using this command, see Chapter 6, Example of CAS Server Administration .

DRAFT

Name

cas-group-remove-entry -- Removing CAS objects from CAS groups

`cas-group-remove-entry` [`common options`] user *`groupName nickname`* `cas-group-remove-entry` [`common options`] object *`groupName objectSpec objectSpecDesc`* `cas-group-remove-entry` [`common options`] serviceAction *`groupName serviceTypeName actionName`*

Tool description

Use **cas-group-remove-entry** to remove users from a user group, objects from an object group, or service/actions from a service/action group. Note: to add or delete groups, see [olink to other clients].

Removing User From A User Group

To remove a user from a user group, the user must have cas/remove_group_entry permission on that particular user group.

If the group member being removed does not exist, an error is *not* thrown.

```
casAdmin$ cas-group-remove-entry [common options] user groupName nickname
```

where:

groupName Indicates the user group name from which the member needs to be removed.

nickname Indicates the nickname of the user to be removed from this group.

Removing Member From An Object Group

To remove an object from an object group the user must have cas/remove_group_entry permission on that particular object group:

If the group member being removed does not exist, an error is *not* thrown.

```
casAdmin$ cas-group-remove-entry [common options] object groupName objectSpec objectSpecDe
```

where:

groupName Indicates the object group name from which the member needs to be removed.

objectSpecDesc Indicates the type of CasObject. Can be one of the following options:

- `trustAnchor`
- `user`
- `userGroup`
- `object`
- `namespace`

- `serviceType`

object Indicates the identifier for the CasObject the user is adding. Can be one of the following:

specifier

- `nickname` if adding a trustAnchor or user
- `groupName` if adding a userGroup
- `objectNamespace` objectName if adding an object
- `nickname` if adding a namespace
- `serviceTypeName` if adding a serviceType

Removing A Service/Action From A Service/Action Group

To remove a service/action from a service/action group, the user must have cas/remove_group_entry permission on that particular service/action group.

If the action being removed does not exist, an error is *not* thrown.

casAdmin$ **cas-group-remove-entry [*common options*] serviceAction *groupName serviceTypeName***

where:

group name Indicates the serviceAction group name from which the service/action needs to be removed.

service type name Indicates the service type name part of the mapping to be removed from the group.

action name Indicates the action name part of the mapping to be removed from the group.

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate *<file>*	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor *<file>*	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.

-l, --contextLifetime *<value>*

Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.

-m, --securityMech *<type>*

Specifies the authentication mechanism. The value *type* can be:

- `msg` for GSI Secure Message, or
- `conv` for GSI Secure Conversation.

-p, --protection *<type>*

Specifies the protection level. *type* can be:

- `sig` for signature, or
- `enc` for encryption.

-s *cas-url*

Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

DRAFT

Name

cas-rights-admin -- Granting or revoking permissions

`cas-rights-admin` [`common options`] [grant | revoke] *`userGroupName objectSpecDesc objectSpec actionSpecDesc actionSpec`*

Tool description

Use **cas-rights-admin** to grant or revoke rights.

Granting Permissions To A User Group On An Object/Object Group

The user may grant permissions to a user group on an object or object group to perform a service action or service action group (that is, to perform any action that is a member of the service action group to which permission is granted), provided the user has both:

- cas/grant permission on the object or object group, and
- permission to perform the service action or service action group on the object or object group.

```
casAdmin$ cas-rights-admin [common options] grant userGroupName objectSpecDesc objectSpec
```

where:

usergroupname Indicates the user group to be granted permission.

objectspec Indicates the identifier for the object or object group.

objectspecdesc Indicates the type:

- `object`
- `objectGroup`

actionspec Indicates the identifier for action or action group.

actionspecdesc Indicates the type:

- `serviceAction`
- `serviceActionGp`

Revoking A Policy In The CAS Database

The user may revoke a policy in the CAS database if the user has cas/revoke permission on the object or object group on which the policy is defined.

```
casAdmin$ cas-rights-admin [common options] revoke userGroupName  objectSpecDesc objectSpe
```

where:

user group name Indicates the user group for which you want to grant permission.

object type Indicates the type of CasObject. Can be one of the following:

- `trustAnchor`
- `user`
- `userGroup`
- `object`
- `namespace`
- `serviceType`
- `userGroup`

object id Indicates the identifier for the object or object group.

action id Indicates the identifier for the action or action group.

action type Indicates the type (serviceAction or serviceActionGp).

Options

Important

If you have an asterisk (*) in your command, you might need to escape it with a backslash (\ ).

-a, --anonymous	Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.
-c, --serverCertificate <*file*>	Specifies the server's *certificate* file used for encryption. Only needed for the GSI Secure Message authentication mechanism.
-debug	Runs the client with debug message traces and error stack traces.
-f, --descriptor <*file*>	Specifies a client security descriptor. Overrides all other security settings.
-help	Prints the usage message for the client.
-l, --contextLifetime <*value*>	Sets the lifetime of the client security context. *value* is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.

DRAFT

-m, --securityMech *<type>*

Specifies the authentication mechanism. The value *type* can be:

- `msg` for GSI Secure Message, or
- `conv` for GSI Secure Conversation.

-p, --protection *<type>*

Specifies the protection level. *type* can be:

- `sig` for signature, or
- `enc` for encryption.

-s *cas-url*

Sets the CAS Service instance, where *cas-url* is the URL of the CAS service instance. Alternatively, an environment variable can be set as shown here.

The instance URL typically looks like `http://Host:Port/wsrf/services/CASService`, where *Host* and *Port* are the host and port where the container with the CAS service is running.

-v

Prints the version number.

-x, --proxyFilename *<value>*

Sets the proxy file to use as client credential.

-z *authorization*

Specifies the type of authorization used, such as `self` or `host`.

If you cannot use a standard method for authorization, you can use the specific CAS server's identity as the value.

Alternatively, an environment variable can be set as shown here.

If none of the above are set, host authorization is done by default and the expected server credential is `cas/<fqdn>`, where *<fqdn>* is the fully qualified domain name of the host on which the CAS service is up.

Note

If the service being contacted is using GSI Secure Transport , then the container credentials configured for the service will be used, even if service/resource level credentials are configured. Hence authorization needs to be done based on the DN of the container credentials.

Usage

For an example of using this command, see Chapter 6, Example of CAS Server Administration .

Chapter 2. Troubleshooting

For a list of common errors in GT, see Error Codes.

1. Credential Troubleshooting

1.1. 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.

DRAFT

Table 2.1. Credential Errors

Error Code	Definition	Possible Solutions
`Your proxy credential may have expired`	Your proxy credential may have expired.	Use grid-proxy-info to check whether the proxy credential has actually expired. If it has, generate a new proxy with grid-proxy-init.
`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 grid-cert-info 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 grid-proxy-destroy) and creating a new one (with grid-proxy-init). **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), grid-proxy-init 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 4.2.0 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 4.2.0 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) .

1.2. Some tools to validate certificate setup

1.2.1. Check that the user certificate is valid

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

1.2.2. 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.

1.2.3. 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
```

2. Error Messages

Table 2.2. WS A&A Authorization Framework Error Messages

Error Code	Definition

`org.globus.cas.impl.databaseAccess.CasDBException, connection refused`	If the CAS service fails with following error:
	
	`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.

CAS clients fail with `org.globus.cas.impl.databaseAccess.CasDBException`	If CAS clients fail with database permission exceptions similar to:
	
	`[Caused by: ERROR:`
	`permission denied for relation service_type_action ]; nested exceptio`
	`is:org.globus.cas.impl.databaseAccess.CasDBException:`
	
	, then there is something wrong with user permissions on the database.
	
	Note
	
	This is a specific instance of an error for the relation *service_type_action*. This error could be raised on any rel

DRAFT

Chapter 3. Information Regarding CAS Licensing

This version of CAS uses the OASIS Security Assertion Markup Language (SAML) standard. Users should be aware that RSA Security has identified four patents it believes could be relevant to implementing certain operational modes of the SAML specifications. Previously, RSA required that users would execute a royalty-free reciprocal license to the RSA patents, and the Globus Alliance had established a license agreement with RSA covering usage of SAML in the Globus Toolkit.

On May 11, 2006, however, RSA issued the following statement:

```
RSA Intellectual Property Rights Statement

    In previous correspondence dated December 6, 2004, January 20, 2003
    and April 22, 2002, RSA Security Inc. ("RSA") disclosed that it is
    the assignee of U.S. Patent Nos. 6,085,320 and 6,189,098, both
    entitled "Client/Server Protocol for Proving Authenticity" and U.S.
    Patent Nos. 5,922,074 and 6,249,873, both entitled "Method of and
    Apparatus for Providing Secure Distributed Directory Services and
    Public Key Infrastructure" (collectively, the "RSA Patents"). At
    that time, RSA believed that these four patents could be relevant to
    practicing certain operational modes of the OASIS Security Assertion
    Markup Language ("SAML") specifications. In the correspondence, RSA
    offered to grant non-exclusive, royalty-free licenses on a
    non-discriminatory basis for the RSA Patents.

    In the interest of encouraging deployment of SAML-based
    technologies, RSA hereby covenants, free of any royalty, that it
    will not assert any claims in the RSA Patents which may be essential
    to the SAML standard v1.0, 1.1 and 2.0 (hereinafter "NECESSARY
    CLAIMS") against any other entity with respect to any implementation
    conforming to the SAML standard v1.0, 1.1 and/or 2.0. This covenant
    shall become null and void with respect to any entity that asserts,
    either directly or indirectly (e.g., through an affiliate), any
    patent claims or threatens or initiates any patent infringement suit
    against RSA and/or its subsidiaries or affiliates. The revocation of
    the covenant shall extend to all prior use by the entity asserting
    the claim.

    RSA will continue to honor existing license agreements for the RSA
    Patents and will continue to offer as an option to interested third
    parties the same licensing arrangement described in our previous
    correspondence. (The license agreement, along with instructions for
    obtaining and completing the license, are available on RSA's website
    www.rsasecurity.com)
```

Please refer to published statement[1]

[1] http://xml.coverpages.org/RSA-SAML-NonAssert.html#statement

Important

Users should always check with their legal counsel about the interpretation of the statement, but the statement may indicate that for most SAML-users no execution of any license will be required.

DRAFT

Glossary

some terms not in the docs but wanted in glossary: scheduler

C

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

P

proxy credentials — The combination of a proxy certificate and its corresponding private key. GSI typically stores proxy credentials in `/tmp/x509up_u<uid>` , where <uid> is the user id of the proxy owner.

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.

DRAFT