EGEE User’s Guide
VOMS Core Services

Abstract: This user's guide explains how to use Service VOMS Core Services.
EGEE USER’S GUIDE

Delivery Slip

<table>
<thead>
<tr>
<th>Name</th>
<th>Partner</th>
<th>Date</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>Vincenzo Ciaschini et al.</td>
<td>INFN</td>
<td>Mar 9, 2005</td>
</tr>
<tr>
<td>Reviewed by</td>
<td>Anthony Wilson</td>
<td>RAL</td>
<td>Mar 11, 2005</td>
</tr>
<tr>
<td>Modified by</td>
<td>Vincenzo Ciaschini</td>
<td>INFN</td>
<td>Sep 6, 2006</td>
</tr>
<tr>
<td>Approved by</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Document Change Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Comment</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>Mar 9, 2005</td>
<td></td>
<td>V. Ciaschini</td>
</tr>
<tr>
<td>Reviewer’s comments</td>
<td>Mar 25, 2005</td>
<td>comments reflected</td>
<td>V. Ciaschini</td>
</tr>
<tr>
<td>Updated version</td>
<td>Sep 06, 2006</td>
<td>Updated to reflect changes up to VOMS 1.7.8</td>
<td>V. Ciaschini</td>
</tr>
</tbody>
</table>

Document Change Record

<table>
<thead>
<tr>
<th>Issue</th>
<th>Item</th>
<th>Reason for Change</th>
</tr>
</thead>
</table>


EGEE (“Enabling Grids for E-science in Europe”) is a project funded by the European Union. For more information on the project, its partners and contributors please see http://www.eu-egee.org.

You are permitted to copy and distribute verbatim copies of this document containing this copyright notice, but modifying this document is not allowed. You are permitted to copy this document in whole or in part into other documents if you attach the following reference to the copied elements: “Copyright ©2004. Members of the EGEE Collaboration. http://www.eu-egee.org”

The information contained in this document represents the views of EGEE as of the date they are published. EGEE does not guarantee that any information contained herein is error-free, or up to date.

EGEE MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.
CONTENTS

I  INTRODUCTION .............................................................. 4

1  INTRODUCTION ............................................................. 5
   1.1  SERVICE ARCHITECTURE ............................................ 5
   1.2  INTERACTIONS WITH OTHER SERVICES ............................ 5

2  BASIC CONCEPTS ............................................................ 5
   2.1  ATTRIBUTE CERTIFICATE ........................................... 5
   2.2  GROUPS, ROLES AND GENERIC ATTRIBUTES ...................... 5
       2.2.1  FQAN .......................................................... 6

II QUICKSTART GUIDE ....................................................... 7

3  COMPATIBILITY .................................................................. 8

4  INSTALLING AND RUNNING A VOMS SERVER ............................ 8

5  USING THE CLIENT COMMAND ............................................. 10
   5.1  GENERAL SETUP ..................................................... 10
       5.1.1  VOMSDIR SETUP ............................................. 10
       5.1.2  VOMSES FILES ............................................... 10
       5.1.3  USER CREDENTIALS .......................................... 11
   5.2  USING THE CLIENT COMMANDS .................................... 11
       5.2.1  VOMS-PROXY-INIT .......................................... 11
       5.2.2  VOMS-PROXY-INFO .......................................... 13
       5.2.3  VOMS-PROXY-DESTROY .................................... 13
       5.2.4  VOMS-PROXY-LIST .......................................... 14

III REFERENCE GUIDE .......................................................... 15

6  VOMSD .......................................................................... 16
   6.1  SOFTWARE REQUIREMENTS ......................................... 16
   6.2  COMMAND LINE INTERFACE ..................................... 16
       6.2.1  LOGGING OPTIONS ........................................... 18
   6.3  SERVER SETUP AND MAINTENANCE ............................... 18
   6.4  EXAMPLE CONFIGURATION ......................................... 19

7  VOMS-PROXY-INIT .......................................................... 19

8  VOMS-PROXY-INFO .......................................................... 23

9  VOMS-PROXY-DESTROY .................................................... 24
10 VOMS-PROXY-LIST

IV APPLICATION PROGRAM INTERFACES .......................... 28

11 C++ API ....................................................... 29

11.1 THE DATA STRUCTURE ........................................ 29

11.1.1 GROUP .................................................... 29
11.1.2 ROLE ..................................................... 30
11.1.3 CAP ....................................................... 30

11.2 THE VOMS STRUCTURE ......................................... 30

11.2.1 VERSION .................................................... 31
11.2.2 SIGLEN .................................................... 31
11.2.3 SIGNATURE ................................................ 31
11.2.4 USER ....................................................... 31
11.2.5 USERCA .................................................... 31
11.2.6 SERVER ..................................................... 31
11.2.7 SERVERCA ................................................ 31
11.2.8 VONAME .................................................... 31
11.2.9 URI ........................................................ 32
11.2.10 DATE1, DATE2 ............................................. 32
11.2.11 TYPE ...................................................... 32
11.2.12 STD ....................................................... 33
11.2.13 CUSTOM ................................................... 33
11.2.14 FQAN ....................................................... 33
11.2.15 SERIAL ..................................................... 33
11.2.16 GETAC() ................................................... 33
11.2.17 GETATTRIBUTES() ....................................... 33
11.2.18 VOMS::VOMS() ........................................... 34
11.2.19 VOMS::VOMS(CONST VOMS &) ............................ 34
11.2.20 VOMS::OPERATOR=(CONST VOMS &) .................... 34

11.3 VOMSDATA .................................................... 34

11.3.1 ERROR ...................................................... 35
11.3.2 DATA ....................................................... 36
11.3.3 VOMSDATA::VOMSDATA(STD::STRING VOMS_DIR, STD::STRING CERT_DIR) 36

11.3.4 BOOL VOMSDATA::LOADSYSTEMCONTACTS(STD::STRING DIR) .... 36
11.3.5 BOOL VOMSDATA::LOADUSERCONTACTS(STD::STRING DIR) ......... 37
11.3.6 STD::VECTOR<CONTACTDATA> VOMSDATA::FINDBYALIAS(STD::STRING ALIAS) ........................................ 37
11.3.7 VOID VOMSDATA::ORDER(STD::STRING ATTRIBUTE) ........... 37
11.3.8 VOID VOMSDATA::RESETORDER(VOID) ..................... 38
11.3.9 VOID VOMSDATA::ADDTARGET(STD::STRING TARGET) ........ 38
11.3.10 STD::VECTOR<STD::STRING> VOMSDATA::LISTTARGETS(VOID) ...... 38
11.3.11 VOID VOMSDATA::RESETTARGETS(VOID) ...................... 38
11.3.12 STD::STRING VOMSDATA::SERVERERRORS() ..................... 39
11.3.13 VOID VOMSDATA::SETVERIFICATIONTYPE(VERIFY_TYPE HOW) ...... 39
11.3.14 VOID VOMSDATA::SETLIFETIME(INT LIFETIME) ................. 40
11.3.15 BOOL VOMSDATA::RETRIEVE*() ............................... 40
11.3.16 BOOL VOMSDATA::CONTACT(STD::STRING HOSTNAME, INT PORT, STD::STRING SERVSUBJECT, STD::STRING COMMAND) ............. 41
11.3.17 BOOL CONTACTRAW(STD::STRING HOSTNAME, INT PORT, STD::STRING SERVSUBJECT, STD::STRING COMMAND, STD::STRING &RAW, INT &VERSION) .......... 42
11.3.18 BOOL EXPORT(STD::STRING &DATA) .......................... 43
11.3.19 BOOL IMPORT(STD::STRING BUFFER) ......................... 43
11.3.20 BOOL VOMSDATA::DEFAULTDATA(VOMS &D) .................... 44
11.3.21 STD::STRING VOMSDATA::ERRORMESSAGE(VOID) .............. 44

12 C API
12.1 THE DATA STRUCTURE ............................................. 44
12.1.1 GROUP .......................................................... 45
12.1.2 ROLE ........................................................... 45
12.1.3 CAP ............................................................. 45
12.2 THE VOMS STRUCTURE ............................................. 45
12.2.1 VERSION ........................................................ 46
12.2.2 SIGLEN .......................................................... 46
12.2.3 USER ............................................................ 46
12.2.4 USERCA .......................................................... 46
12.2.5 SERVER .......................................................... 46
12.2.6 SERVERCA ........................................................ 46
12.2.7 VONAME .......................................................... 47
12.2.8 URI ............................................................... 47
12.2.9 DATE1, DATE2 .................................................... 47
12.2.10 TYPE ............................................................ 47
12.2.11 STD ............................................................. 48
12.2.12 CUSTOM ......................................................... 48
12.2.13 FQAN ............................................................ 48
12.2.14 VOMSDATA ....................................................... 48
12.2.15 DATA ............................................................. 48
12.2.16 WORKVO, VOLEN ............................................... 49
12.2.17 EXTRA_DATA, EXTRALEN ...................................... 49
12.2.18 CDIR, VDIR ..................................................... 49
12.3 FUNCTIONS

12.3.1 GENERALITIES

12.3.2 STRUCT CONTACTDATA **VOMS_FINDBY*(STRUCT VOMSDATA *VD, CHAR *ALIAS, CHAR *SYSTEM, CHAR *USER, INT *ERROR)

12.3.3 VOID VOMS_DELETECONTACTS(STRUCT CONTACTDATA **LIST)

12.3.4 STRUCT VOMSDATA *VOMS_INIT(CHAR *VOMS, CHAR *CERT)

12.3.5 STRUCT VOMS *VOMS_COPY(STRUCT VOMS *, INT *ERROR)

12.3.6 STRUCT VOMSDATA *VOMS_COPYALL(STRUCT VOMSDATA *VD, INT *ERROR)

12.3.7 VOID VOMS_DELETE(STRUCT VOMS *V)

12.3.8 INT VOMS_ADDTARGET(STRUCT VOMSDATA *VD, CHAR *TARGET, INT *ERROR)

12.3.9 VOID VOMS_FREETARGETS(STRUCT VOMSDATA *VD, INT *ERROR)

12.3.10 CHAR *VOMS_LISTTARGETS(STRUCT VOMSDATA *VD, INT *ERROR)

12.3.11 INT VOMSDATA_SETVERIFICATIONTYPE(INT TYPE, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.12 INT VOMS_SETLIFETIME(INT LENGTH, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.13 VOID VOMS_DESTROY(STRUCT VOMSDATA *VD)

12.3.14 INT VOMS_ORDERING(CHAR *ORDER, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.15 INT VOMS_RESETORDER(STRUCT VOMSDATA *CD, INT *ERROR)

12.3.16 INT VOMS_CONTACT(CHAR *HOSTNAME, INT PORT, CHAR *SERVSUBJECT, CHAR *COMMAND, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.17 INT VOMS_CONTACTRAW(CHAR *HOSTNAME, INT PORT, CHAR *SERVSUBJECT, CHAR *COMMAND, VOID **DATA, INT *DATALEN, INT *VERSION, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.18 INT VOMS_RETRIEVE*(X509 *CERT, STACK OF(X509) *CHAIN, INT HOW, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.19 INT VOMS_IMPORT(CHAR *BUFFER, INT BUFLEN, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.20 INT VOMS_EXPORT(CHAR **BUFFER, INT *BUFLEN, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.21 STRUCT VOMS *VOMS_DEFAULTDATA(STRUCT VOMSDATA *VD, INT *ERROR)

12.3.22 CHAR *VOMS_ERRORMESSAGE(STRUCT VOMSDATA *VD, INT *ERROR, CHAR *BUFFER, INT LEN)

12.3.23 INT VOMS_GETATTRIBUTESOURCEHANDLE(STRUCT VOMS *V, INT NUM, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.24 VOMS_GETATTRIBUTESOURCESNUMBER(STRUCT VOMS *V, STRUCT VOMSDATA *VD, INT *ERROR)

12.3.25 VOMS_GETATTRIBUTTEGRANTOR(STRUCT VOMS *V, INT HANDLE, STRUCT VOMSDATA *VD, INT *ERROR)
12.3.26 VOMS_GETATTRIBUTESNUMBER(STRUCT VOMS *V, INT HANDLE, STRUCT VOMSDATA *VD, INT *ERROR) .......................... 60
12.3.27 VOMS_GETATTRIBUTE(STRUCT VOMS *V, INT HANDLE, INT NUM, STRUCT ATTRIBUTE *A, STRUCT VOMDATA *VD, INT *ERROR) .............. 60

13 JAVA APIS .................................................. 61
 13.1 CLASS BASICVOMSTRUSTSTORE .......................... 61
    13.1.1 DECLARATION ........................................ 61
    13.1.2 FIELD SUMMARY .................................... 62
    13.1.3 CONSTRUCTOR SUMMARY .............................. 62
    13.1.4 METHOD SUMMARY ................................... 62
    13.1.5 FIELDS ................................................. 62
    13.1.6 CONSTRUCTORS ..................................... 62
    13.1.7 METHODS ............................................. 62
 13.2 CLASS FQAN .............................................. 63
    13.2.1 DECLARATION ........................................ 63
    13.2.2 CONSTRUCTOR SUMMARY .............................. 63
    13.2.3 METHOD SUMMARY ................................... 63
    13.2.4 CONSTRUCTORS ..................................... 64
    13.2.5 METHODS ............................................. 64
 13.3 CLASS LSCFILE .......................................... 64
    13.3.1 DECLARATION ........................................ 64
    13.3.2 CONSTRUCTOR SUMMARY .............................. 64
    13.3.3 METHOD SUMMARY ................................... 64
    13.3.4 CONSTRUCTORS ..................................... 65
    13.3.5 METHODS ............................................. 65
 13.4 CLASS PKISTORE ......................................... 65
    13.4.1 DECLARATION ........................................ 65
    13.4.2 FIELD SUMMARY .................................... 65
    13.4.3 CONSTRUCTOR SUMMARY .............................. 65
    13.4.4 METHOD SUMMARY ................................... 66
    13.4.5 FIELDS ................................................. 66
    13.4.6 CONSTRUCTORS ..................................... 66
    13.4.7 METHODS ............................................. 67
 13.5 CLASS PKIUTILS ......................................... 69
    13.5.1 DECLARATION ........................................ 69
    13.5.2 CONSTRUCTOR SUMMARY .............................. 69
    13.5.3 METHOD SUMMARY ................................... 69
    13.5.4 CONSTRUCTORS ..................................... 70
    13.5.5 METHODS ............................................. 70
13.6 CLASS PKIVERIFIER ............................................. 75
  13.6.1 DECLARATION .............................................. 75
  13.6.2 FIELD SUMMARY .......................................... 75
  13.6.3 CONSTRUCTOR SUMMARY ................................... 75
  13.6.4 METHOD SUMMARY .......................................... 75
  13.6.5 FIELDS .................................................. 75
  13.6.6 CONSTRUCTORS ............................................ 76
  13.6.7 METHODS ................................................ 77
13.7 CLASS SIGNINGPOLICY ........................................... 77
  13.7.1 DECLARATION .............................................. 77
  13.7.2 CONSTRUCTOR SUMMARY ................................... 78
  13.7.3 METHOD SUMMARY .......................................... 78
  13.7.4 CONSTRUCTORS ............................................ 78
  13.7.5 METHODS ................................................ 78
13.8 CLASS VOMSATTRIBUTE ......................................... 79
  13.8.1 DECLARATION .............................................. 80
  13.8.2 CONSTRUCTOR SUMMARY ................................... 80
  13.8.3 METHOD SUMMARY .......................................... 80
  13.8.4 CONSTRUCTORS ............................................ 80
  13.8.5 METHODS ................................................ 81
13.9 CLASS VOMSVVALIDATOR ......................................... 85
  13.9.1 DECLARATION .............................................. 85
  13.9.2 FIELD SUMMARY .......................................... 85
  13.9.3 CONSTRUCTOR SUMMARY ................................... 85
  13.9.4 METHOD SUMMARY .......................................... 86
  13.9.5 FIELDS .................................................. 86
  13.9.6 CONSTRUCTORS ............................................ 86
  13.9.7 METHODS ................................................ 87
13.10 CLASS VOMSVVALIDATOR.FQANTREE ................................ 90
  13.10.1 DECLARATION .............................................. 90
  13.10.2 CONSTRUCTOR SUMMARY ................................... 90
  13.10.3 METHOD SUMMARY .......................................... 90
  13.10.4 CONSTRUCTORS ............................................ 90
  13.10.5 METHODS ................................................ 91
13.11 INTERFACE ACRUSTSTORE ....................................... 92
  13.11.1 DECLARATION .............................................. 92
  13.11.2 ALL KNOWN SUBINTERFACES ................................. 92
  13.11.3 ALL CLASSES KNOWN TO IMPLEMENT INTERFACE .......... 92
  13.11.4 METHOD SUMMARY .......................................... 92
13.11 METHODS ......................................................... 93
13.12 INTERFACE VOMSTRUSTSTORE ................................. 93
  13.12.1 DECLARATION ............................................. 93
  13.12.2 ALL KNOWN SUBINTERFACES .............................. 93
  13.12.3 ALL CLASSES KNOWN TO IMPLEMENT INTERFACE ......... 93
  13.12.4 METHOD SUMMARY ....................................... 93
  13.12.5 METHODS .................................................. 93
13.13 CLASS ACCERTS .................................................. 94
  13.13.1 DECLARATION ............................................. 94
  13.13.2 CONSTRUCTOR SUMMARY ................................ 94
  13.13.3 METHOD SUMMARY ....................................... 94
  13.13.4 CONSTRUCTORS .......................................... 94
  13.13.5 METHODS .................................................. 95
13.14 CLASS ACGENERATOR ........................................... 95
  13.14.1 DECLARATION ............................................. 96
  13.14.2 CONSTRUCTOR SUMMARY ................................ 96
  13.14.3 METHOD SUMMARY ....................................... 96
  13.14.4 CONSTRUCTORS .......................................... 96
  13.14.5 METHODS .................................................. 96
13.15 CLASS ACTARGET ................................................ 97
  13.15.1 DECLARATION ............................................. 98
  13.15.2 CONSTRUCTOR SUMMARY ................................ 98
  13.15.3 METHOD SUMMARY ....................................... 98
  13.15.4 CONSTRUCTORS .......................................... 98
  13.15.5 METHODS .................................................. 98
13.16 CLASS ACTARGETS ............................................. 100
  13.16.1 DECLARATION ............................................. 101
  13.16.2 CONSTRUCTOR SUMMARY ................................ 101
  13.16.3 METHOD SUMMARY ....................................... 101
  13.16.4 CONSTRUCTORS .......................................... 101
  13.16.5 METHODS .................................................. 101
13.17 CLASS ACVALIDATOR .......................................... 102
  13.17.1 DECLARATION ............................................. 102
  13.17.2 FIELD SUMMARY ....................................... 102
  13.17.3 CONSTRUCTOR SUMMARY ................................ 103
  13.17.4 METHOD SUMMARY ....................................... 103
  13.17.5 FIELDS ..................................................... 103
  13.17.6 CONSTRUCTORS .......................................... 103
  13.17.7 METHODS .................................................. 103
13.23.5 METHODS ................................................. 115
13.24 CLASS HOLDER ........................................... 116
  13.24.1 DECLARATION ........................................ 116
  13.24.2 CONSTRUCTOR SUMMARY ........................... 116
  13.24.3 METHOD SUMMARY .................................. 116
  13.24.4 CONSTRUCTORS .................................... 117
  13.24.5 METHODS ........................................... 117
13.25 CLASS IETFATTRSYNTAX ................................. 117
  13.25.1 DECLARATION ........................................ 117
  13.25.2 FIELD SUMMARY .................................... 117
  13.25.3 CONSTRUCTOR SUMMARY ............................ 118
  13.25.4 METHOD SUMMARY .................................. 118
  13.25.5 FIELDS .............................................. 118
  13.25.6 CONSTRUCTORS .................................... 118
  13.25.7 METHODS ........................................... 118
13.26 CLASS OBJECTDIGESTINFO ............................... 118
  13.26.1 DECLARATION ........................................ 118
  13.26.2 CONSTRUCTOR SUMMARY ............................ 118
  13.26.3 METHOD SUMMARY .................................. 119
  13.26.4 CONSTRUCTORS .................................... 119
  13.26.5 METHODS ........................................... 119
13.27 CLASS UTIL ............................................... 119
  13.27.1 DECLARATION ........................................ 119
  13.27.2 CONSTRUCTOR SUMMARY ............................ 120
  13.27.3 METHOD SUMMARY .................................. 120
  13.27.4 CONSTRUCTORS .................................... 120
  13.27.5 METHODS ........................................... 120
13.28 CLASS V2FORM ........................................... 120
  13.28.1 DECLARATION ........................................ 120
  13.28.2 CONSTRUCTOR SUMMARY ............................ 120
  13.28.3 METHOD SUMMARY .................................. 121
  13.28.4 CONSTRUCTORS .................................... 121
  13.28.5 METHODS ........................................... 121

14 KNOWN PROBLEMS AND CAVEATS ............................ 121
Part I

INTRODUCTION
1 INTRODUCTION

This guide will explain how to install, setup and use the VOMS server and its associated client-side utilities. It will also describe how to use the provided API.

1.1 SERVICE ARCHITECTURE

The service follows an established client-server architecture. It consists of a server (vomsd), a client (voms-proxy-init) and some ancillary utilities (voms-proxy-info, voms-proxy-destroy, voms-proxy-list). After the service has been setup, users are supposed to replace use of grid-proxy-init with use of voms-proxy-init, generating a proxy certificate that while backward-compatible with the one generated by the globus command, contains extra informations about the user and the VO's he belongs to.

1.2 INTERACTIONS WITH OTHER SERVICES

There is no direct interaction with other services. Services that wish to take advantage of VOMS, must do so through the API described in a later section.

2 BASIC CONCEPTS

2.1 ATTRIBUTE CERTIFICATE

An Attribute Certificate (AC for short) is a PKI container, defined in RFC 3281[1], capable of containing a set of attributes tied to a specific identity. It is the system used by VOMS to issue its attributes.

2.2 GROUPS, ROLES AND GENERIC ATTRIBUTES

Members of a Virtual Organization can be organized into groups. These groups can then be directly represented as voms groups. Groups are organized in a hierarchical tree, where each group may have zero, one or more subgroups, with no limitation on tree depth. The root of the tree is fixed, and is the VO itself.

A group name contains the representation of the path leading to it from the root. For example, if a user were member of the subgroup bologna of the group italian in the VO test, the group name as represented by VOMS will be /test/italian/bologna.

Groups There are no effective limits on the length of a group name, excepts those enforced by the underlying DB in which the name is stored. However, only alphanumeric characters plus ‘-’, ‘.’ and ‘~’ are allowed in group name.

Finally, membership in a subgroup requires membership in its parent group. From this it should be evident that all users must at least be members of the root group.

Roles Not all members of a group are necessarily equal, but some may occasionally have some special rights that, while not always needed, may indeed be necessary for the execution of special tasks. A simple example of this is the softwaremanager or sgm role that corresponds to the right of installing software on grid nodes.

This need is represented by the VOMS Roles. A VOMS Role is always associated to a specific group. E.g, holding the role sgm in the /test/italian group is a different thing than holding the same role in the /test group.
Restriction on role names are the same as restriction on group names. Furthermore, Role names starting with VOMS are reserved for use by VOMS itself. Finally, the special name NULL may indicate that no specific role is held.

Generic Attributes Finally, it should be noted that not all the characteristics of a user can be represented by a combination of groups and roles. For example, consider the need for registering the guarantor of a user.

For these cases, generic attributes are used. They consist of triple (name, value, qualifier), with the qualifier optional. As an example, the guarantor case above could be represented by the following tuple: (guarantor, “Guarantor Smith”)

There are no hardcoded limits on the number, length and character usable in generic attributes.

2.2.1 FQAN

A Fully Qualified Attribute Name (FQAN for short) is a compact way to represent a user’s membership in a group, along with its role holdership, if any.

Its general syntax would be: <groupnam>/Role=<rolename>/Capability=NULL where the /Capability=NULL may be omitted, since it refers to a deprecated feature of VOMS.

For example, belonging to the group /test/italian may be represented by the following FQAN:

/test/italian/Role=NULL/Capability=NULL

while holding the role sgm in the same group will be represented by the following FQAN:

/test/italian/Role=sgm/Capability=NULL
Part II

Quickstart Guide
3 Compatibility

With version 1.6.0 and onwards, compatibility with VOMS version 1.1.x and previous version has been dropped. This means that servers that are not capable of generating ACs are now unsupported.

4 Installing and running a VOMS server

To complete configuration of voms, you are supposed to execute the 
/opt/glite/libexec/voms_install_db command. It takes the following options:

- **mysql-home** This option lets you specify the home directory of mysql. This information is usually included in the $MYSQL_HOME environment variable, and if that is the case on your machine then you do not need to specify this option. For this reason, this option is only needed if MySQL support is desired.

- **oracle-home** This option specifies where the Oracle installation is based. For this reason, it should only be specified if Oracle support is desired. There are no defaults.

- **db** This is the name of the database that will contain the information about the VO. Its default name is “voms_<vo name>”.

- **port** This is the port number where the VOMS server will be listening. The default is 15000, and recommended choices for servers other than the first are 15001, 15002, etc...

- **voms-vo** This is the name of the VO to which the VOMS server belongs. There is no default. For this reason, this option must be always specified.

- **db-admin** This is the name of the DB user which will create the tables. It is needed because the script needs to create a new DB and a new user. Its default value is “root”, which is the standard on the default MySQL installation.

- **db-pwd** This is the password of the DB account specified by the previous option. There are no defaults. This is a required option.

- **voms-name** This is the username of the voms MySQL account that will be setup to access the newly created DB. Its default value is “voms_[VONAME]”, and it is perfectly fine if you are installing a single server. If you are installing further servers on the same machine, you MUST change this name to some other value.

- **voms-pwd** This is the password associated with the voms-name account. If not specified, a random password is created. You should always specify a new value.
This is a unique code for each server installed on the same host. It is a value between 0 and 65535, and its default is the value of \texttt{--port}.

**Compatibility**

This option is only present for backwards compatibility with previous versions. Its default value is perfectly acceptable, and so should never be explicitly specified anymore.

\texttt{--db-type}

This specifies the type of db that will be used by the server. Currently accepted values are mysql and oracle. There is no default for this option. This is a required option.

\texttt{--sqlloc}

This specifies the name of the DB interface library. Again, there is no default for this option. The library should be in $LD\_LIBRARY\_PATH or in ld.so.conf, or the full path name should be specified. This is a required option.

\texttt{--compat}

This option must be specified if you plan to use voms 1.5.x on a MySQL backend with an old version of voms-admin. It requires \texttt{--db-type} to be mysql.

\texttt{--newformat}

This forces the server to generate ACs in the new (correct) format. This is meant as a compatibility feature to ease migration while the servers upgrade to the new version.

\texttt{--socktimeout}

This option specifies, in seconds, the maximum amount of time after which a stalled connection will be closed by the server.

\texttt{--loglevel}

\texttt{--logtype}

\texttt{--logformat}

\texttt{--logdateformat}

These four options setup the logging mechanism. For an explanation of their meaning, see the definition at 6.2.1

\texttt{--help}

Prints a short reminder of these options.

A couple of example invocations follows: for the first VO,

\begin{verbatim}
voms-install-db --vo-name my-vo --mysql-pwd 'some' --voms-pwd 'thing'
\end{verbatim}

For a second VO on the same host,

\begin{verbatim}
voms-install-db --db new-vo --port 15001 --vo-name new-vo --mysql-pwd 'some' --voms-name 'voms2' --voms-pwd 'thing'
\end{verbatim}

The server also needs to have a host certificate installed. Obtain it from your CA using the CA-specific procedures, and then copy the certificate in /etc/grid-security/hostcert.pem and the private key.
to /etc/grid-security/hostkey.pem. The owners should be set to root.root for both files, and permission should be, respectively, 644 and 600 or, better, 444 and 400.

5 USING THE CLIENT COMMAND

5.1 GENERAL SETUP

There is some common setup that must be done prior to use of either the client commands or the APIs.

5.1.1 VOMSDIR SETUP

The X509_VOMS_DIR environment variable must point to a directory where a copy of all the certificates of the supported VOMS servers are kept. In this directory there should be a subdirectory for each VO, named as the VO itself, and all certificates belonging to a server for that VO should be placed in the subdirectory.

Alternatively, a file named <hostname>.lsc can be placed in the directory. If so, its format should be a list of subject/issuer DN couples detailing the exact certification path, up to and including the CA, of a certificate authorized to sign ACs for the specific VO. More than one such path can be specified by separating the paths with a '—— NEXT CHAIN ——' line.

If the X509_VOMS_DIR variable is not specified or is empty, than it defaults to "/etc/grid-security/vomsdir".

Compatibility

As a backwards compatibility feature, if a server certificate is not found in the $X509_VOMS_DIR/<VO> directory, then it is searched for also in $X509_VOMS_DIR.

This is temporary, and will eventually be removed, so do not rely on this working.

5.1.2 VOMSES FILES

Be sure in "/opt/glite/etc/vomses" or in "$HOME/.glite/vomses" you have put a copy of the vomses file distributed by all the VOMS servers you wish to contact. This subtree will be recursed into to examine all pertinent files.

The format of a vomses file is a set of lines, each with the following format: "nick" "hostname" "port" "server’s DN" "alias" "globus version"

Where nick is the nickname which will be used as the VO name for the voms-proxy-init command, hostname is the host on which the server is running, port is the port on which it is listening, server’s DN is the DN of the server’s certificate, alias is ignored for now, but should be put equal to nick, and finally globus version is optional, but should be encoded, in the same way as for the --globus option of voms-proxy-init, the version of globus on which the server is running.

It is possible to assign the same nick to several servers. This is interpreted by voms-proxy-init as declaring the servers as replicas of each other, in which case they will be contacted in random order until one succeeds or all fail.
The easier way to comply to both previous points is to install the VO config RPM that should be distributed by the VOMS servers themselves.

## 5.1.3 User credentials

While user credentials may be put anywhere, and then their location passed to voms-proxy-init via the appropriate options, there are obviously default values.

User credentials should be posted in the `.globus` subdirectory. Both PKCS12 and PEM formatted credentials are okay. The default name for the PKCS12 one is `usercert.p12`, while `usercert.pem` and `userkey.pem` are the default names for the PEM formatted one.

In case both the PEM and PKCS12 formats are present, PEM takes precedence.

The user certificate should at the most have permission 600, while the user key should be 400.

## 5.2 Using the client commands

What follows here is an explanation of all the client commands, along with some examples of their usage. It is *not* meant to be a complete reference. For this see the Reference Guide section of this document.

### 5.2.1 voms-proxy-init

voms-proxy-init is the command that should be used to create a proxy for usage on the grid. Its basic syntax is:

```
voms-proxy-init --voms valerio
```

where `vname` is the name of the VO to which the user belongs. This will create a proxy containing all the groups to which the user belongs, but none of the roles. Also, the `--voms` option may be specified multiple times in case the user belongs to more than one VO.

It is also possible to omit the `--voms` option entirely. This will however result in the creation of a completely globus-standard proxy, and is not advised since such proxies will not be usable under gLite 3.0.0 and beyond.

As stated above, no roles are ever include in proxy by default. In case they are needed, they must be explicitly requested. For example, to request the role `sgm` in the `/test/italian` group, the following syntax should be used:

```
voms-proxy-init --voms test:/test/italian/Role=sgm
```

thus obtaining a role that will be included in the AC, in addition to all the other information that will be normally present. In case multiple roles are needed, the `--voms` option may be used several times.

Aside from the fact that the root group will always be present first, the order of all the others is unspecified, and there is no guarantee that it will remain constant between call to voms-proxy-init. If a specific order is needed, it should be explicitly requested via the `--order` option. For example, the following command line:

```
voms-proxy-init --voms test:/test/Role=sgm --order /test/Role=sgm \  --order /test/Role=Null
```

...
Asks for the Role sgm in the root group, and specifies that the resulting AC should begin with that role and followed by membership in the root group, while posing no requirements on the ordering of the remaining FQANs.

The validity of an AC created by VOMS will generally be as long as the proxy which contains it. However, this is not always true. For starters, the VOMS server is configured with a maximum validity for all the ACs it will create, and a request to exceed it will simply be ignored. If this happens, the output of voms-proxy-init will indicate the fact.

For example, in the following output (slightly reformatted for a shorter line then on screen):

```
[marotta@datatag6 certificates]$ voms-proxy-init --voms valerio --vomslife 50:15
Enter GRID pass phrase:
Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
Creating temporary proxy ......................... Done
Contacting datatag6.cnaf.infn.it:50002
[/C=IT/O=INFN/OU=Host/L=CNAF/CN=datatag6.cnaf.infn.it] "valerio" Done
Warning: datatag6.cnaf.infn.it:50002:
  The validity of this VOMS AC in your proxy is shortened to 86400 seconds!
Creating proxy ..................................... Done
Your proxy is valid until Fri Sep 8 01:55:34 2006
```

You can see that the life of the voms AC has been clearly shortened to 24 hours, even though 50 hours and 15 minutes had been requested.

If your certificate is not in the default place, you may specify it explicitly by using the --cert and --key options, like in the following example:

```
voms-proxy-init --voms valerio --cert /home/marotta/mycert.pem --key /home/marotta/mykey.pem
```

Finally, in case several options have to be specified several times, profiles can be created. For examples:

```
[marotta@datatag6 marotta]$ cat voms.profile
--voms=valerio
--lifetime=50:15
--cert=/home/marotta/mycert.pem
--key=/home/marotta/mykey.pem
--order=/valerio/group1
```

followed by:

```
[marotta@datatag6 marotta]$ voms-proxy-init --conf voms.profile
```

is equivalent to:

```
[marotta@datatag6 marotta]$ voms-proxy-init --voms valerio
   --lifetime 50:15 --cert /home/marott/mycert.pem
   --key /home/marotta/mykey.pem --order /valerio/group1
```

with the obvious advantages of being much less error-prone.
5.2.2 VOMS-PROXY-INFO

Once a proxy has been created, the voms-proxy-info command allows the user to retrieve several information from it. The two most basic uses are:

```
[marotta@datatag6 certificates]$ voms-proxy-info
subject : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini/CN=proxy
issuer : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
identity : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
type : proxy
strength : 512 bits
path : /tmp/x509up_u502
timeleft : 10:33:52
```

which, as you can see, prints the same information that would be printed by a plain grid-proxy-info, and then there is:

```
[marotta@datatag6 certificates]$ voms-proxy-info --all
subject : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini/CN=proxy
issuer : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
identity : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
type : proxy
strength : 512 bits
path : /tmp/x509up_u502
timeleft : 11:59:59
=== VO valerio extension information ===
VO : valerio
subject : /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
issuer : /C=IT/O=INFN/OU=Host/L=CNAF/CN=datatag6.cnaf.infn.it
attribute : /valerio/Role=NULL/Capability=NULL
attribute : /valerio/asdasd/Role=NULL/Capability=NULL
attribute : /valerio/qwerty/Role=NULL/Capability=NULL
attribute : attributeOne = 111 (valerio)
attribute : attributeTwo = 222 (valerio)
timeleft : 11:59:59
```

which prints everything that there is to know about the proxy and the included ACs. Several options enable the user to select just a subset of the information shown here.

5.2.3 VOMS-PROXY-DESTROY

The voms-proxy-destroy command erases an existing proxy from the system. Its basic use is:

```
[marotta@datatag6 certificates]$ voms-proxy-destroy
[marotta@datatag6 certificates]$
```

As can be seen, no output is given in case of a successful usage.
5.2.4 VOMS-PROXY-LIST

The `voms-proxy-list` command is used to interrogate a VOMS server to discover the list of group/role combination the user belongs to or may hold.

Its basic usage is:

```bash
[marotta@datatag6 certificates]$ voms-proxy-list --voms valerio
Enter GRID pass phrase:
Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Vincenzo Ciaschini
Creating temporary proxy ........................................... Done
Contacting datatag6.cnaf.infn.it:50002
[/C=IT/O=INFN/OU=Host/L=CNAF/CN=datatag6.cnaf.infn.it] "valerio" Done
Available attributes:
/valerio/Role=NULL/Capability=NULL
/valerio/asdasd/Role=NULL/Capability=NULL
/valerio/qwerty/Role=NULL/Capability=NULL
/valerio/qwerty/Role=King/Capability=NULL
```
Part III

Reference Guide
6 VOMSD

Installing the server using the above described procedure should correctly create a set of configuration files that will execute it with the proper options. However, there are many other options that are not used by the default configuration script. The following lines will so describe the totality of the options.

6.1 Software Requirements

Running a VOMS servers poses a few requirements on the physical host: aside from the usual C/C++ libraries, expat 1.95 or later must be installed, along with C client libraries for either MySQL or Oracle. In the former case, at least version 4.0.10 is required, while for the latter 10.1 is the absolute minimum. Also, at least version 2.0 of Globus is required. Note however that, due to several incompatibilities among different versions, it should be the same version against which the software was compiled.

6.2 Command line interface

- **--port** The port number on which the server should be listening. The default value is 50000
- **--vo** The name of the VO to which this server will belong. The default value is “unspecified”.
- **--globusid** The value of the GLOBUSID environment variable. There is no default value.
- **--globuspwd** The value of the GLOBUSPWD environment variable. There is no default value.
- **--x509_cert_dir** The location where the CA certificates are kept. The default value is /etc/grid-security/certificates
- **--x509_cert_file** A file containing all the CA certificates. There is no default value.
- **--x509_user_proxy** The location of the server’s proxy. There is no default value. Usage of this option is strongly discouraged.
- **--x509_user_cert** The location of the server’s certificate. The default value is “/etc/grid-security/hostcert.pem”
- **--x509_user_key** The location of the server’s private key. The default value is “/etc/grid-security/hostkey.pem”
- **--desired_name** OBSOLETE. This option will be removed in the future. Do not use it.
- **--foreground** OBSOLETE. This option will be removed in the future. Do not use it.
- **--username** The name of the user with which VOMS will access the DB. The default value is “voms”
- **--dbname** The name of the DB that VOMS will use. The default value is “voms”.
- **--timeout** The maximum length of validity of the ACs that VOMS will grant. (in seconds) The default value is 24 hours
-passfile  The location of the file containing the password needed to access the DB. This file should be owned by root and have permissions set to 400. There is no default value. If this option is not specified, than the password will be asked to the user during server startup.

-uri     The URI that the server will publish for himself. The default value is :<port>.

-globus   The version of Globus installed on the server’s host. Use 20 for Globus 2.0 or Globus 2.1, and 22 for Globus 2.2 and Globus 2.4. The default value is 22.

-version  Prints the version number and compilation date and then exits.

-backlog Sets the backlog on the socket. The default value is 50.

-debug   Slightly modifies the internal workings of the server to ease debug. Never use it on production servers. Use of this option is guaranteed to severely hurt scalability.

-conf    Lets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format "option=value".

-code    OBSOLETE. This option will be removed in the future. Do not use it.

-sqlloc  Specifies the location of the DB interface library. If this is not in the library search path, than the full path should be specified. This is a required option.

-compat  Specify for compatibility with databases created by VOMS 1.5.4 or before. Only makes sense id the unerlying DB is MySQL.

-socktimeout Specifies the timeout after which an inactive connection should be dropped. Defaults to 60 seconds.

-contactstring Specified the hostname for the DB.

-mysql-port Specifies the port where MySQL is listening.

-mysql-socket Specifies the socket where MySQL is listening.

-newformat Creates AC in the newer, correct format. Warning! Specifying this option breaks compatibility with VOMS APIs version 1.6.8 or before.

-logfile

-loglevel

-logdateformat

-logformat

-logtype

-logmax These six options are explained below in the Logging Options section.
6.2.1 Logging Options

Logging in VOMS is highly configurable both in its format and content. For this reason, an entire section is dedicated to it.

First of all, the `--logfile` option specifies where the main logfile is placed. If not specified, this defaults to “/opt/glite/var/log/<voname>.log”.

However this is just the main file. When the amount specified by the `--logmax` option is reached, the existing logfile is renamed by prepending `.1` to its name and a new one is created. If the `.1` file already existed, than it is renamed as a `.2`, and so on and on.

Then, the `--loglevel` option specified how much to log. It takes parameter according to the following table:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LEV_NONE</td>
</tr>
<tr>
<td>2</td>
<td>LEV_ERROR</td>
</tr>
<tr>
<td>3</td>
<td>LEV_WARN</td>
</tr>
<tr>
<td>4</td>
<td>LEV_INFO</td>
</tr>
<tr>
<td>&gt;=5</td>
<td>LEV_DEBUG</td>
</tr>
</tbody>
</table>

Where each successive level also logs all information that would be logged by the previous ones. The default value is 4, (LEV_INFO). Please note the specifying LEV_DEBUG more than quintuples the amount of information logged.

The option `--logtype` decides what type of information to log. Its value is decided by OR-ing values from the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T_STARTUP</td>
</tr>
<tr>
<td>2</td>
<td>T_REQUEST</td>
</tr>
<tr>
<td>4</td>
<td>T_RESULT</td>
</tr>
</tbody>
</table>

The default value is 7, i.e: (T_REQUEST | T_STARTUP | T_RESULT). It is strongly suggested that this value is left as it is.

6.3 Server setup and maintenance

Aside from the setup specified by the above command line options, the following configuration needs to be done:

- a host certificate/host key couple needs to be installed, along with a directory containing all acceptable CAs along with the corresponding CRL.

- if the underlying DB is meant to be Oracle, a `oranames.tns` file should be available to detail the connection. Such a file should either be referenced by the `$TNS_ADMIN` variable or be placed in the `/opt/glite/etc/voms/<voname>/` directory. The content and syntax of this file is out of the scope of this document. See Oracle documentation for details.

While the server is running, it is possible to change the value of most of his options without requiring a server restart. If the options were loaded from a `.conf` file, then after changing it it is usually enough to send an (HANGUP) signal (via `kill -HUP`) to the VOMS process with the higher PID.

This is not valid for the following options, which do require a restart when changed: `--port`, `--backlog`, `--conf`, `--sqlloc` and, if the underlying DB is Oracle, the following options also require a restart when changed: `--dbname`, `--username`, `--passfile` and the content of the pointed file.
6.4 Example Configuration

The following example configuration is taken from the dteam production server, with passwords and usernames obviously changed.

- **Content of** `/opt/glite/etc/voms/dteam`

  - `-rw-r------ 1 root root 228 Jun 8 16:55 voms.conf`
  - `-rw-r------ 1 root root 13 Jun 8 16:53 voms.pass`

- **Content of** `/opt/glite/etc/voms/dteam/voms.conf`

  ```
  --code=15004
  --dbname=prod
  --logfile=/var/log/glite/voms.dteam
  --loglevel=4
  --logtype=7
  --passfile=/opt/glite/etc/voms/dteam/voms.pass
  --port=15004
  --sqlloc=/opt/glite/lib/libvomsoracle.so
  --username=voms_user
  --vo=dteam
  ```

- **Content of** `/opt/glite/etc/voms/dteam/voms.pass`

  `LunReC@f`

- **Content of** `$TNS_ADMIN`

  ```
  prod=
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = lcgr4-v.cern.ch)(PORT = 10121))
   (ADDRESS = (PROTOCOL = TCP)(HOST = lcgr1-v.cern.ch)(PORT = 10121))
   (ADDRESS = (PROTOCOL = TCP)(HOST = lcgr2-v.cern.ch)(PORT = 10121))
   (ADDRESS = (PROTOCOL = TCP)(HOST = lcgr3-v.cern.ch)(PORT = 10121))
   (LOAD_BALANCE = yes)
   (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = lcg_voms.cern.ch)
    (FAILOVER_MODE = (TYPE = SELECT)(METHOD = BASIC)(RETRIES = 200)(DELAY = 15))
   )
  )
  ```

7 VOMS-PROXY-INIT

This command is used to contact the VOMS server and retrieve an AC containing user attributes that will be included in the proxy certificates.
Before running voms-proxy-init, or any other client command, it is necessary to install the VO config RPM that should be distributed by the VOMS servers themselves, and has probably already been done by the system manager.

This usually results in installing files in the “/etc/grid-security/vomsdir” and “/opt/glite/etc/vomses” directories.

One step that cannot be done by the system manager is instead to install the user certificate. To do so, the certificate must be installed in $HOME/.globus either as a PKCS12 container called usercert.p12 or as a PEM certificate/key couple, called respectively usercert.pem and userkey.pem.

Permissions should be set at 600 or 400 for usercert.pem and 400 for userkey.pem and usercert.p12.

The voms-proxy-init command can be invoked with the following options:

- **–voms** Specifies which server to contact. The parameter has the following syntax: 
  <alias>[:<command>] where <alias> is the alias of the server as specified in the vomses files. If the same alias is associated to more than a single server, than those servers are considered replicas of each other, and are contacted in random order until one succeeds or all fail.
  The [:<command>] part is optional. If not specified then the information returned will include only group membership, while if you specify :<groupname>/Role=<rolename> then you will also get the role you asked for, provided that the server is already prepared to grant it to you.
  This option can be specified multiple times, and the operations will be carried out in the exact order in which these options are specified in the command line.
  You may also specify :all, which will get you all possible roles.

- **–version** Prints version information and exits.

- **–quiet** Prints only minimal informations. WARNING: some vital warnings may get overlooked by this option.

- **–verify** Verifies the certificate from which to create the proxy. This is not normally done, since in any case, an invalid user certificate will be detected when the proxy is actually used.

- **–pwstdin** Specifies that the private key’s passphrase should be received from stdin instead than directly from the console.

- **–limited** Creates a limited certificate.

- **–valid** Specifies the length of the validity of both the generated proxy and the AC, measure in hours:minutes.

- **–hours** Specifies the length of the validity of the generated proxy, measure in hours. The default value is 12 hours.


–bits
Specifies the length in bits of the private key of the newly generated proxy certificate. The default value is 512.

–cert
Specifies a non-standard location of the user’s certificate. The default value is “$X509_USER_CERT” or, if this value is unset, “/HOME/.globus/usercert.pem”, or “/HOME/.globus/usercert.p12” if the former one is not present..

–key
Specifies a non-standard location of the user’s private key. The default value is “$X509_USER_KEY” or, if this value is unset, “/HOME/.globus/userkey.pem”. This option is ignored if the PKCS12 format for the credentials is used.

–certdir
Specifies a non-standard location of the trusted cert (CA) directory. The default value is “/etc/grid-security/certificates”.

–out
Specifies a non-standard location of the generated proxy certificate. The default value is “$X509_USER_PROXY” or, if this is empty, “/tmp/x509up_u<id>” where <id> is the user’s UID.

–order
This option specifies the order in which the attributes granted by the VOMS servers should be returned.

The format of the parameter for this option is: <group[:role]>, where “group” is a group name and “role” is an (optional) role name. This option may be specified multiple times, to create an ordered list of attributes.

Each server will receive this list, and will strive to return the attributes he will grant in the exact order specified by this list. All attributes not on this list will be returned in an unspecified order, but after the recognized attributes. Also, should this list include an attribute unknown to a specific server, such an attribute will be simply ignored.

Finally, should a server be unable to grant the first attribute of the list, it will return a warning to the user. However, this warning will only be significat for the first server contacted.

–target
This option take advantage of the capability ACs have to target themselves to a specific set of receivers, so that only those receivers should, in conforming implementation, act on the data they get, while all others should reject it.

This options lets you specify a set of FQHNs, each on a separate option, that will constitute the set of targets for the generated AC.
This option lets you specify the validity, in h:m format, that you wish for the generated ACs. Remember that this value has only an advisory role. VOMS servers may lower this duration if the requested value exceeds the maximum they have been configured to grant. The default value of this option is “the value of the –hours or –valid options.”

The version of Globus installed on the server’s host. Use 20 for Globus 2.0 or Globus 2.1, 22 for Globus 2.2 and Globus 2.4, 30 for globus 3.x and 40 for globus 4.0 The default value is the value of the $GLOBUS_VERSION variable, or 22 if that is unset.

For its normal workings, voms-proxy-init first creates a proxy with which to contact the VOMS servers, and then creates a new proxy to hold all of the returned ACs. This option skips the creation of the first proxy, and assumes that such a proxy already exists.

This option save the ACs in a separate file, instead than including them into a proxy certificate.

Specify this if you do not want to allow warnings to be printed.

Specify this if you want warnings to be upgraded into errors.

Deprecated. Use –vomses instead.

Deprecated. Use –vomses instead.

This option specifies the location of an additional directory in which to search for vomses files or directly such a file. /opt/glite/etc/vomses and $HOME/.glite/vomses are always added by default. This option may be specified multiple times.

Lets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format “option=value”.

This option prints a series of additional debug informations on stdout. The additional output returned by this option should always be included into bug reports for the voms-proxy-init command. User should not, however, ever rely on informations printed by this options. Both content and format are guaranteed to change between software releases.

This options allows a policy file to be included in the proxy.

This option lets you specify the validity, in h:m format, that you wish for the generated ACs. Remember that this value has only an advisory role. VOMS servers may lower this duration if the requested value exceeds the maximum they have been configured to grant. The default value of this option is “the value of the –hours or –valid options.”

The version of Globus installed on the server’s host. Use 20 for Globus 2.0 or Globus 2.1, 22 for Globus 2.2 and Globus 2.4, 30 for globus 3.x and 40 for globus 4.0 The default value is the value of the $GLOBUS_VERSION variable, or 22 if that is unset.

For its normal workings, voms-proxy-init first creates a proxy with which to contact the VOMS servers, and then creates a new proxy to hold all of the returned ACs. This option skips the creation of the first proxy, and assumes that such a proxy already exists.

This option save the ACs in a separate file, instead than including them into a proxy certificate.

Specify this if you do not want to allow warnings to be printed.

Specify this if you want warnings to be upgraded into errors.

Deprecated. Use –vomses instead.

Deprecated. Use –vomses instead.

This option specifies the location of an additional directory in which to search for vomses files or directly such a file. /opt/glite/etc/vomses and $HOME/.glite/vomses are always added by default. This option may be specified multiple times.

Lets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format “option=value”.

This option prints a series of additional debug informations on stdout. The additional output returned by this option should always be included into bug reports for the voms-proxy-init command. User should not, however, ever rely on informations printed by this options. Both content and format are guaranteed to change between software releases.

This options allows a policy file to be included in the proxy.
–policy-language  This option specifies the OID for the string policy language.

–proxyver  This option specifies to which version of the proxy certificate format will the generated format conform to. Possible values are 2, 3 and 4, while the default is the one implied by the –globus option.

–pathlength  This option specifies the maximum path length for the proxy. Note that this is NOT the certificate path length, but the policy path length.

–list  This option turns the command in voms-proxy-list. When this happens, several other options get ignored. See that command manual for a description of the still valid options and their meaning.

8 VOMS-PROXY-INFO

This command is used to print to the screen the informations included in an already generated VOMS proxy.

The configuration is the same as voms-proxy-init.

The following options may be used:

–debug  This option prints a series of additional debug informations on stdout. The additional output returned by this option should always be included into bug reports for the voms-proxy-info command. User should not, however, ever rely on informations printed by this options. Both content and format are guaranteed to change between software releases.

–version  Prints version information and exits.

–conf  Lets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format “option=value”.

–file  This option lets you specify a non-standard location of the user proxy. The default value is “$X509_USER_PROXY” or, if this is empty, “/tmp/x509up_u<id>”, where <id> is the user’s UID.

–subject  Prints the DN of the proxy’s subject.

–issuer  Prints the DN of the proxy’s issuer.

–identity  Prints the DN corresponding to the identity represented by the proxy.

–type  Prints the proxy’s type.

–strength  Prints the length (in bits) of the private key.

–valid  Prints the start and end validity times.
–timeprints the end validity as a number of seconds for which the object will still be valid.
–timeleftprints how much time is left to the proxy until expiration.
–voprints the VO name of each AC.
–textprints all of the certificate.
–infolet “–subject”, “–issuer”, “–valid” and “–time” also apply to ACs, and prints attributes values.
–extraprints extra informations that were included in the proxy.
–allprints everything. (Implies all other options.)
–fqanSpecifies that attributes should be printed in the FQAN format. (default)
–extendedSpecifies that attributes should be printed in the extended format.
–existsActivates the “–hours” and “–bits” options.
–hoursVerifies that the proxy, and the ACs if “–info” was specified, will be valid for at least <H> hours.
–bitsVerifies that the proxy key has at least <B> bits.
–acexistsVerifies wether an AC of the specified VO is included in the proxy.
–acsubjectPrints the DN of the AC’s subject.
–acissuerPrints the DN of the AC’s issuer.
–serialPrints the serial number of the AC.

9 VOMS-PROXY-DESTROY

This command destroys an already existing VOMS proxy. No configuration is needed.
The following options may be used:

–debugThis option prints a series of additional debug informations on stdout. The additional output returned by this option should always be included into bug reports for the voms-proxy-info command. User should not, however, ever rely on informations printed by this options. Both content and format are guaranteed to change between software releases.
–versionPrints version information and exits.
–confLets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format “option=value”.
–quietPrints only minimal informations. WARNING: some vital warnings may get overlooked by this option.
–file This option lets you specify a non-standard location of the user proxy. The default value is “$X509_USER_PROXY” or, if this is empty, “/tmp/x509up_u<id>”, where <id> is the user’s UID.

–dryrun Only prints messages, but do not take any actions.

10 VOMS-PROXY-LIST

This command allows a user to get a list of all the group/role combination that he may request. The following options may be used:

–voms Specifies which server to contact. The parameter has the following syntax: <alias>:<command> where <alias> is the alias of the server as specified in the vomses files. If the same alias is associated to more than a single server, than those servers are considered replicas of each other, and are contacted in random order until one succeeds or all fail. The [:<command>] part is optional. If not specified then the information returned will include only group membership, while if you specify :/<groupname>/Role=<rolename> then you will also get the role you asked for, provided that the server is already prepared to grant it to you. This option can be specified multiple times, and the operations will be carried out in the exact order in which these options are specified in the command line. You may also specify :all, which will get you all possible roles.

–version Prints version information and exits.

–quiet Prints only minimal informations. WARNING: some vital warnings may get overlooked by this option.

–pwstdin Specifies that the private key’s passphrase should be received from stdin instead than directly from the console.

–cert Specifies a non-standard location of the user’s certificate. The default value is “$X509_USER_CERT” or, if this value is unset, “/HOME/.globus/usercert.pem”, or “/HOME/.globus/usercert.p12” if the former one is not present..
–key Specifies a non-standard location of the user’s private key. The default value is “$X509_USER_KEY” or, if this value is unset, “$HOME/.globus/userkey.pem”. This option is ignored if the PKCS12 format for the credentials is used.

–certdir Specifies a non-standard location of the trusted cert (CA) directory. The default value is “/etc/grid-security/certificates”.

–out Specifies a non-standard location of the generated proxy certificate. The default value is “$X509_USER_PROXY” or, if this is empty, “/tmp/x509up_u<id>” where <id> is the user’s UID.

–globus The version of Globus installed on the server’s host. Use 20 for Globus 2.0 or Globus 2.1, 22 for Globus 2.2 and Globus 2.4, 30 for globus 3.x and 40 for globus 4.0 The default value is the value of the $GLOBUS_VERSION variable, or 22 if that is unset.

–noregen For its normal workings, voms-proxy-init first creates a proxy with which to contact the VOMS servers, and then creates a new proxy to hold all of the returned ACs. This option skips the creation of the first proxy, and assumes that such a proxy already exists.

–ignorewarn Specify this if you do not want to allow warnings to be printed.

–failonwarn Specify this if you want warnings to be upgraded into errors.


–vomses This option specifies the location of an additional directory in which to search for vomses files or directly such a file. /opt/glite/etc/vomses and $HOME/.glite/vomses are always added by default. This option may be specified multiple times.

–conf Lets you specify a file from which options will be loaded. This file should have exactly one option per line, and option that do have values should be specified in the format “option=value”.

–debug This option prints a series of additional debug informations on stdout. The additional output returned by this option should always be included into bug reports for the voms-proxy-init command. User should not, however, ever rely on informations printed by this options. Both content and format are guaranteed to change between software releases.
-list  This is always specified by default.
Part IV

Application Program Interfaces
The VOMS API already come with their own documentation in doxygen format. However, that documentation is little more than a simple enumeration of functions, with a very terse description.

The aim of this document is different. Here the intention is not only to describe the different functions that comprise the API, but also to show how they are supposed to work together, what particular care the user needs to take when calling them, what should be done to maintain compatibility between the different versions, etc.

Throughout this whole document, you will find sections marked thus:

**Compatibility**

Some information

These section contain informations regarding both back and forward compatibility between different versions of the API.

**Compatibility**

Finally, please note that everything not explicitly defined in this section shall be considered a private detail and subject to change without notice.

## 11 C++ API

There are three basic classes: `data`, `voms` and `vomsdata`.

### 11.1 The data structure

The first one, `data` contains the data regarding a single attribute, giving its specification in terms of Groups, Roles and Capabilities. It is defined as follows:

```cpp
struct data {
    std::string group;
    std::string role;
    std::string cap;
};
```

All the values of these strings must be composed from regular expression: `[a-zA-Z0-9/]`.*

#### 11.1.1 GROUP

This field contains the name of a group which the user belongs into. The format of entries in this group is reminiscent of the structure of pathnames, and is the following:

```
/group/group/.../group
```

where the name of the first group is by convention the name of the Virtual Organization (VO), while each other `/group` component is a subgroup of the group immediately preceding it on the left. The character `'/` is therefore acceptable as part of a group name.

This field MUST always be filled.
11.1.2 ROLE

This field contains the name of the role which the user owns in the group specified by group. If the user does not own any particular role in that group, than this field contains the value “NULL”.

11.1.3 CAP

This field details a capability that the user has as a member of the group specified by group while owning the role specified by role. If there is no specific capability, than this value is “NULL”.

No specific format is associated to a capability. They are basically free-form strings, whose value should be agreed between the AA and the Attribute verifier.

11.2 THE VOMS STRUCTURE

The second one, voms is used to group together all the informations that can be gleaned from a single AC, and is defined as follows:

```cpp
enum data_type { 
    TYPE_NODATA, /*!< no data */
    TYPE_STD,    /*!< group, role, capability triplet */
    TYPE_CUSTOM /*!< result of an S command */
};

struct voms { 
    friend class vomsdata;
    int version;
    int siglen;
    std::string signature;
    std::string user;
    std::string userca;
    std::string server;
    std::string serverca;
    std::string voname;
    std::string uri;
    std::string date1;
    std::string date2;
    data_type type;
    std::vector<data> std;
    std::string custom;
    /* Data below this line only makes sense if version >= 1 */
    std::vector<std::string> fqan;
    std::string serial;
    /* Data below this line is private. */
private:
    /* Private data not shown here */
public:
    voms(const voms &);
    voms();
    voms &operator=(const voms &);
    ~voms();
};
```

INFSO-RI-508833 PUBLIC 38/122
The purpose of this structure is to present, in a readable format, the data that has been included in a single Attribute Certificate (AC). While the various public fields may be freely modified to simplify internal coding, such changes have no effect on the underlying AC. Let's examine the various fields in detail.

11.2.1 VERSION
int version;

This field specifies the version of this structure that is currently being used. A value of 0 indicates that it comes from an old format extension, while a value of 1 indicates that this structure comes from an AC.

Compatibility

Starting from VOMS 1.6.0, support for version 0 is no longer present. What this means is that now VOMS 1.6.0 or greater is incompatible with VOMS 1.1.x or lesser. As a consequence of this, all mentions of version 0’s peculiarities has been removed.

11.2.2 SIGLEN
int siglen;

The length of the data signature.

11.2.3 SIGNATURE
std::string signature;

This field contains a copy of the signature of the AC.

11.2.4 USER
std::string user;

This field contains the subject of the holder’s certificate in slash-separated format.

11.2.5 USERCA
std::string userca;

This field contains the subject of the CA that issued the holder’s certificate, in slash-separated format.

11.2.6 SERVER
std::string server;

This field contains the subject of the certificate that the AA used to issue the AC, in slash-separated format.

11.2.7 SERVERCA
std::string serverca;
This field contains, in slash-separated format, the subject of the CA that issued the certificate that the AA used to issue the AC.

### 11.2.8 VONAME

```cpp
std::string voname;
```

This field contains the name of the Virtual Organization (VO) to which the rest of the data contained in this structure applies to.

### 11.2.9 URI

```cpp
std::string uri;
```

This is the URI at which the AA that issued this particular AC can be contacted. Its format is:

```
fqdn:port
```

where *fqdn* is the Fully Qualified Domain Name of the server which hosts the AA, and *port* is the port at which the AA can be contacted on that server.

### 11.2.10 DATE1, DATE2

```cpp
std::string date1;
std::string date2;
```

These are the dates of start and end of validity of the rest of the informations. They are in a string representation readable to humans, but they may be easily converted back to their original format.

Here follows a code example doing that conversion:

```cpp
ASN1_TIME *
convtime(std::string data)
{
    ASN1_TIME *t = ASN1_TIME_new();

    t->data = (unsigned char *)(data.data());
    t->length = data.size();
    switch(t->length) {
        case 15:
            t->type = V_ASN1GENERALIZEDTIME;
            break;
        default:
            ASN1_TIME_free(t);
            return NULL;
    }
    return t;
}
```

### 11.2.11 TYPE

```cpp
data_type type;
```

This datum specifies the type of data that follows. It can assume the following values:

**TYPE_NODATA** There actually was no data returned.

**Compatibility**

This is only true for version 0 structures. The following versions will simply not generate a *voms* structure in this case.
**TYPE_CUSTOM** The data will contain the output of an “S” command sent to the server.

Compatibility

Again, this type of datum will only be present in version 0 structures. Due to lack of use, support for it has been disabled in new versions of the server.

**TYPE_STD** The data will contain (group, role, capabilities) triples.

11.2.12 **STD**

```cpp
std::vector<data> std;
```

This vector contains all the attributes found in an AC, in the exact same order as they were found, in the format specified by the `data` structure. It is only filled if the value of the `type` field is `TYPE_STD`.

Compatibility

This structure is filled in both version 1 and version 0 structures, although this is scheduled to be left empty after the transition period has passed.

11.2.13 **CUSTOM**

```cpp
std::string custom;
```

This field contains the data returned by the “S” server command, and it is only filled if the `type` value is `TYPE_CUSTOM`.

11.2.14 **FQAN**

```cpp
std::vector<std::string> fqan;
```

This field contains the same data as the `std` field, but specified in the Fully Qualified Attribute Name (FQAN) format.

11.2.15 **SERIAL**

```cpp
std::string serial;
```

This field contains a text representation of the serial number of the certificate.

11.2.16 **GETAC()**

```cpp
AC *GetAC();
```

This method returns a copy of the underlying AC. This means that eventual changes are of no consequence.

11.2.17 **GETATTRIBUTES()**

```cpp
std::vector<attributelist>& GetAttributes();
```

This method is used to retrieve the list of generic attributes that were present in the AC. The result is a vector of `attributelist` structures, one each for every source of attributes. The definition of `attributelist` follows;

```cpp
struct attributelist {
    std::string grantor;
    std::vector<attribute> attributes;
};
```

Here `grantor` is a string identifying who granted the attributes, while `attributes` is a vector of the attributes themselves, defined as follows:
struct attribute {
    std::string name;
    std::string qualifier;
    std::string value;
};

In this structure, name is the name of the attribute, value is its value, while qualifier is an optional qualifier. qualifier may be the empty string if no special qualifier was specified for this attribute.

11.2.18 VOMS::VOMS()

voms::voms()

This is the standard default constructor. Please note that a structure created this way would not contain any real data. The only use for this constructor is to create a “placeholder” structure to which you will copy data using the copy operator.

11.2.19 VOMS::VOMS(CONST VOMS &)

voms::voms(const voms &)

This is the standard copy constructor. Structures allocated via this method will retain an exact copy of the data of their source. Use of either this constructor or the assignment operator are the only supported method to copy a voms object.

11.2.20 VOMS::OPERATOR=(CONST VOMS &)

voms::voms & operator=(const voms &)

This defines an assignment operator between two different voms structures. Use of either this constructor or the copy constructor are the only supported method to copy a voms object.

11.3 VOMSDATA

The purpose of this object is to collect in a single place all informations present in a VOMS extension. It is defined so.

struct vomsdata {
    private:

    /* Private data not shown. */

    public:
        verror_type error; /*! < Error code */

        vomsdata(std::string voms_dir = "", std::string cert_dir = ");
        bool LoadSystemContacts(std::string dir = ");
        bool LoadUserContacts(std::string dir = ");
        std::vector<contactdata> FindByAlias(std::string alias);
        std::vector<contactdata> FindByVO(std::string vo);
        void Order(std::string att);
        void ResetOrder(void);
        void AddTarget(std::string target);
        std::vector<std::string> ListTargets(void);
        void ResetTargets(void);
        std::string ServerErrors(void);
        bool Retrieve(X509 *cert, STACK_OF(X509) *chain,
                 recurse_type how = RECURSE_CHAIN);
bool Contact(std::string hostname, int port,
            std::string servsubject,
            std::string command);
bool ContactRaw(std::string hostname, int port,
                std::string servsubject,
                std::string command,
                std::string &raw, int &version);
void SetVerificationType(verify_type how);
void SetLifetime(int lifetime);
bool Import(std::string buffer);
bool Export(std::string &data);
bool DefaultData(voms &);
std::vector<voms> data;
std::string workvo;
std::string extra_data;

private:
    /* More not shown private methods. */

public:
    std::string ErrorMessage(void);
    bool RetrieveFromCtx(gss_ctx_id_t context, recurse_type how);
    bool RetrieveFromCred(gss_cred_id_t credential, recurse_type how);
    bool Retrieve(X509_EXTENSION *ext);
    bool RetrieveFromProxy(recurse_type how);
~vomsdata();

private:
    /* Even more private methods. */

public:
    vomsdata(const vomsdata &);
};

Let us see the fields in detail.

11.3.1 ERROR
verror_type error;

This field contains the error code returned by one of the methods. Please note that the value of this field
is only significant if the last method called returns an error value. Also, the value of this field is subject
to change without notice during method executions, regardless of whether an error effectively occurred.
The possible values returned are the following:

enum verror_type {
    VERR_NONE,
    VERR_NOSOCKET,
    VERR_NOIDENT,
    VERR_COMM,
    VERR_PARAM,
    VERR_NOEXT,
    VERR_NOINIT,
VERR_TIME,
VERR_IDCHECK,
VERR_EXTRAINFO,
VERR_FORMAT,
VERR_NODATA,
VERR_PARSE,
VERR_DIR,
VERR_SIGN,
VERR_SERVER,
VERR_MEM,
VERR_VERIFY,
VERR_TYPE,
VERR_ORDER,
VERR_SERVERCODE,
VERR_NOTAVAIL
}

In general, a first idea of what each code means can be gleaned from the code name, but every method description will document what errors its execution may generate and on which conditions.

11.3.2 DATA

This field contains a vector of voms structures, in the exact same order as the corresponding ACs appeared in the proxy certificate, and containing the informations present in that AC.

11.3.3 VOMSDATA::VOMSDATA(STRING VOMS_DIR, STRING CERT_DIR)

vomsdata::vomsdata(std::string voms\_dir='''', std::string cert\_dir=''''}

This is the standard constructor that also doubles as the default constructor.

voms_dir This is the directory where the VOMS server’s certificates are kept. If this value is empty (the default), then the value of $X509$VOMS_DIR is considered, and if this is also empty than its default is /etc/grid-security/vomsdir.
cert_dir This is the directory where the CA certificates are kept. If this value is empty (the default), then the value of $X509$CERT_DIR is considered, and if this is also empty than its default is /etc/grid-security/certificate.

Compatibility

This function is the only supported way to create and initialize a vomsdata structure other than the copy constructor. It is forbidden to ever take the sizeof() of this class.

The default values are strongly suggested. If you want to hardcode specific ones, think very thoroughly about the loss of configurability that it would entail.

11.3.4 BOOL VOMSDATA::LOADSYSTEMCONTACTS(STRING DIR)

bool vomsdata::LoadSystemContacts(std::string dir = ‘’}

This function loads the vomses files that are shared system-wide.
dir This is the directory in which the various vomses files are kept. If left blank, it defaults to /opt/glite/etc/vomses.

RETURNS

The return value is true if all went well and false otherwise. In the latter case the vomsdata::error member becomes significant, and it may assume the following values:
VERR_DIR

The function tried to access something that either was not a directory or a regular file, could not be read, or it had the wrong permissions. The acceptable permissions are, at the most, 644 for files and 755 for directories.

VERR_FORMAT

The file was not in the expected format.

11.3.5  BOOL VOMSDATA::LOADUSERCONTACTS(STD::STRING DIR)

```cpp
bool vomsdata::LoadUserContacts(std::string dir = "");
```

This function loads the vomses files that are user-specific.

**dir**  This is the directory in which the various vomses files are kept. If left blank, it defaults to `${VOMS_USERCONF}`. If this is also empty, then the last default is `/glite/vomses`.

**RETURNS**

The return value is true if all went well and false otherwise. In the latter case the `vomsdata::error` member becomes significant, and it may assume the following values:

VERR_DIR

The function tried to access something that either was not a directory or a regular file, could not be read, or it had the wrong permissions. The correct permissions are at the most 644 for files and 755 for directories.

VERR_FORMAT

The file was not in the expected format.

11.3.6  STD::VECTOR<CONTACTDATA> VOMSDATA::FINDBYALIAS(STD::STRING ALIAS)

```cpp
std::vector<ContactData> vomsdata::FindByAlias(std::string alias);
```

Where `contactdata` is defined as:

```cpp
struct contactdata { /*!\ You must never allocate directly this structure. 
     *\ Its sizeof() is subject to change without notice. */ 
     *\ The only supported way to obtain it is via the 
     *\ FindBy* functions. */
     std::string nick; /*!< The alias of the server */
     std::string host; /*!< The hostname of the server */
     std::string contact; /*!< The subject of the server’s certificate */
     std::string vo; /*!< The VO served by this server */
     int port; /*!< The port on which the server is listening */
};
```

This function looks in the vomses files loaded by `vomsdata::LoadSystemContacts()` and `vomsdata::LoadUserContacts()` for servers that have been registered with a particular alias.

**alias**  The alias that will be searched for. The search will be case sensitive.

**RETURNS**

The return value is a vector containing the data (in `contactdata` format) of all the servers known by the system that go by the specified alias. This function does not have an error code, but the vector may be empty if no servers satisfying the query are found or if there are no known servers altogether, typically because the `Load*Contacts()` function have not been called.

11.3.7  VOID VOMSDATA::ORDER(STD::STRING ATTRIBUTE)

```cpp
void vomsdata::Order(std::string attribute);
```

This function should be called before the various `Contact*()` ones, and it is used to specify in which order the clients would like to have the attributes returned by the server.
It can be called multiple times, each time specifying a new attribute, creating in this way an ordered list of attributes. Then, when the server is contacted, it will examine this list of attributes against the one it would grant the client, and order the latter in the same way, with the following provisions:

- All attributes not explicitly indicated in the order list will be placed in an unspecified order after all the specified ones.
- An attribute present in the order list but not present among the attributes that the server is prepared to grant will be silently ignored.

attribute The attribute that should be ordered

Compatibility

Both FQAN (e.g: /group/Role=role) and legacy (e.g: <group name>:<role name>) formats are supported. However, please note that older versions may not be able to handle the FQAN format.

SEE ALSO
ResetOrder

11.3.8 void vomsdata::ResetOrder(VOID)
void vomsdata::ResetOrder(void);

This function clears the list of attributes that has been setup via calls to the Order() function. SEE ALSO
Order

11.3.9 void vomsdata::AddTarget(STD::STRING TARGET)
void vomsdata::AddTarget(std::string target);

This function takes advantage of ACs capability to target themselves to a specific set of hosts. Through consecutive calls of this function, the user can target the AC that the server will generate to any set of hosts it likes. Obviously, this function should be called before the Contact*() ones.

target The name of the host to which the AC will be targeted. The name MUST be expressed in Fully Qualified Host Name format. SEE ALSO
ListTargets, ResetTargets

11.3.10 STD::VECTOR<STD::STRING> vomsdata::ListTargets(VOID)
std::vector<std::string> vomsdata::ListTargets(void);

function returns a vector containing the list of hosts that will constitute the targets that will be include in the AC.

RETURNS
A vector whose members are the FQHNs of the machines against which the AC will be targeted. This may be empty if the list has been cleared or it has never been filled.

SEE ALSO
AddTarget, ResetTargets

11.3.11 void vomsdata::ResetTargets(VOID)
void vomsdata::ResetTargets(void);
This function clears the list of targets for an AC. SEE ALSO AddTarget, ListTargets

11.3.12 std::string vomsdata::ServerErrors()

std::string vomsdata::ServerErrors();

In case one of the other functions returned a VERR_SERVER message, meaning that some error has occurred on the server side of a connection, calling this function MAY return a message from the server itself detailing the error.

RETURNS
The error message itself

11.3.13 void vomsdata::SetVerificationType(verify_type how)

void vomsdata::SetVerificationType(verify_type how);

This function sets the type of AC verification done by the Retrieve() and Contact() functions. The choices are detailed in the verify_type type.

enum verify_type {
    VERIFY_FULL = 0xffffffff, 
    VERIFY_NONE = 0x00000000, 
    VERIFY_DATE = 0x00000001, 
    VERIFY_TARGET = 0x00000002, 
    VERIFY_KEY = 0x00000004, 
    VERIFY_SIGN = 0x00000008, 
    VERIFY_ORDER = 0x00000010, 
    VERIFY_ID = 0x00000020, 
    VERIFY_CERTLIST = 0x00000040
};

The meaning of these types is the following:

VERIFY_DATE This flag verifies that the current date is within the limits specified by the AC itself.

VERIFY_TARGET This flag verifies that the AC is being evaluated in a machine that is included in the target extension of the AC itself.

VERIFY_KEY This flag is for a future extension and is unused at the moment.

VERIFY_SIGN This flag verifies that the signature of the AC is correct.

VERIFY_ORDER This flag verifies that the attributes present in the AC are in the exact order that was requested. Please note that this can ONLY be done when examining an AC right after generation with the Contact() function. This flag is meaningless in all other cases.

VERIFY_ID This flag verifies that the holder information present in the AC is consistent with:

1. The enveloping user proxy in case the AC was contained in one. 
2. The user’s own certificate in case the AC was received without an enclosing proxy.

VERIFY_CERTLIST This flag verifies the issuer certificates included with the AC, if present.

VERIFY_FULL This flag implies all other verifications.

VERIFY_NONE This flag disables all verifications.
These flags can be combined by OR-ing them together. However, if VERIFY_NONE is OR-ed to any other flag, it can be dismissed, while if VERIFY_FULL is OR-ed to any other flag, all other flags can be dismissed.

If this function is not explicitly called by the user, a VERIFY_FULL flag is considered to be in effect.

11.3.14 **VOID VOMSDATA::SETLIFETIME(INT LIFETIME)**

```c
void vomsdata::SetLifetime(int lifetime);
```

This function should be called before the Contact*() ones. Its aim is to set the requested lifetime for the AC that the server would create. Please note that this is only a suggestion, and that the server may well override it if the requested time is against its own policy.

`lifetime` The requested lifetime, in seconds.

11.3.15 **BOOL VOMSDATA::RETRIEVE(*)**

```c
bool vomsdata::Retrieve(X509 *cert, STACK_OF(X509) *chain, recurse_type how = RECURSE_CHAIN);
bool RetrieveFromCtx(gss_ctx_id_t context, recurse_type how);
bool RetrieveFromCred(gss_cred_id_t credential, recurse_type how);
bool RetrieveFromProxy(recurse_type how);
```

This set of functions retrieve a VOMS AC from a VOMS-enabled credential, executes the verifications requested by the SetVerificationType() function and interprets the data.

Each function retrieves the AC from their corresponding credential type. RetrieveFromProxy() retrieves the credential from an existing proxy.

`cert` This is the X509 proxy certificate from which we want to retrieve the informations.

`chain` This is the certificate chain associated to the proxy certificate. This parameter is only significant if the value of the next parameter is RECURSE_CHAIN.

`context` A GSS context. This is created in a server when a GSS authenticated connection occurs and includes the other side’s credentials.

`credential` A GSS credential.

`ext` An X509 extension, which should be the extension used by VOMS to include the AC.

`how` This parameters may have two values:

**RECURSE_NONE** meaning that the VOMS extension MUST be found in the certificate proper, or

**RECURSE_CHAIN** meaning that if the VOMS extension are not found in the certificate proper, the certificate chain may be descended until either the extension is found or the chain ends.

The default value is RECURSE_CHAIN.

RECURSE_NONE should only be used in special circumstances, since it is guaranteed that in a normal Grid environment the process of credential delegation will make the VOMS extension to be only present in the certificate chain.

The result value is a boolean that is true if and only if there have not been errors. If the value is false, then you should check the error code, which may have one of the following values:
VERR_PARAM There was something wrong with the parameters passed to the function, or some of the required information (holder, etc...) is empty.

VERR_FORMAT If the format of the data is unknown (e.g. neither an AC nor an old-style blob.

VERR_NOIDENT If it was impossible to discover the holder of the AC. This may also be returned by one of the RetrieveFrom*() methods if the corresponding credential could not be found.

VERR_NOINIT The vomsdata object hasn’t been properly initialized. Most likely the voms_dir and ca_dir parameters are empty.

VERR_PARSE There has been some problem in parsing the AC or blob.

VERR_VERIFY It was not possible to verify the signature.

VERR_SERVER It was not possible to properly identify the Attribute Issuer.

VERR_TIME The check on the validity dates failed.

VERR_IDCHECK The holder of the AC is not the same entity as the holder of the enclosing certificate.

VERR_NOTAVAIL This may be returned from RetrieveFromCred(), RetrieveFromCtx or RetrieveFromProxy() if the version of the library linked is the one without Globus dependencies.

SEE ALSO
SetVerificationType()

11.3.16 bool vomsdata::Contact(std::string hostname, int port, std::string servsubject, std::string command)

This function is used to contact a specified server and use the received AC to fill the vomsdata structure.

hostname The fully qualified hostname of the machine on which the server runs.

port The port number on which the server is listening.

servsubject The subject of the server’s certificate.

command The command to be sent to the server.

These parameters may be obtained by using the FindByAlias() and FindByVO() methods.

RETURNS
The return value is true if everything went well, false otherwise. In the latter case, the error field becomes significant, and it may assume the following values.
VERR_NOSOCKET The client was unable to connect to the server.
VERR_COMM Some communication errors (Usually related to certificate problems)
VERR_SERVERCODE The server returned an error code. More detailed information may be obtained by the ServeError() function.
VERR_PARAM There was something wrong with the parameters passed to the function, or some of the required information (holder, etc...) is empty.
VERR_FORMAT If the format of the data is unknown (e.g. neither an AC nor an old-style blob.
VERR_NOIDENT If it was impossible to discover the holder of the AC or the client was unable to find its own proxy certificate.
VERR_NOINIT The vomsdata object hasn’t been properly initialized. Most likely the voms_dir and ca_dir parameters are empty.
VERRPARSE There has been some problem in parsing the AC or blob.
VERR_VERIFY It was not possible to verify the signature.
VERR_SERVER It was not possible to properly identify the Attribute Issuer.
VERR_TIME The check on the validity dates failed.
VERR_IDCHECK The holder of the AC is not the same entity as the holder of the enclosing certificate.

SEE ALSO
FindByAlias, FindByVO

11.3.17 bool ContactRaw(std::string hostname, int port, std::string servsubject, std::string command, std::string &raw, int &version)

This function is used to contact a specified server and use the received AC to fill the vomsdata structure.

hostname The fully qualified hostname of the machine on which the server runs.
port The port number on which the server is listening.
servsubject The subject of the server’s certificate.
command The command to be sent to the server.
raw This is an output parameter, and it will contain the data received by the server.
version This, too, is an output parameter, and it will contain the version number of the data included.

The first four parameters may be obtained by using the FindByAlias() and FindByVO() methods.

RETURNS
The return value is true if everything went well, false otherwise. In the latter case, the error field becomes significant, and it may assume the following values.
VERR_NOSOCKET The client was unable to connect to the server.
VERR_COMM Some communication error (Usually related to certificate problems)
VERR_SERVERCODE The server returned an error code. More detailed information may be obtained by the ServeError() function.
VERR_PARAM There was something wrong with the parameters passed to the function, or some of the required information (holder, etc...) is empty.
VERR_FORMAT If the format of the data is unknown (e.g. neither an AC nor an old-style blob.
VERR_NOIDENT If the client was unable to find its own proxy certificate.
VERR_NOINIT The vomsdata object hasn’t been properly initialized. Most likely the voms_dir and ca_dir parameters are empty.

SEE ALSO
FindByAlias, FindByVO

11.3.18 BOOL EXPORT(STD::STRING &DATA)

This function is used to create a string representation of all the data that has been read from VOMS certificates so far.
data This is an output parameter, and it will contain the data in encoded format.

RETURNS
The return value is true if everything went well, false otherwise. In the latter case, the error field becomes significant, and it may assume the following values.

VERR_MEM There is not enough memory free.
VERR_FORMAT There is an inconsistency in the internal data.
VERR_TYPE The same as above. The difference is only for debugging purposes.

SEE ALSO
Import()

11.3.19 BOOL IMPORT(STD::STRING BUFFER)

This function is used to add a string created by the Export() call back into the vomsdata structure. This function also runs verification again.
buffer The string to convert.

RETURNS
The return value is true if everything went well, false otherwise. In the latter case, the error field becomes significant, and it may assume the following values:
VERR_PARAM There was something wrong with the parameters passes to the function, or some of the required information (holder, etc...) is empty.

VERR_FORMAT If the format of the data is unknown (e.g. neither an AC nor an old-style blob.

VERR_NOIDENT If it was impossible to discover the holder of the AC or there was not a user certificate ready.

VERR_NOINIT The vomsdata object hasn’t been properly initialized. Most likely the voms_dir and ca_dir parameters are empty.

VERR_PARSE There has been some problem in parsing the AC or blob.

VERR_VERIFY It was not possible to verify the signature.

VERR_SERVER It was not possible to properly identify the Attribute Issuer.

VERR_TIME The check on the validity dates failed.

VERR_IDCHECK The holder of the AC is not the same entity as the holder of the enclosing certificate.

11.3.20 bool vomsdata::DefaultData(voms &d)

This function returns the default attributes from a vomsdata class.

d This is the voms structure that will contain the default attributes.

RETURNS
The return value is true if everything went well, false otherwise. In the latter case, the error field becomes significant, and it may assume the following values:

VERR_NOEXT If there was no default attributes (most likely because no attributes were read in.

11.3.21 std::string vomsdata::ErrorMessage(void)

std::string vomsdata::ErrorMessage(void);

This function returns the error message associated with the error code returned by the latest executed method.

RETURNS
The error message itself. Please note that this is completely meaningless if the latest method did not return an error code.

12 C API

There are three basic data structures: data, voms and vomsdata.

12.1 The data structure

The first one, data contains the data regarding a single attribute, giving its specification in terms of Groups, Roles and Capabilities. It is defined as follows:

```c
struct data {
    char *group;
    char *role;
```

char *cap;
};

All the values of these strings must be composed from regular expression: a-zA-Z0-9/]*.

12.1.1 GROUP

This field contains the name of a group into which the user belongs. The format of entries in this group is reminiscent of the structure of pathnames, and is the following:

/group/group/.../group

where the name of the first group is by convention the name of the Virtual Organization (VO), while each other /group component is a subgroup of the group immediately preceding it on the left. The character '/' is not acceptable as part of a group name.

This field MUST always be filled.

12.1.2 ROLE

This field contains the name of the role to which the user owns in the group specified by group. If the user does not own any particular role in that group, than this field contains the value “NULL”.

12.1.3 CAP

This field details a capability that the user has as a member of the group specified by group while owning the role specified by role. If there is no specific capability, than this value is “NULL”.

No specific format is associated to a capability. They are basically free-form strings, whose value should be agreed between the AA and the Attribute verifier.

12.2 THE VOMS STRUCTURE

The second one, voms is used to group together all the informations that can be gleaned from a single AC, and is defined as follows:

```c
#define TYPE_NODATA 0 /*!< no data */
#define TYPE_STD 1 /*!< group, role, capability triplet */
#define TYPE_CUSTOM 2 /*!< result of an S command */

struct voms {
  int siglen;
  char *signature;
  char *user;
  char *userca;
  char *server;
  char *serverca;
  char *voname;
  char *uri;
  char *date1;
  char *date2;
};
```

INFSO-RI-508833 PUBLIC 53/122
int type;
struct data **std;
char *custom;
int datalen;
int version;
char **fqan;
char *serial;
/* Fields below this line are reserved. */
}

The purpose of this structure is to present, in a readable format, the data that has been included in a single Attribute Certificate (AC). While the various public fields may be freely modified to simplify internal coding, such changes have no effect on the underlying AC. Let’s examine the various fields in detail, starting with the constructors.

### 12.2.1 VERSION

This field specifies the version of this structure that is currently being used. A value of 0 indicates that it comes from an old format extension, while a value of 1 indicates that this structure comes from an AC.

**Compatibility**

As of VOMS 1.6.0, version 0 is no longer supported. What this means is that VOMS 1.6.0 or later is no longer compatible with VOMS 1.1.x or earlier.

Please do note that modifying the fields of a version 0 structure associated with a `versiondata struct` invalidates the result of the `VOMS_Export()` function on that object.

### 12.2.2 SIGLEN

The length of the data signature.

### 12.2.3 USER

This field contains the subject of the holder’s certificate in slash-separated format.

### 12.2.4 USERCA

This field contains the subject of the CA that issued the holder’s certificate, in slash-separated format.

### 12.2.5 SERVER

This field contains the subject of the certificate that the AA used to issue the AC, in slash-separated format.

### 12.2.6 SERVERCA

This field contains, in slash-separated format, the subject of the CA that issued the certificate that the AA used to issue the AC.
12.2.7 VONAME

This field contains the name of the Virtual Organization (VO) to which the rest of the data contained in this structure applies.

12.2.8 URI

This is the URI at which the AA that issued this particular AC can be contacted. Its format is:

fqdn:port

where \textit{fqdn} is the Fully Qualified Domain Name of the server which hosts the AA, while \textit{port} is the port at which the AA can be contacted on that server.

12.2.9 DATE1, DATE2

These are the dates of start and end of validity of the rest of the informations. They are in a string representation readable to humans, but they may be easily converted back to their original format.

Here follows a code example doing that conversion:

```
ASN1_TIME *
convtime(char *data)
{
    char *data2 = strdup(data);

    if (data2) {
        ASN1_TIME *t = ASN1_TIME_new();

        t->data = (unsigned char *)data2;
        t->length = strlen(data);
        switch(t->length) {
            case 15:
                t->type = V_ASN1_GENERALIZEDTIME;
                break;
            default:
                ASN1_TIME_free(t);
                return NULL;
        }
        return t;
    }
    return NULL;
}
```

12.2.10 TYPE

This datum specifies the type of data that follows. It can assume the following values:

\textbf{TYPE\_NODATA} There actually was no data returned.
Compatibility

This is actually only true for version 0 structures. The following versions will simply not generate a `voms` structure in this case.

**TYPE_CUSTOM** The data will contain the output of an “S” command sent to the server.

Compatibility

Again, this type of datum will only be present in version 0 structures. Due to lack of use, support for it has been disabled in new versions of the server.

**TYPE_STD** The data will contain (group, role, capabilities) triples.

### 12.2.11 STD

This vector contains all the attributes found in an AC, in the exact same order in which they were found, in the format specified by the `data` structure. It is only filled if the value of the `type` field is `TYPE_STD`.

### 12.2.12 CUSTOM

This field contains the data returned by the “S” server command, and it is only filled if the `type` value is `TYPE_CUSTOM`.

### 12.2.13 FQAN

This field contains the same data as the `std` field, but specified in the Fully Qualified Attribute Name (FQAN) format.

### 12.2.14 VOMSDATA

The purpose of this object is to collect in a single place all informations present in a VOMS extension. All the fields should be considered read-only. Changing them has indefinite results.

```c
struct vomsdata {
    char *cdir;
    char *vdir;
    struct voms **data;
    char *workvo;
    char *extra_data;
    int volen;
    int extralen;
    /* Fields below this line are reserved. */
};
```

Let us see the fields in detail.

### 12.2.15 DATA

This field contains a vector of `voms` structures, in the exact same order as the corresponding ACs appeared in the proxy certificate, and containing the informations present in that AC.
12.2.16 WORKVO, VOLEN

Compatibility

This field is obsolete in the current version. Expect workvo to be set to NULL and volen to be set to 0.

12.2.17 EXTRA_DATA, EXTRALEN

This field contains additional data that has been added by the user via to the proxy via the --include command line option. extralen represents the length of that data.

12.2.18 CDIR, VDIR

This fields contain the paths, respectively, of the CA certificates and of the VOMS servers certificates.

12.3 FUNCTIONS

12.3.1 Generalities

Most of these functions share two parameters, struct vomsdata *vd, and int *error. To avoid repetition, these two parameters are described here.

error  This field contains the error code returned by one of the methods. Please note that the value of this field is only significant if the last method called returns an error value. Also, the value of this field is subject to change without notice during method executions, regardless of whether an error effectively occurred.

The possible values returned are: VERR_NONE, VERR_NOSOCKET, VERR_NOIDENT, VERR_COMM, VERR_PARAM, VERR_NOEXIT, VERR_NOINIT, VERR_TIME, VERR_IDCHECK, VERR_EXTRAINFO, VERR_FORMAT, VERR_NODATA, VERR_PARSE, VERR_DIR, VERR_SIGN, VERR_SERVER, VERR_MEM, VERR_VERIFY, VERR_TYPE, VERR_ORDER, VERR_SERVERCODE, VERR_NOTAVAL

In general, a first idea of what each code means can be gleaned from the code name, but in any case every method description will document which errors its execution may generate and on which conditions.

vd  This parameter is a pointer to the vomsdata structure that should be used by the function for both configuration and data retrieval and also for data storage.

12.3.2 STRUCT CONTACTDATA **VOMS_FindByAlias(struct vomsdata *vd, char *alias, char *SYSTEM, char *USER, int *ERROR)

struct contactdata **VOMS\_FindByAlias(struct vomsdata *vd, char *alias, char *system, char *user, int *error);

struct contactdata **VOMS\_FindByVO(struct vomsdata *vd, char *vo, char *system, char *user, int *error);

These two functions look in the vomses files installed in both the system-wide and user-specific directories for servers that have been registered with a particular alias or VO respectively.

The contactdata structure is defined below:

```c
struct contactdata { /*!< You must never allocate directly this structure. Its sizeof() is subject to change without notice. The only supported way to obtain it is via the VOMS_FindBy* functions. */
    char *nick; /*!< The alias of the server */
};
```
struct contactdata *host; /*!< The hostname of the server */
char *contact; /*!< The subject of the server's certificate */
char *vo; /*!< The VO served by this server */
int port; /*!< The port on which the server is listening */
char *reserved; /*!< HANDS OFF! */
};

alias The alias that will be searched for. The search will be case sensitive.
vo The vo that will be searched for. The search will be case sensitive.
system The directory where the system-wide files are located. If NULL then its default is /opt/glite/etc/vomses.
user The directory where the user-specific files are stored. If this field is NULL, then the default of $VOMS_USERCONF is used. If this is also empty, then the default become $HOME. RETURNS The return value is a NULL-terminated vector containing the data (in contactdata format) of all the servers known by the system that go by the specified alias. This may be NULL if there was an error or no server was found registered with the specified alias.
The errors that you may find are:

    VERR_MEM Not enough memory.
    VERR_DIR There were some problems while traversing the directory.
    VERR_NONE No error occurred. Simply, no servers were found.

12.3.3 VOID VOMS_DELETECONTACTS(STRUCT CONTACTDATA **LIST)

This function deletes a vector of server data returned by either the VOMS_FindByAlias() or the VOMS_FindByVO() functions. This is the only supported way to deallocate the vector. Any other attempt will result in undefined behavior.

It is although possible to deallocate only part of a vector. See the following code for an example.

    /*
    * Supposing that v is a vector returned by one of the VOMS_FindBy*() functions. Also suppose that n is the vector's size (including the NULL ending element).
    * The following snippet will delete just the first member.
    */
    struct contactdata *dummy[2];

dummy[1] = NULL;
dummy[0] = v[0];
v[0] = v[n-1];
v[n-1] = NULL;
VOMS_DeleteContacts(dummy);

list The data to be deleted.

RETURNS None.
12.3.4 struct vomsdata *VOMS_Init(char *voms, char *cert)

This function allocates and initializes a vomsdata structure. This is the only way to do so. Trying to allocate a vomsdata structure by any other way will trigger undefined behavior, since the structure that is published is only a small part of the real one.

voms  The directory that contains the certificates of the VOMS servers. If this value is NULL, then $X509_VOMS_DIR is considered. If this is also empty than its default is .
cert  The directory that contains the certificates of the CAs recognized by the server. If this value is NULL, then $X509_CERT_DIR is considered. If this is also empty than its default is: “/oetc/grid-security/certificates”.

RETURNS
A pointer to a properly initialized vomsdata structure, or NULL if something went wrong. This is the only case in which an error code would no be associated to the function.

The default values are strongly suggested. If you want to hardcode specific ones, think very hard about the less of configurability that it would entail.

12.3.5 struct voms *VOMS_Copy(struct voms *, int *error)

This function duplicates an existing voms structure. It is the only supported way to do so.

voms  The voms structure that you wish to be duplicated.

RETURNS
A pointer to a voms structure that duplicates the content of the one you passed, or NULL if something went wrong.

ERRORS
VERR_MEM  Not enough memory.

12.3.6 struct vomsdata *VOMS_CopyAll(struct vomsdata *vd, int *error)

This function duplicates an existing vomsdata structure. It is the only supported way to do so.

RESULTS
A pointer to a voms structure that duplicates the content of the one you passed, or NULL if something went wrong.

ERRORS
VERR_MEM  Not enough memory.

12.3.7 void VOMS_Delete(struct voms *v)

This functions deletes an existing voms structure. It is the ONLY supported way to do so.
v  A pointer to the voms structure to delete. It is safe to call this function with a NULL pointer.

RESULTS
None.

12.3.8 int VOMS_AddTarget(struct vomsdataa *vd, char *target, int *error)

This function adds a target to the target list for the AC that will be generated by a server when it will be contacted by the VOMS_Contact*() function.
target  The target to add. It should be a Fully Qualified Domain Name.
RESULTS

0 If something went wrong.
<>0 Otherwise.

ERRORS
VERR_NOINIT The vomsdata structure was not properly initialized.
VERR_PARAM The target parameter was NULL.
VERR_MEM There was not enough memory.

12.3.9 void VOMS_FreeTargets(STRUCT vomsdata *vd, INT *ERROR)
This function resets the list of targets. It always succeeds. It is also safe to call this function when targets have been set.

12.3.10 CHAR *VOMS_ListTargets(STRUCT vomsdata *vd, INT *ERROR)
This function returns a comma separated string containing all the targets that have been set by the VOMS_AddTarget() function. The caller is the owner of the returned string, and is responsible for calling free() over it when he no longer needs it.

RESULTS
A string with the result, or NULL.

VERR_NOINIT The vomsdata structure was not properly initialized.
VERR_MEM There was not enough memory.

12.3.11 INT VOMS_SetVerificationType(INT type, STRUCT vomsdata *vd, INT *ERROR)
This function sets the type of AC verification done by the VOMS_Retrieve() and Contact() functions. The choices are detailed in the verify_type type.

#define VERIFY_FULL 0xffffffff
#define VERIFY_NONE 0x00000000
#define VERIFY_DATE 0x00000001
#define VERIFY_NOTARGET 0x00000002
#define VERIFY_KEY 0x00000004
#define VERIFY_SIGN 0x00000008
#define VERIFY_ORDER 0x00000010
#define VERIFY_ID 0x00000020
#define VERIFY_CERTLIST 0x00000040

The meaning of these types is the following:

VERIFY_DATE This flag verifies that the current date is within the limits specified by the AC itself.

VERIFY_TARGET This flag verifies that the AC is being evaluated in a machine that is included in the target extension of the AC itself.
VERIFY_KEY  This flag is for a future extension and is unused at the moment.

VERIFY_SIGN  This flag verifies that the signature of the AC is correct.

VERIFY_ORDER  This flag verifies that the attributes present in the AC are in the exact order that was requested. Please note that this can ONLY be done when examining an AC right after generation with the Contact() function. This flag is meaningless in all other cases.

VERIFY_ID  This flag verifies that the holder information present in the AC is consistent with:

1. The enveloping user proxy in case the AC was contained in one.
2. The user’s own certificate in case the AC was received without an enclosing proxy.

VERIFY_CERTLIST  This flag verifies the issuer’s certificate included in the AC, if present.

VERIFY_FULL  This flag implies all other verifications.

VERIFY_NONE  This flag disables all verifications.

These flags can be combined by OR-ing them together. However, if VERIFY_NONE is OR-ed to any other flag, it can be dismissed, while if VERIFY_FULL is OR-ed to any other flag, all other flags can be dismissed.

If this function is not explicitly called by the user, a VERIFY_FULL flag is in effect.

RESULTS

0  If there is an error.

<> 0  otherwise.

VERR_NOINIT  The vomsdata structure was not properly initialized.

12.3.12  int VOMS_SetLifetime(int length, struct vomsdata *vd, int *error)

int VOMS\_SetLifetime(int length, struct vomsdata *vd, int *error);

This function sets the requested lifetime for ACs that would be generated as the result of a VOMS\_Contact() or VOMS\_ContactRaw() request. Note however that this is only an hint sent to the server, since it can lower it at will if the requested length is against server policy.

length  The lifetime requested, measured in seconds.

RESULTS

0  If there is an error.

<> 0  otherwise.

VERR_NOINIT  The vomsdata structure was not properly initialized.

SEE ALSO

VOMS\_Contact(), VOMS\_ContactRaw()
12.3.13  **void VOMS_Destroy(struct vomsdata *vd)**

```c
void VOMS_Destroy(struct vomsdata *vd);
```

This function destroys an allocated vomsdata structure. It is the *only* supported way to do so. It is also safe to pass a NULL pointer to it.

**RESULTS**

None.

12.3.14  **int VOMS_Ordering(char *order, struct vomsdata *vd, int *error)**

```c
int VOMS_Ordering(char *order, struct vomsdata *vd, int *error);
```

This function is used to request a specific ordering of the attributes present in an AC returned by the VOMS_Contact() or by the VOMS_ContactRaw() functions.

This function can be called several times, each time specifying a new attribute. The attributes in th AC created by the server will be in the same order as the calls to this function, ignoring attributes specified by this function that the server does not wish to grant. Attributes not explicitly specified in this list will be inserted, in an unspecified order, after all the others.

Never calling this function means that the corresponding list will be empty, and as a consequence all the attributes will be in an unspecified ordering.

**order** The name of an attribute, in either the FQAN format (i.e: /group/Role=role) or in the &lt;group&gt;:<&lt;role&gt;: format.

**Compatibility**

Not all versions of the APIs support the FQAN format yet.

**RESULTS**

0  If there is an error.

< > 0  otherwise.

**ERRORS**

- VERR_NOINIT  The vomsdata structure was not properly initialized.
- VERR_PARAM  The order parameter is NULL.
- VERR_MEM  There is not enough memory.

**SEE ALSO**

VOMS_ResetOrder(), VOMS_Contact(), VOMS_ContactRaw()

12.3.15  **int VOMS_ResetOrder(struct vomsdata *cd, int *error)**

```c
int VOMS_ResetOrder(struct vomsdata *cd, int *error);
```

This function resets the attribute ordering set by the VOMS_Ordering function.

**RESULTS**

0  If there is an error.

< > 0  otherwise.
VERR_NOINIT The vomsdata structure was not properly initialized.

SEE ALSO
VOMS_Ordering()

12.3.16 int VOMS_CONTACT(char *hostname, int port, char *servsubject, char *command, struct vomsdata *vd, int *error)

This function is used to contact a VOMS server to receive an AC containing the calling user’s authorization informations. A prerequisite to calling this function is the existence of a valid proxy for the user himself. This function does not create such a proxy, which then must already exist. Also, the parameters needed to call this function should have been obtained by calling one of VOMS_FindByAlias() or VOMS_FindByVO().

hostname This is the hostname of the machine hosting the server.
port This is the port number on which the server is listening.
servsubject This is the subject of the VOMS server’s certificate. This is needed for the mutual authentication.
command This is the command to be sent to the server. For more info about it, consult the voms-proxy-init() manual.

RESULTS

0 If there is an error.

<0 If there is an error.

\geq 0 otherwise. Furthermore, the data returned by the server has been parsed and added to the vomsdata structure.

ERRORS

VERR_NOINIT If the vomsdata structure was not properly initialized.

VERR_NOSOCKET If it was impossible to contact the server.

VERR_MEM If there was not enough memory.

VERR_IDCHECK If a proxy certificate was not found or the data returned by the server did not contain identifying information.

VERR_FORMAT If there was an error in the format of the data received.

VERR_NODATA If no data was received at all. (Usually as a consequence of either a server error or not being recognized by the server as a valid user.)

VERR_ORDER If the attribute that the client requested, via the VOMS_Ordering() function, to be first in the list of attributes received is not first in the attributes returned by the server. This particular code means that the data has been correctly interpreted and is available in the vomsdata structure if you want to use it.

VERR_SERVERCODE Some strange error occurred in the server.

SEE ALSO
VOMS_FindByVO(), VOMS_FindByAlias()
12.3.17  

```c
int VOMS_ContactRaw(char *hostname, int port, char *servsubject, char *command, void **data, int *datalen, int *version, struct vomsdata *vd, int *error)
```

This function, like `VOMS_Contact()` can be used to contact a server and receive Authorization info from it. The difference between the two functions is that this version does not interpret the raw data, but on the contrary returns it to the caller. This function has all the same prerequisites as `VOMS_Contact()`.

**hostname** This is the hostname of the machine hosting the server.

**port** This is the port number on which the server is listening.

**servsubject** This is the subject of the VOMS server’s certificate. This is needed for the mutual authentication.

**command** This is the command to be sent to the server. For more info about it, consult the voms-proxy-init() manual.

**data** A pointer to a pointer to an area of memory where the data returned from the server is stored. It is the caller’s responsibility to free() this memory when it is no longer useful.

**datalen** The length of the data returned.

**version** The version of the AC returned. Note that this is a minimum version, it only guarantees that the data is at least in that version of the format.

**RESULTS**

0  If there is an error.

<>0  otherwise. Furthermore, the data returned by the server has been parsed and added to the `vomsdata` structure.

**ERRORS**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERR_NOINIT</td>
<td>If the vomsdata structure was not properly initialized.</td>
</tr>
<tr>
<td>VERR_NOSOCKET</td>
<td>If it was impossible to contact the server.</td>
</tr>
<tr>
<td>VERR_MEM</td>
<td>If there was not enough memory.</td>
</tr>
<tr>
<td>VERR_IDCHECK</td>
<td>If a proxy certificate was not found or the data returned by the server did not contain identifying information.</td>
</tr>
<tr>
<td>VERR_FORMAT</td>
<td>If there was an error in the format of the data received.</td>
</tr>
<tr>
<td>VERR_NODATA</td>
<td>If no data was received at all. (Usually as a consequence of either a server error or not being recognized by the server as a valid user.)</td>
</tr>
<tr>
<td>VERR_ORDER</td>
<td>If the attribute that the client requested, via the VOMS_Ordering() function, to be first in the list of attributes received is not first in the attributes returned by the server. This particular code means that the data has been correctly interpreted and is available in the vomsdata structure if you want to use it.</td>
</tr>
<tr>
<td>VERR_SERVERCODE</td>
<td>Some strange error occurred in the server.</td>
</tr>
</tbody>
</table>

12.3.18  

```c
int VOMS_Retrieve(X509 *cert, STACK_OF(X509) *chain, int how, struct vomsdata *vd, int *error)
```

```c
int VOMS\_Retrieve(X509 *cert, STACK\_OF(X509) *chain, int how, struct vomsdata
```
int VOMS\_RetrieveEXT(X509\_Extension *ext, struct vomsdata *vd, int *error);

int VOMS\_RetrieveFromCred(gss\_cred\_id\_t cred, int how, struct vomsdata *vd, int *error);

int VOMS\_RetrieveFromCtx(gss\_ctx\_id\_t cred, int how, struct vomsdata *vd, int *error);

int VOMS\_RetrieveFromProxy(int how, struct vomsdata *vd, int *error);

This functions are used to extract from a credential the VOMS-specific extension, to parse them and to insert the results into the `vomsdata` structure. Each variant of it extracts information from their specific type of credential.

cert This is the certificate that contains the VOMS information. No checks are done on the validity of this certificate, that is supposed to have already been verified by some other means.

chain This is the chain of certificates that signed the cert certificate. This pointer may be null, but see the next parameter.

context A GSS context. This is created in a server when a GSS authenticated connection occurs and includes the other side’s credentials.

credential A GSS credential.

ext An X509 extension, which should be the extension used by VOMS to include the AC.

how This parameter indicates how the search for the VOMS info will be performed. If RECURSE\_CHAIN then the information is searched first into the cert and then, if it was not found, in the walking the chain, from the certificates to the CA. If RECURSE\_NONE is specified, then the information is only searched in the cert. In case the first value is specified, then the searches stop as soon as the info is found, ignoring further extension that may be found down the chain.

RESULTS

0 If there is an error.

<>0 otherwise. Furthermore, the data returned by the server has been parsed and added to the `vomsdata` structure.

ERRORS

VERR\_NOCERT If the vomsdata structure was not properly initialized.

VERR\_PARAM If there is something wrong with one of the parameters.

VERR\_MEM If there was not enough memory.

VERR\_IDCHECK If a proxy certificate was not found or the data returned by the server did not contain identifying information.

VERR\_FORMAT If there was an error in the format of the data received.

VERR\_NOEXT If the extension was not found.

VERR\_NOTAVAIL If the `RetrieveFromCred()`, `RetrieveFromCtx()` or `RetrieveFromProxy()` functions were used with the version of the libraries which add globus support disabled.

12.3.19 int VOMS\_IMPORT(CHAR*buffer, INT buflen, STRUCT vomsdata *vd, INT *error)

This function is used to add a string created with `VOMS\_Export()` back into the vomsdata structure.

buffer A pointer to the string.

buflen The length of the string.
RESULTS

0 If there is an error.
< >0 otherwise. Furthermore, the data returned by the server has been parsed and added to the vomsdata structure.

ERRORS
VERR_NOINIT If the vomsdata structure was not properly initialized.
VERR_FORMAT If there was an error in the format of the data received.
VERR_PARAM If there is something wrong with one of the parameters.
VERR_MEM If there was not enough memory.
VERR_IDCHECK If a proxy certificate was not found or the data returned by the server did not contain identifying information.
VERR_SERVER The VOMS server was unidentifiable.
VERR_PARSE There has been some problem in parsing the AC or blob.
VERR_SIGN It was not possible to verify the signature.
VERR_SERVER It was not possible to properly identify the Attribute Issuer.
VERR_TIME The check on the validity dates failed.

12.3.20  
int VOMS_Export(char **buffer, int *bufflen, struct vomsdata *vd, int *error)

This function will take the current vomsdata structure and encode it in a string that can then be exported.

buffer A pointer to an area of memory that will be allocated and filled by the function. It is the caller’s responsibility to free() this memory. It is possible that this pointer will be set to NULL, in case the vomsdata structure is empty.

bufflen The size of the data pointed by buffer.

RESULTS

0 If there is an error.
< >0 otherwise. Furthermore, the data returned by the server has been parsed and added to the vomsdata structure.

ERRORS
VERR_PARAM If there is something wrong with one of the parameters.
VERR_MEM If there was not enough memory.

12.3.21  
struct voms *VOMS_DefaultData(struct vomsdata *vd, int *error)

struct voms *VOMS\_DefaultData(struct vomsdata *vd, int *error);
This function returns the default attributes from a vomsdata class.

RESULTS

NULL  There has been an error or the vomsdata structure was empty.

<>NULL  There is some data.

ERRORS

VERR_NOINIT  The vomsdata structure was not properly initialized.
VERR_NONE   The vomsdata structure was empty.

12.3.22  char *VOMS_ErrorMessage(struct vomsdata *vd, int *error, char *buffer, int len)

This function returns an error message describing the error of the last function called. It is significant if and only if the last call returned an error, otherwise the returned string will be meaningless.

buffer  A pointer to a location of memory where the memory message will be written. This may be NULL, in which case the message will be written in memory newly allocated by this function.

len  The length of the previous buffer. This is only significant if buffer is not NULL.

RESULTS

A pointer to a string describing the error. If the passed buffer was NULL, then the caller is responsible for free()ing this memory when it is no longer needed.

12.3.23  int VOMS_GetAttributeSourceHandle(struct voms *v, int num, struct vomsdata *vd, int *error)

This function returns an handle to a specific attribute container.

v  The voms structure from which attributes are to be extracted.
num  The number of the desired container. See VOMS_GetAttributeSourcesNumber() to get the maximum value.

RESULTS

An handle to the container, or -1 if the specified container does not exist.

ERRORS

VERR_PARAM  If the specified container does not exist..

SEE ALSO

VOMS_GetAttributeSourcesNumber()

12.3.24  VOMS_GetAttributeSourcesNumber(struct voms *v, struct vomsdata *vd, int *error)

VOMS\_GetAttributeSourcesNumber(struct voms *v, struct vomsdata *vd, int *error)

The Generic Attributes that may be present in an AC are organized by grantor. This function returns the number of grantors.

v  The voms structure from which attributes are to be extracted.
RESULTS
The number of grantors, 0 if Generic Attributes were not present, or -1 in case of errors.

ERRORS
  VERR_PARAM  If an error is present in the format of the AC.

12.3.25 VOMS_GET_ATTRIBUTE_GRANTOR(STRUCT VOMS *V, INT HANDLE, STRUCT VOMSDATA *VD, INT *ERROR)
VOMS\_GetAttributeGrantor(struct voms *v, int handle, struct vomsdata*vd, int *error);
This function will return the name of the attributes grantor of the attribute block identified by the handle. This memory belongs to the function, and MUST NOT be tampered with.

v The voms structure from which attributes are to be extracted.
handle The handle to the attribute container as returned by VOMS_GetAttributeSourceHandle()

RESULTS
The string representing the grantor, or NULL in case of errors.

ERRORS
  VERR_PARAM  If the specified container does not exist, or another error is present in the format of the AC.

12.3.26 VOMS_GET_ATTRIBUTES_NUMBER(STRUCT VOMS *V, INT HANDLE, STRUCT VOMSDATA *VD, INT *ERROR)
VOMS\_GetAttributesNumber(struct voms *v, int handle, struct vomsdata *vd, int *error);
This function will return the number of attributes present in a container.

v The voms structure from which attributes are to be extracted.
handle The handle to the attribute container as returned by VOMS_GetAttributeSourceHandle()

RESULTS
The number of attributes, or -1 in case of errors.

ERRORS
  VERR_PARAM  If an error is present in the format of the AC.

12.3.27 VOMS_GET_ATTRIBUTE(STRUCT VOMS *V, INT HANDLE, INT NUM, STRUCT ATTRIBUTE *A, STRUCT VOMDATA *VD, INT *ERROR)
VOMS\_GetAttribute(struct voms *v, int handle, int num, struct attribute *a, struct vomdata *vd, int *error);
This function extracts the selected attribute, and fills the passed attribute structure with the relevant information.

Format of the attribute structure:
struct attribute {
    char *name;
    char *value;
    char *qualifier;
};

Where name is the name of the attribute, value is its value, and qualifier is an optional qualifier, which may be NULL if not passed.

v The voms structure from which attributes are to be extracted.
handle  The handle to the attribute container as returned by VOMS_GetAttributeSourceHandle()
num  The number of the attribute requested

OUTPUT PARAMETER: The members of this structure get filled with the corresponding values from the attribute

RESULTS
1 in case of success, 0 otherwise.

ERRORS
VERR_PARAM  If an error is present in the format of the AC.

13  JAVA APIs

Package org.glite.security.voms

Package Contents  Page

Classes

BasicVOMSTrustStore .......................................................... 61
  Deprecated! Use PKIStore instead.
FQAN ................................................................. 63
  Parses and assembles Fully Qualified Attribute Names (FQANs) used by VOMS.
LSCFile ............................................................... 64
  The job of this class is to represent a *.lsc file in the vomsdir directory.
PKIStore ............................................................ 65
  PKIStore is the class serving to store all the components of a common PKI insta-
  llation, i.e.: CA certificates, CRLs, Signing policy files...
PKIUtils .......................................................... 69

PKIVerifier .......................................................... 75

SigningPolicy ........................................................... 77
  The purpose of this class is to represent a *.signing_policy file.
VOMSAttribute .......................................................... 79
  Representation of the authorization information (VO, server address and list of
  Fully Qualified Attribute Names, or FQANs) contained in a VOMS attribute cer-
  tificate.
VOMSValidator .......................................................... 85
  The main (top) class to use for extracting VOMS information from a certificate
  and/or certificate chain.
VOMSValidator.FQANTree .................................................. 90
  Class to sort out the hierarchial properties of FQANs.

13.1  CLASS BasicVOMSTrustStore

Deprecated! Use PKIStore instead.

13.1.1  DECLARATION

public final class BasicVOMSTrustStore
extends java.lang.Object
implements org.glite.security.voms.ac.ACTrustStore
13.1.2 Field summary

DEFAULT_TRUST_STORE_LISTING

13.1.3 Constructor summary

BasicVOMSTrustStore() Creates a default VOMS trust store.
BasicVOMSTrustStore(String, long) Creates and manages an in-memory cache of VOMS issuers by periodically scanning a directory containing the trusted issuers.

13.1.4 Method summary

getAACandidate(X500Principal)
getDirList()
refresh() Refreshes the in-memory cache of trusted signer certificates.
stopRefresh()

13.1.5 Fields

• public static final java.lang.String DEFAULT_TRUST_STORE_LISTING

13.1.6 Constructors

• BasicVOMSTrustStore
  public BasicVOMSTrustStore()
  – Description
  Creates a default VOMS trust store. Equivalent to
  new BasicVOMSTrustStore(DEFAULT_TRUST_STORE_LISTING, 300000);

• BasicVOMSTrustStore
  public BasicVOMSTrustStore(java.lang.String trustedDirList, long refreshPeriod)
  – Description
  Creates and manages an in-memory cache of VOMS issuers by periodically scanning a directory containing the trusted issuers. If refreshPeriod is 0, it never refreshes.
  – Parameters
    * trustedDirList – directory listing containing trusted VOMS certs
    * refreshPeriod – refresh period in milliseconds
  – See also
    * DirectoryList

13.1.7 Methods

• getAACandidate
  java.security.cert.X509Certificate[] getAACandidate(javax.security.auth.x500.X500Principal issuer)
Description copied from ac.ACTrustStore (in 13.11, page 92)
Returns an array of issuer candidates, by performing a name comparison of the AC’s issuer and the subject names of the certificates in the trust store.

NOTE: No actual verification or validation of signature takes place in this function.

Parameters
* issuer – the principal to find an issuer for. If null, all known AAs will be returned.

Returns – an array of issuer candidates, or null in case of an error.

• getDirList
public java.lang.String getDirList()   

• refresh
public void refresh()   

– Description
Refreshes the in-memory cache of trusted signer certificates.

• stopRefresh
public void stopRefresh()   

13.2 Class FQAN

Parses and assembles Fully Qualified Attribute Names (FQANs) used by VOMS. FQANs are defined as <group>[/Role=[<role>]][/Capability=<capability>]]

13.2.1 Declaration

public class FQAN
extends java.lang.Object

13.2.2 Constructor summary

FQAN(String)
FQAN(String, String, String)

13.2.3 Method summary

equals(Object)
getCapability()
getFQAN()
getGroup()
getRole()
split()
toString()
13.2.4 Constructors

- **FQAN**
  ```java
  public FQAN(java.lang.String fqan)
  ```

- **FQAN**
  ```java
  public FQAN(java.lang.String group, java.lang.String role, java.lang.String capability)
  ```

13.2.5 Methods

- **equals**
  ```java
  public boolean equals(java.lang.Object)
  ```

- **getCapability**
  ```java
  public java.lang.String getCapability()
  ```

- **getFQAN**
  ```java
  public java.lang.String getFQAN()
  ```

- **getGroup**
  ```java
  public java.lang.String getGroup()
  ```

- **getRole**
  ```java
  public java.lang.String getRole()
  ```

- **split**
  ```java
  protected void split()
  ```

- **toString**
  ```java
  public java.lang.String toString()
  ```

13.3 Class LSCFile

The job of this class is to represent a *.lsc file in the vomsdir directory.

13.3.1 Declaration

```java
public class LSCFile
extends java.lang.Object
```

13.3.2 Constructor summary

- **LSCFile(File)** Loads a *.lsc file from a File

13.3.3 Method summary

- **getDNLists()** Returns the allowed subject/issuer DN sequences for this file.
- **getName()** Returns the basename of the file from which this was loaded.
13.3.4 Constructors

- **LSCFile**
  ```java
  public LSCFile(java.io.File f) throws java.io.IOException
  ```

  - **Description**
    Loads a *.lsc file from a File
  - **Parameters**
    - `f` – the file to load from
  - **Throws**
    - `java.io.IOException` – if there are problems loading the file.

13.3.5 Methods

- **getDNLists**
  ```java
  public java.util.Vector getDNLists()
  ```

  - **Description**
    Returns the allowed subject/issuer DN sequences for this file.
  - **Returns**
    - a vector whose elements are vectors of strings describing the exact sequences.

- **getName**
  ```java
  public java.lang.String getName()
  ```

  - **Description**
    Returns the basename of the file from which this was loaded.
  - **Returns**
    - the filename, or null if nothing was loaded.

13.4 Class PKIStore

PKIStore is the class serving to store all the components of a common PKI installation, i.e.: CA certificates, CRLs, Signing policy files... It is also capable of storing files specific to the handling of VOMS proxies, i.e. the content of the vomsdir directory.

13.4.1 Declaration

```java
public class PKIStore
extends java.lang.Object
implements org.glite.security.voms.ac.VOMSTrustStore
```

13.4.2 Field Summary

- **TYPE_CADIR** This PKIStore object will contain data from a CA directory.
- **TYPE_VOMSDIR** This PKIStore object will contain data from a vomsdir directory.

13.4.3 Constructor Summary

- `PKIStore()`  
- `PKIStore(String, int)` This is equivalent to PKIStore(dir, type, true)
- `PKIStore(String, int, boolean)`
### 13.4.4 Method Summary

- **getAACandidate(X500Principal, String)** Gets an array of candidate issuer certificates for an AC with the given issuer and belonging to the given VO.
- **getCAs()**
- **getCRLs()**
- **getLSC(String, String)** Gets the LSC file corresponding to the given VO, for the given server.
- **getSignings()**
- **load()** Loads the files from the directory specified in the constructors
- **refresh()** Refreshes the content of the PKIStore object.
- **rescheduleRefresh(int)** Changes the interval between refreshes of the store.
- **setAggressive(boolean)** Changes the aggressive mode of the store.
- **stopRefresh()** Stop all refreshes.

### 13.4.5 Fields

- public static final int **TYPE_VOMSDIR**
  - This PKIStore object will contain data from a vomsdir directory.

- public static final int **TYPE_CADIR**
  - This PKIStore object will contain data from a CA directory.

### 13.4.6 Constructors

- **PKIStore**
  ```java
public PKIStore()
```  

- **PKIStore**
  ```java
public PKIStore(java.lang.String dir, int type) throws java.io.IOException, java.security.cert.CertificateException, java.security.cert.CRLException
```  
  - **Description**
    - This is equivalent to PKIStore(dir, type, true)
  - **See also**
    - PKIStore(java.lang.String,int,boolean) (in 13.4.6, page 66)

- **PKIStore**
  ```java
public PKIStore(java.lang.String dir, int type, boolean aggressive) throws java.io.IOException, java.security.cert.CertificateException, java.security.cert.CRLException
```  
  - **Parameters**
    - **dir** -- The directory from which to read the files. If null or the empty string, this will default to "/etc/grid-security/certificates" if type is TYPE_CADIR, or "etc/grid-security/vomsdir" if type is TYPE_VOMSDIR.
    - **type** -- either TYPE_CADIR for CA certificates, or TYPE_VOMSDIR for VOMS certificate.
    - **aggressive** -- if true, loading of data will continue even if a particular file could not be loaded, while if false loading will stop as soon as an error occur.
  - **Throws**
* java.io.IOException – if type is neither TYPE_CADIR nor TYPE_VOMSDIR.
* java.security.cert.CertificateException – if there are parsing errors while loading a certificate.
* java.security.cert.CRLException – if there are parsing errors while loading a CRL.

13.4.7 METHODS

- **getAACandidate**
  
  ```java
  public java.security.cert.X509Certificate[] getAACandidate( javax.security.auth.x500.X500Principal issuer, java.lang.String voName )
  ```

  - **Description**
    
    Gets an array of candidate issuer certificates for an AC with the given issuer and belonging to the given VO.
  
  - **Parameters**
    
    - * issuer – The issuer of the AC.
    - * voName – The name of the VO.
  
  - **Returns** – the array of candidates, or null if none is found.

- **getCAs**
  
  ```java
  public java.util.Hashtable getCAs()
  ```

  - **Returns** – hashtable containing CA certificates. The key is the PKIUtils.getHash() of the subject of the CA. The value is a Vector containing all the CA certificates with the given hash.

  - **See also**
    
    - PKIUtils.getHash(java.security.cert.X509Certificate) (in 13.5.5, page 71)
    - PKIUtils.getHash(javax.security.auth.x500.X500Principal) (in 13.5.5, page 71)
    - PKIUtils.getHash(org.bouncycastle.jce.X509Principal) (in 13.5.5, page 72)
    - java.util.Vector

- **getCRLs**
  
  ```java
  public java.util.Hashtable getCRLs()
  ```

  - **Returns** – hashtable containing CRL. The key is the PKIUtils.getHash() of the issuer of the CRL. The value is a Vector containing all the CRL with the given hash.

  - **See also**
    
    - PKIUtils.getHash(java.security.cert.X509Certificate) (in 13.5.5, page 71)
    - PKIUtils.getHash(javax.security.auth.x500.X500Principal) (in 13.5.5, page 71)
    - PKIUtils.getHash(org.bouncycastle.jce.X509Principal) (in 13.5.5, page 72)
    - java.util.Vector

- **getLSC**
  
  ```java
  public LSCFile getLSC( java.lang.String voName, java.lang.String hostName )
  ```

  - **Description**
    
    Gets the LSC file corresponding to the given VO, for the given server.
– Parameters
  * voName – The name of the VO.
  * hostName – The hostname of the issuing server.

– Returns – The corresponding LSCFile object, or null if none is present.

• getSignings
  
  public java.util.Hashtable getSignings()

  – Returns – hashtable containing SigningPolicy objects. The key is the PKIUtils.getHash() of
  the issuer of the SigningPolicy. The value is a Vector containing all the CRL with the given
  hash.

  – See also
    * SigningPolicy (in 13.7, page 77)
    * PKIUtils.getHash(java.security.cert.X509Certificate) (in 13.5.5, page 71)
    * PKIUtils.getHash(java.security.auth.x500.X500Principal) (in 13.5.5, page 71)
    * PKIUtils.getHash(org.bouncycastle.jce.X509Principal) (in 13.5.5, page 72)
    * java.util.Vector

• load
  
  public void load() throws java.io.IOException, java.security.cert.CertificateException,
java.security.cert.CRLException

  – Description
    Loads the files from the directory specified in the constructors

  – Throws
    * java.io.IOException – if type is neither TYPE_CADIR nor TYPE_VOMSDIR,
    * java.security.cert.CertificateException – if there are parsing errors while loading
      a certificate.
    * java.security.cert.CRLException – if there are parsing errors while loading a
      CRL.

• refresh
  
  public synchronized void refresh()

  – Description
    Refreshes the content of the PKIStore object.

• rescheduleRefresh
  
  public void rescheduleRefresh( int millisec )

  – Description
    Changes the interval between refreshes of the store.

  – Parameters
    * millisec – New interval (in milliseconds)

• setAggressive
  
  public void setAggressive( boolean b )

  – Description
    Changes the aggressive mode of the store.
-- Parameters

- `b` – if true (default) load as much as possible, otherwise stop loading at the first error.

- `stopRefresh`
  ```java
  public void stopRefresh()
  ```

  -- Description

  Stop all refreshes. NOTE: This method must ALWAYS be called prior to disposing of a PKIStore object. The penalty for not doing it is a memory leak.

13.5 Class PKIUtils

13.5.1 Declaration

```java
public class PKIUtils
extends java.lang.Object
```

13.5.2 Constructor Summary

```java
PKIUtils()
```

13.5.3 Method Summary

- `checkIssued(X509Certificate, X509Certificate)` Checks if a certificate issued another certificate, according to RFC 3280.
- `DNCompare(String, String)` Compares two DNs for equality, taking into account different representations for the Email and UserID tags.
- `getAKID(X509Certificate)` Gets the AuthorityKeyIdentifier extension form the passed certificate.
- `getBaseName(File)` Gets the basename of a file.
- `getBasicConstraints(X509Certificate)` Gets the BasicConstraints extension form the passed certificate.
- `getHash(byte[])` Gets the MD5 hash value of the given byte array.
- `getHash(X500Principal)` Gets the MD5 hash value of the given principal.
- `getHash(X509Certificate)` Gets the MD5 hash value of the subject of the given certificate.
- `getHash(X509CRL)` Gets the MD5 hash value of the issuer of the given CRL.
- `getHash(X509Principal)` Gets the MD5 hash value of the given principal.
- `getOpenSSLFormatPrincipal(Principal)` Gets an OpenSSL-style representation of a principal.
- `getSKID(X509Certificate)` Gets the SubjectKeyIdentifier extension form the passed certificate.
- `isCA(X509Certificate)` Checks if the passed certificate is a CA certificate.
- `isProxy(X509Certificate)` Checks if the passed certificate is a proxy certificate.
- `loadCertificates(File)` Loads a set of credentials from a file.
- `loadCertificates(String)` Loads a set of credentials from a file.
- `loadCRL(File)` Loads a CRL from a file.
- `loadCRL(String)` Loads a CRL from a file.
- `readObject(File)` Reads either a certificate or a CRL from a file.
- `selfIssued(X509Certificate)` Checks if the give certificate is self-issued.
skipToCertBeginning(BufferedInputStream) Prepares a BufferedInputStream to read either a certificate or a CRL from it.

13.5.4 Constructors

- PKIUtils
  
  ```java
  public PKIUtils()
  ```

13.5.5 Methods

- checkIssued
  
  ```java
  public static boolean checkIssued(java.security.cert.X509Certificate issuer, java.security.cert.X509Certificate issued)
  ```

  - Description
    Checks if a certificate issued another certificate, according to RFC 3280.
  
  - Parameters
    * issuer – The candidate issuer certificate.
    * issued – The candidate issued certificate.
  
  - Returns
    - true if issuer issued issued, false otherwise.

- DNCompare
  
  ```java
  public static boolean DNCompare(java.lang.String dn1, java.lang.String dn2)
  ```

  - Description
    Compares two DNs for equality, taking into account different representations for the Email and UserID tags.
  
  - Parameters
    * dn1 – the first dn to compare.
    * dn2 – the second dn to compare
  
  - Returns
    - true if dn1 and dn2 are equal, false otherwise.

- getAKID
  
  ```java
  public static org.bouncycastle.asn1.x509.AuthorityKeyIdentifier getAKID(java.security.cert.X509Certificate cert)
  ```

  - Description
    Gets the AuthorityKeyIdentifier extension form the passed certificate.
  
  - Parameters
    * cert – The certificate from which to get the extension.
  
  - Returns
    - the extension if present, or null if not present.

- getBaseName
  
  ```java
  public static java.lang.String getBaseName(java.io.File f)
  ```

  - Description
    Gets the basename of a file.
  
  - Parameters
f – File object representing a file.

- Returns – a string representing the file name, minus the path.

- getBasicConstraints
  public static org.bouncycastle.asn1.x509.BasicConstraints getBasicConstraints(java.security.cert.X509Certificate cert)

  - Description
    Gets the BasicConstraints extension from the passed certificate.
  
  - Parameters
    * cert – The certificate from which to get the extension.
  
  - Returns – the extension if present, or null if not present.

- getHash
  public static java.lang.String getHash(byte[] name)

  - Description
    Gets the MD5 hash value of the given byte array.
  
  - Parameters
    * name – the data from which to compute the hash.
  
  - Returns – the hash value.
  
  - Throws
    * java.lang.IllegalArgumentException – if crl is null.
    * InvalidStateException – if the MD5 algorithm is not supported.

- getHash
  public static java.lang.String getHash(javax.security.auth.x500.X500Principal principal)

  - Description
    Gets the MD5 hash value of the given principal.
  
  - Parameters
    * principal – the principal.
  
  - Returns – the hash value.
  
  - Throws
    * java.lang.IllegalArgumentException – if crl is null.
    * InvalidStateException – if the MD5 algorithm is not supported.

- getHash
  public static java.lang.String getHash(java.security.cert.X509Certificate x509)

  - Description
    Gets the MD5 hash value of the subject of the given certificate.
  
  - Parameters
    * x509 – The certificate from which to get the subject.
  
  - Returns – the hash value.
  
  - Throws
• `getHash`

  ```java
  public static java.lang.String getHash( java.security.cert.X509CRL crl )
  ```

  - **Description**
    Gets the MD5 hash value of the issuer of the given CRL.
  - **Parameters**
    * `crl` – The CRL from which to get the issuer.
  - **Returns** – the hash value.
  - **Throws**
    * `java.lang.IllegalArgumentException` – if `crl` is null.
    * `InvalidStateException` – if the MD5 algorithm is not supported.

• `getHash`

  ```java
  public static java.lang.String getHash( org.bouncycastle.jce.X509Principal principal )
  ```

  - **Description**
    Gets the MD5 hash value of the given principal.
  - **Parameters**
    * `principal` – the principal.
  - **Returns** – the hash value.
  - **Throws**
    * `java.lang.IllegalArgumentException` – if `crl` is null.
    * `InvalidStateException` – if the MD5 algorithm is not supported.

• `getOpenSSLFormatPrincipal`

  ```java
  public static java.lang.String getOpenSSLFormatPrincipal( java.security.Principal principal )
  ```

  - **Description**
    Gets an OpenSSL-style representation of a principal.
  - **Parameters**
    * `principal` – the principal
  - **Returns** – a String representing the principal.

• `getSKID`

  ```java
  public static org.bouncycastle.asn1.x509.SubjectKeyIdentifier getSKID( java.security.cert.X509Certificate cert )
  ```

  - **Description**
    Gets the SubjectKeyIdentifier extension form the passed certificate.
  - **Parameters**
    * `cert` – The certificate from which to get the extension.
  - **Returns** – the extension if present, or null if not present.

• `isCA`

  ```java
  public static boolean isCA( java.security.cert.X509Certificate cert )
  ```
- **Description**
  Checks if the passed certificate is a CA certificate.

- **Parameters**
  * `cert` – the candidate CA certificate.

- **Returns** – true if `cert` is a CA certificate.

- **isProxy**
  ```java
  public static boolean isProxy(java.security.cert.X509Certificate cert)
  ```

  - **Description**
    Checks if the passed certificate is a proxy certificate. Recognizes GT2, GT3 and GT4 proxies.

  - **Parameters**
    * `cert` – the candidate proxy certificate.

  - **Returns** – true if `cert` is a proxy certificate.

- **loadCertificates**
  ```java
  ```

  - **Description**
    Loads a set of credentials from a file.

  - **Parameters**
    * `file` – the File object from which to load the certificates.

  - **Returns** – an array containing the certificates that were present in the file.

  - **Throws**
    * `java.security.cert.CertificateException` – if there were problems parsing the certificates.
    * `java.lang.IllegalArgumentException` – if the file cannot be found.

  - **See also**
    * `java.io.File`

- **loadCertificates**
  ```java
  public static java.security.cert.X509Certificate[] loadCertificates(java.lang.String filename) throws java.security.cert.CertificateException
  ```

  - **Description**
    Loads a set of credentials from a file.

  - **Parameters**
    * `filename` – the name of the file from which to load the certificates.

  - **Returns** – an array containing the certificates that were present in the file.

  - **Throws**
    * `java.security.cert.CertificateException` – if there were problems parsing the certificates.
    * `java.lang.IllegalArgumentException` – if the file cannot be found.

- **loadCRL**
  ```java
  public static java.security.cert.X509CRL loadCRL(java.io.File file) throws java.security.cert.CRLException
  ```
– **Description**
  Loads a CRL from a file.

– **Parameters**
  * file – the File object from which to load the CRL.

– **Returns** – an array containing the certificates that were present in the file.

– **Throws**
  * java.security.cert.CRLException – if there were problems parsing the CRL.
  * java.lang.IllegalArgumentException – if the file cannot be found.

**loadCRL**

```java
public static java.security.cert.X509CRL loadCRL( java.lang.String filename )
throws java.security.cert.CRLException
```

– **Description**
  Loads a CRL from a file.

– **Parameters**
  * filename – the name of the file from which to load the CRL.

– **Returns** – an array containing the certificates that were present in the file.

– **Throws**
  * java.security.cert.CRLException – if there were problems parsing the CRL.
  * java.lang.IllegalArgumentException – if the file cannot be found.

**readObject**

```java
public static java.lang.Object readObject( java.io.File f ) throws java.io.IOException,
java.security.cert.CertificateException, java.security.cert.CRLException
```

– **Description**
  Reads either a certificate or a CRL from a file.

– **Parameters**
  * f – the file from which to read;

– **Returns** – the Object loaded.

– **Throws**
  * java.io.IOException – if there have been problems reading the file.
  * java.security.cert.CertificateException – if there have been problems parsing the certificate.
  * java.security.cert.CRLException – if there have been problems parsing the CRL.

**selfIssued**

```java
public static boolean selfIssued( java.security.cert.X509Certificate cert )
```

– **Description**
  Checks if the give certificate is self-issued.

– **Parameters**
  * cert – The certificate to check.

– **Returns** – true if the certificate is self-issued, false otherwise.

**skipToCertBeginning**

```java
public static int skipToCertBeginning( java.io.BufferedInputStream stream ) throws
java.io.IOException
```


- **Description**
  prepares a BufferedInputStream to read either a certificate or a CRL from it. Skips everything in front of "—–BEGIN" in the stream.

- **Parameters**
  
  * stream – The stream to read and skip.

- **Returns**
  - CERT if a certificate is the next object to be read from the stream, CRL if the next object is a CRL, -1 if the next object is of type unknown.

- **Throws**
  - java.io.IOException – Thrown if there is a problem skipping. Note: this a modified version of code originally written by Joni Hakala

### 13.6 Class PKIVerifier

#### 13.6.1 Declaration

public class PKIVerifier
extends java.lang.Object

#### 13.6.2 Field Summary

- AUTHORITY_KEY_IDENTIFIER
- BASIC_CONSTRAINTS_IDENTIFIER
- KEY_USAGE_IDENTIFIER
- PROXYCERTINFO
- PROXYCERTINFO_OLD
- SUBJECT_KEY_IDENTIFIER
- TARGET

#### 13.6.3 Constructor Summary

- PKIVerifier() Initializes the verifier.
- PKIVerifier(VOMSTrustStore) Initializes the verifier.
- PKIVerifier(VOMSTrustStore, PKISStore) Initializes the verifier.

#### 13.6.4 Method Summary

- cleanup() Cleans up resources allocated by the verifier.
- setCAStore(PKISStore) Sets a new CAStore.
- setVOMSStore(VOMSTrustStore) Sets a new VOMSStore.
- verify(AttributeCertificate) Verifies an Attribute Certificate according to RFC 3281.
- verify(X509Certificate[]) Verifies an certificate chain according to RFC 3280.

#### 13.6.5 Fields

- public static final java.lang.String SUBJECT_KEY_IDENTIFIER
- public static final java.lang.String AUTHORITY_KEY_IDENTIFIER
public static final java.lang.String PROXYCERTINFO

public static final java.lang.String PROXYCERTINFO_OLD

public static final java.lang.String BASIC_CONSTRAINTS_IDENTIFIER

public static final java.lang.String KEY_USAGE_IDENTIFIER

public static final java.lang.String TARGET

13.6.6 Constructors

PKIVerifier

class PKIVerifier ( ) throws java.io.IOException, java.security.cert.CertificateException, java.security.cert.CRLException

- Description
  Initializes the verifier. The CA store is initialized at: "/etc/grid-security/certificates.", while the VOMS store is initialized at "/etc/grid-security/vomsdir".

- Throws
  * java.io.IOException – if there have been IO errors.
  * java.security.cert.CertificateException – if there have been problems parsing a certificate
  * java.security.cert.CRLException – if there have been problems parsing a CRL.

PKIVerifier

class PKIVerifier ( ac.VOMSTrustStore vomsStore ) throws java.io.IOException, java.security.cert.CertificateException, java.security.cert.CRLException

- Description
  Initializes the verifier. The CA store is initialized at: "/etc/grid-security/certificates.”

- Parameters
  * vomsStore – the VOMSTrustStore object which represents the vomsdir store.

- Throws
  * java.io.IOException – if there have been IO errors.
  * java.security.cert.CertificateException – if there have been problems parsing a certificate
  * java.security.cert.CRLException – if there have been problems parsing a CRL.

PKIVerifier

class PKIVerifier ( ac.VOMSTrustStore vomsStore, PKIStore caStore )

- Description
  Initializes the verifier.

- Parameters
  * vomsStore – the VOMSTrustStore object which represents the vomsdir store.
  * caStore – the PKIStore object which represents the CA store.
13.6.7 Methods

- **cleanup**
  
  ```java
class Verifier{
  public void cleanup()
  }
  ```

  **Description**
  Cleans up resources allocated by the verifier. This method MUST be called prior to disposal of this object, otherwise memory leaks and runaway threads will occur.

- **setCAStore**

  ```java
class Verifier{
  public void setCAStore(PKIStore store)
  }
  ```

  **Description**
  Sets a new CAStore.
  **Parameters**
  * store – the new CA store.

- **setVOMSStore**

  ```java
class Verifier{
  public void setVOMSStore(ac.VOMSTrustStore store)
  }
  ```

  **Description**
  Sets a new VOMSStore.
  **Parameters**
  * store – the new VOMS store.

- **verify**

  ```java
class Verifier{
  public boolean verify(ac.AttributeCertificate ac)
  }
  ```

  **Description**
  Verifies an Attribute Certificate according to RFC 3281.
  **Parameters**
  * ac – the Attribute Certificate to verify.
  **Returns**
  true if the attribute certificate is verified, false otherwise.

- **verify**

  ```java
class Verifier{
  public boolean verify(java.security.cert.X509Certificate[] certs)
  }
  ```

  **Description**
  Verifies an certificate chain according to RFC 3280.
  **Parameters**
  * certs – the chain to verify.
  **Returns**
  true if the chain is verified, false otherwise.

13.7 Class SigningPolicy

The purpose of this class is to represent a *.signing_policy file.

13.7.1 Declaration

```java
public class SigningPolicy
extends java.lang.Object
```
13.7.2 Constructor summary


13.7.3 Method summary

findIssuer(String) Finds the record in the signing policy which deals with the specified issuer.
findIssuer(String, int) Finds the record in the signing policy which deals with the specified issuer, starting from a specified record.
getAccessIDCA() Gets the AccessIDCA from the current record.
getCondSubjects() Gets the CondSubjects from the current record.
getName() Gets the basename of the file from which this was loaded.
getPosRights() Gets the PosRights from the current record.
setCurrent(int) Sets the indicate record as the current record.

13.7.4 Constructors

- SigningPolicy
  public SigningPolicy(java.io.File f) throws java.io.IOException
  - Description
    Loads a *.signing_policy file.
  - Parameters
    * f – the File from which to load the Signing Policy.
  - Throws
    * java.io.IOException – if there have been problems loading the file.

13.7.5 Methods

- findIssuer
  public int findIssuer(java.lang.String issuer)
  - Description
    Finds the record in the signing policy which deals with the specified issuer.
  - Parameters
    * issuer – an OpenSSL-style representation of the issuer.
  - Returns
    – the record number, or -1 if none is found.

- findIssuer
  public int findIssuer(java.lang.String issuer, int previous)
  - Description
    Finds the record in the signing policy which deals with the specified issuer, starting from a specified record.
  - Parameters
    * issuer – an OpenSSL-style representation of the issuer.
    * previous – the previous match, or -1 if there was no previous match.
- **Returns** – the record number, or -1 if none is found.

**getAccessIDCA**

```java
public java.lang.String getAccessIDCA()
```

- **Description**
  
  Gets the AccessIDCA from the current record.

- **Returns** – the AccessIDCA.

- **Throws**
  
  * java.lang.IllegalArgumentException – if the record number has not been set.

**getCondSubjects**

```java
public java.util.Vector getCondSubjects()
```

- **Description**
  
  Gets the CondSubjects from the current record.

- **Returns** – a Vector of CondSubjects. Each element is a String.

- **Throws**
  
  * java.lang.IllegalArgumentException – if the record number has not been set.

**getName**

```java
public java.lang.String getName()
```

- **Description**
  
  Gets the basename of the file from which this was loaded.

- **Returns** – the basename or null if nothing was loaded.

**getPosRights**

```java
public java.lang.String getPosRights()
```

- **Description**
  
  Gets the PosRights from the current record.

- **Returns** – the PosRight

- **Throws**
  
  * java.lang.IllegalArgumentException – if the record number has not been set.

**setCurrent**

```java
public void setCurrent( int index )
```

- **Description**
  
  Sets the indicate record as the current record.

- **Parameters**
  
  * index – the record number

- **Throws**
  
  * java.lang.IllegalArgumentException – if the record number is too great or 0.

### 13.8 Class VOMSAttribute

Representation of the authorization information (VO, server address and list of Fully Qualified Attribute Names, or FQANs) contained in a VOMS attribute certificate.
13.8.1 Declaration

public class VOMSAttribute
extends java.lang.Object

13.8.2 Constructor summary

**VOMSAttribute(AttributeCertificate)** Parses the contents of an attribute certificate.

**NOTE:** Cryptographic signatures, time stamps etc.

13.8.3 Method summary

- **getAC()** Deprecated! Use the getXXX() methods instead.
- **getCertList()** Gets the certificates that signed the AC, if the ACCerts extension is present.
- **getFullAttributes()** Gets a copy of the Generic Attributes extension.
- **getFullyQualifiedAttributes()**
- **getHolder()** Returns an String representation of the AC holder.
- **getHolderX509()** Returns an OpenSSL-style representation of the AC holder.
- **getHost()** Returns the hostname of the issuing VOMS server.
- **getHostPort()** Returns the address of the issuing VOMS server, on the form `<host>:<port>`
- **getIssuer()** Returns an OpenSSL-style representation of the AC issuer.
- **getIssuerX509()** Returns an OpenSSL-style representation of the AC issuer.
- **getListOfFQAN()**
- **getNotAfter()** Returns the end date of the AC validity.
- **getNotBefore()** Return the start date of the AC validity.
- **getPort()** Returns the port on which the issuing VOMS server is listening
- **getSerial()** Returns the serial number of the AC.
- **getSignature()** Returns the signature of the AC.
- **getTargets()** Gets the targets of this AC.
- **getVO()** Returns the VO name
- **isHolder(X509Certificate)** Checks the given X509 certificate to see if it is the holder of the AC.
- **isIssuer(X509Certificate)** Checks the given X509 certificate to see if it is the issuer of the AC.
- **isValid()** Checks if the Attribute is valid.
- **toString()** Gets a (brief) string representation of this attribute.
- **validAt(Date)** Checks if the AC was valid at the provided timestamp.

13.8.4 Constructors

- **VOMSAttribute**
  
  ```java
  public VOMSAttribute( ac.AttributeCertificate ac )
  ```

  - **Description**
    
    Parses the contents of an attribute certificate.

    **NOTE:** Cryptographic signatures, time stamps etc. will **not** be checked.

  - **Parameters**
    
    * ac – the attribute certificate to parse for VOMS attributes
13.8.5 Methods

- **getAC**
  ```java
  public ac.AttributeCertificate getAC()
  ```
  Deprecated! Direct access to the Attribute Certificate is going to be removed. Use the getXXX methods in this same classe instead.
  
  - **Returns** – The AttributeCertificate containing the VOMS information

- **getCertList**
  ```java
  public ac.ACCerts getCertList()
  ```
  
  - **Description**
    Gets the certificates that signed the AC, if the ACCerts extension is present.
  
  - **Returns** – the ACCerts extension, or null if it is not present.

  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getFullAttributes**
  ```java
  public ac.FullAttributes getFullAttributes()
  ```
  
  - **Description**
    Gets a copy of the Generic Attributes extension.
  
  - **Returns** – the attributes, or null if they are not present.

  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getFullyQualifiedAttributes**
  ```java
  public java.util.List getFullyQualifiedAttributes()
  ```
  
  - **Returns** – List of String of the VOMS fully qualified attributes names (FQANs):
    vo[/group[/group2...]/Role=[role]]/[Capability=capability]
  
  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getHolder**
  ```java
  public java.lang.String getHolder()
  ```
  
  - **Description**
    Returns an String representation of the AC holder.
  
  - **Returns** – the AC holder.

  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getHolderX509**
  ```java
  public java.lang.String getHolderX509()
  ```
  
  - **Description**
    Returns an OpenSSL-style representation of the AC holder.
  
  - **Returns** – the AC holder.

  - **Throws**
* java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getHost**
  public java.lang.String getHost()  
  
  - **Description**
  Returns the hostname of the issuing VOMS server.
  
  - **Returns**
  hostname.
  
  - **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getHostPort**
  public java.lang.String getHostPort()  
  
  - **Description**
  Returns the address of the issuing VOMS server, on the form <host>:<port>
  
  - **Returns**
  String
  
  - **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getIssuer**
  public java.lang.String getIssuer()  
  
  - **Description**
  Returns an OpenSSL-style representation of the AC issuer.
  
  - **Returns**
  the AC issuer.
  
  - **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getIssuerX509**
  public java.lang.String getIssuerX509()  
  
  - **Description**
  Returns an OpenSSL-style representation of the AC issuer.
  
  - **Returns**
  the AC issuer.
  
  - **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getListOfFQAN**
  public java.util.List getListOfFQAN()  
  
  - **Returns**
  List of FQAN of the VOMS fully qualified attributes names (FQANs)
  
  - **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.
  
  - **See also**
  * FQAN (in 13.2, page 63)

- **getNotAfter**
  public java.util.Date getNotAfter() throws java.text.ParseException
- **Description**
  Returns the end date of the AC validity.
- **Returns** – the end Date.
- **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getNotBefore**
  public java.util.Date getNotBefore() throws java.text.ParseException
  
  - **Description**
    Return the start date of the AC validity.
  - **Returns** – the start Date.
  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getPort**
  public int getPort()
  
  - **Description**
    Returns the port on which the issuing VOMS server is listening
  - **Returns** – the port, or -1 if the informations could not be found.
  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getSerial**
  public java.lang.String getSerial()
  
  - **Description**
    Returns the serial number of the AC.
  - **Returns** – the serial number of the AC.
  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.

- **getSignature**
  public byte[] getSignature()
  
  - **Description**
    Returns the signature of the AC.
  - **Returns** – the byte representation of the AC signature.

- **getTargets**
  public ac.ACTargets getTargets()
  
  - **Description**
    Gets the targets of this AC.
  - **Returns** – the ACTargets extension if present, or null otherwise.
  - **Throws**
    * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded.
- **getVO**
  
  ```java
  public java.lang.String getVO()
  ```

  - **Description**
    - Returns the VO name
  - **Returns**
    - the VO name
  - **Throws**
    - `java.lang.IllegalArgumentException` – if no Attribute Certificate has been loaded.

- **isHolder**
  
  ```java
  public boolean isHolder(java.security.cert.X509Certificate cert)
  ```

  - **Description**
    - Checks the given X509 certificate to see if it is the holder of the AC.
  - **Parameters**
    - `cert` – the X509 certificate to check.
  - **Returns**
    - true if the give certificate is the holder of the AC.
  - **Throws**
    - `java.lang.IllegalArgumentException` – if no Attribute Certificate has been loaded.

- **isValid**
  
  ```java
  public boolean isValid()
  ```

  - **Description**
    - Checks if the Attribute is valid. Only checks start and end of validity.
  - **Returns**
    - true if is valid, false otherwise.
  - **Throws**
    - `java.lang.IllegalArgumentException` – if no Attribute Certificate has been loaded.

- **toString**
  
  ```java
  public java.lang.String toString()
  ```

  - **Description**
    - Gets a (brief) string representation of this attribute.
  - **Returns**
    - the Representation.

- **validAt**
  
  ```java
  public boolean validAt(java.util.Date date)
  ```
– **Description**
  Checks if the AC was valid at the provided timestamp.

– **Parameters**
  * date – if null, current time is used

– **Returns**
  – true if the AC was valid at the time in question.

– **Throws**
  * java.lang.IllegalArgumentException – if no Attribute Certificate has been loaded or the dates have been encoded incorrectly.

13.9 **CLASS VOMSVAlidator**

The main (top) class to use for extracting VOMS information from a certificate and/or certificate chain. The VOMS information can simply be parsed or validated. No validation is performed on the certificate chain – that is assumed to already have happened.

The certificate chain is assumed to already be validated. It is also assumed to be sorted in TLS order, that is certificate issued by trust anchor first and client certificate last.

Example of use: this will validate any VOMS attributes in the certificate chain and check if any of the attributes grants the user the "admin" role in the group (VO) "MyVO".

```java
boolean isAdmin = new VOMSVValidator(certChain).validate().getRoles("MyVO").contains("admin");
```

13.9.1 **DECLARATION**

```java
public class VOMSVValidator
extends java.lang.Object
```

13.9.2 **FIELD SUMMARY**

- isParsed
- isValidated
- myFQANTree
- myValidatedChain
- myValidator
- myVomsAttributes
- theTrustStore
- VOMS_EXT_OID
- vomsStore

13.9.3 **CONSTRUCTOR SUMMARY**

- **VOMSVValidator(X509Certificate)** Convenience constructor in the case where you have a single cert and not a chain.
- **VOMSVValidator(X509Certificate[])** Convenience constructor
  ```java
  Same as VOMSVValidator(validatedChain, null)
  ```
- **VOMSVValidator(X509Certificate[], ACValidator)** If validatedChain is null, a call to setValidatedChain() MUST be made before calling parse() or validate().
13.9.4 Method Summary

- cleanup() Cleans up the object.
- getCapabilities(String) Returns a list of all capabilities attributed to a (sub)group, by combining all VOMS attributes in a hierarchical fashion.
- getRoles(String) Returns a list of all roles attributed to a (sub)group, by combining all VOMS attributes in a hierarchical fashion.
- getVOMSAttributes() Returns a list of VOMS attributes, parsed and possibly validated.
- isValidated()
- parse() Parses the assumed-validated certificate chain (which may also include proxy certs) for any occurrences of VOMS extensions containing attribute certificates issued to the end entity in the certificate chain.
- parse(X509Certificate[]) Parses the assumed-validated certificate chain (which may also include proxy certs) for any occurrences of VOMS extensions containing attribute certificates issued to the end entity in the certificate chain.
- setClientChain(X509Certificate[]) Convenience method: enables you to reuse a VOMSValidator instance for another client chain, thus avoiding overhead in instantiating validators and trust stores and other potentially expensive operations.
- setTrustStore(ACTrustStore) Sets the ACTrustStore instance to use with the default ACValidator.
- setTrustStore(VOMSTrustStore) Sets the trustStore to use with the default ACValidator.
- toString() validates() Parses the assumed-validated certificate chain (which may also include proxy certs) for any occurrences of VOMS extensions containing attribute certificates issued to the end entity in the certificate chain.

13.9.5 Fields

- public static final java.lang.String VOMS_EXT_OID
- protected static ac.ACTrustStore theTrustStore
- protected ac.ACValidator myValidator
- protected java.security.cert.X509Certificate myValidatedChain
- protected java.util.Vector myVomsAttributes
- protected boolean isParsed
- protected boolean isValidated
- protected VOMSValidator.FQANTree myFQANTree
- protected static ac.VOMSTrustStore vomsStore

13.9.6 Constructors

- VOMSValidator
  public VOMSValidator{ java.security.cert.X509Certificate validatedCert }
  - Description
    Convenience constructor in the case where you have a single cert and not a chain.
- Parameters
  * validatedCert –

- See also
  * VOMSValidator(java.security.cert.X509Certificate[]) (in 13.9.6, page 87)

- VOMSValidator
  public VOMSValidator( java.security.cert.X509Certificate[] validatedChain )

  **Description**
  Convenience constructor
  Same as VOMSValidator(validatedChain, null)

  **Parameters**
  * validatedChain –

- VOMSValidator
  public VOMSValidator( java.security.cert.X509Certificate[] validatedChain, ac.ACValidator acValidator )

  **Description**
  If validatedChain is null, a call to setValidatedChain() MUST be made before calling parse() or validate().

  **Parameters**
  * validatedChain – The (full), validated certificate chain
  * acValidator – The AC validator implementation to use (null is default with a BasicVOMSTrustStore)

  **See also**
  * ac.ACValidator (in 13.17, page 102)
  * BasicVOMSTrustStore (in 13.1, page 61)

13.9.7 METHODS

- cleanup
  public void cleanup( )

  **Description**
  Cleans up the object. This method MUST be called before disposing of the object, on pains of a memory leak.

- getCapabilities
  public java.util.List getCapabilities( java.lang.String subGroup )

  Deprecated! Capabilities are deprecated.

  **Description**
  Returns a list of all capabilities attributed to a (sub)group, by combining all VOMS attributes in a hierarchical fashion.

  **Note:** One of the methods parse() or validate() must have been called before calling this method. Otherwise, an IllegalStateException is thrown.

  **Parameters**
• **getRoles**
  
  `public java.util.List getRoles(java.lang.String subGroup)`

  **Description**
  
  Returns a list of all roles attributed to a (sub)group, by combining all VOMS attributes in a
  hierarchial fashion.

  **Note:** One of the methods `parse()` or `validate()` must have been called before calling this
  method. Otherwise, an `IllegalStateException` is thrown.

  **Parameters**

  * `subGroup` – the List of roles.

  **See also**

  * `VOMSValidator.FQANTree` (in 13.10, page 90)

• **getVOMSAttributes**

  `public java.util.List getVOMSAttributes()`

  **Description**
  
  Returns a list of VOMS attributes, parsed and possibly validated.

  **Returns** – List of VOMSAttribute

  **See also**

  * `VOMSAttribute` (in 13.8, page 79)
  * `VOMSValidator.parse()` (in 13.9.7, page 88)
  * `VOMSValidator.validate()` (in 13.9.7, page 90)
  * `VOMSValidator.isValidated()` (in 13.9.7, page 88)

• **isValidated**

  `public boolean isValidated()`

  **Returns** – whether the VOMS attributes are validated or not

  **See also**

  * `VOMSValidator.validate()` (in 13.9.7, page 90)

• **parse**

  `public VOMSValidator parse()`

  **Description**
  
  Parses the assumed-validated certificate chain (which may also include proxy certs) for any
  occurances of VOMS extensions containing attribute certificates issued to the end entity in
  the certificate chain.

  **No validation of timestamps and/or signatures are performed by this method.**
  
  This method returns the object itself, to allow for chaining of commands:

  ```java
  new VOMSValidator(certChain).parse().getVOMSAttributes();
  ```
• Returns – the object itself

• See also
  * VOMSValidator.validate() (in 13.9.7, page 90)

• parse
global static java.util.Vector parse( java.security.cert.X509Certificate[] myValidatedChain )

  – Description
  Parses the assumed-validated certificate chain (which may also include proxy certs) for any occurances of VOMS extensions containing attribute certificates issued to the end entity in the certificate chain.
  No validation of timestamps and/or signatures are performed by this method.

  – Returns – the voms attributes

  – See also
    * VOMSValidator.validate() (in 13.9.7, page 90)

• setClientChain
global VOMSValidator setClientChain( java.security.cert.X509Certificate[] validatedChain )

  – Description
  Convenience method: enables you to reuse a VOMSValidator instance for another client chain, thus avoiding overhead in instantiating validators and trust stores and other potentially expensive operations.
  This method returns the object itself, to allow for chaining of commands:
  vomsValidator.setValidatedChain(chain).validate().getVOMSAttributes();

  – Parameters
    * validatedChain – The new validated certificate chain to inspect

  – Returns – the object itself

• setTrustStore
global static void setTrustStore( ac.ACTrustStore trustStore )

  Deprecated! Use setTrustStore(VOMSTrustStore trustStore) instead.

  – Description
  Sets the ACTrustStore instance to use with the default ACValidator. Default is BasicVOMSTrustStore

  – Parameters
    * trustStore –

  – See also
    * VOMSValidator.setTrustStore(ac.VOMSTrustStore) (in 13.9.7, page 89)
    * BasicVOMSTrustStore (in 13.1, page 61)

• setTrustStore
global static void setTrustStore( ac.VOMSTrustStore trustStore )

  – Description
  Sets the trustStore to use with the default ACValidator.
- **Parameters**
  
  * trustStore – the trustStore.

- **See also**
  
  * ac.VOMSTrustStore (in 13.12, page 93)

- **toString**
  
  ```java
  public java.lang.String toString()
  ```

- **validate**
  
  ```java
  public VOMSValidator validate()
  ```

  **Description**

  Parses the assumed-validated certificate chain (which may also include proxy certs) for any occurances of VOMS extensions containing attribute certificates issued to the end entity in the certificate chain. The attribute certificates are validated: any non-valid entries will be ignored.

  This method returns the object itself, to allow for chaining of commands:

  ```java
  new VOMSValidator(certChain).parse().getVOMSAttributes();
  ```

  **Returns** – the object itself

  **See also**

  * VOMSValidator.parse() (in 13.9.7, page 88)

### 13.10 Class VOMSValidator.FQANTree

Class to sort out the hierarchial properties of FQANs. For example, given the FQANs `/VO/Role=admin` and `/VO/SubGroup/Role=user`, this means that the applicable roles for `/VO/SubGroup` is both `admin` as well as `user`.

#### 13.10.1 Declaration

```java
public class VOMSValidator.FQANTree extends java.lang.Object
``` 

#### 13.10.2 Constructor Summary

```java
VOMSValidator.FQANTree()
``` 

#### 13.10.3 Method Summary

- `add(FQAN)`
- `add(List)`
- `getCapabilities(String)`
- `getRoles(String)`
- `traverse(String)`

#### 13.10.4 Constructors

- **VOMSValidator.FQANTree**
  
  ```java
  public VOMSValidator.FQANTree()
  ```
13.10.5 METHODS

- add
  ```java
  public void add(FQAN fqan)
  ```

- add
  ```java
  public void add(java.util.List fqans)
  ```

- getCapabilities
  ```java
  public java.util.List getCapabilities(java.lang.String voGroup)
  ```

- getRoles
  ```java
  public java.util.List getRoles(java.lang.String voGroup)
  ```

- traverse
  ```java
  protected VOMSValidator.RoleCaps traverse(java.lang.String voGroup)
  ```

Package org.glite.security.voms.ac

Interfaces

ACTrustStore

VOMSTrustStore

Classes

ACCerts

This class represents the ACCerts extension which may be present in the AC.

ACGenerator

AttributeCertificateInfo ::= SEQUENCE {
  version AttCertVersion -- version is v2,
  holder Holder,
  issuer AttCertIssuer,
  signature AlgorithmIdentifier,
  serialNumber CertificateSerialNumber,
  attrCertValidityPeriod AttCertValidityPeriod,
  attributes SEQUENCE OF Attribute,
  issuerUniqueID UniqueIdentifier OPTIONAL,
  extensions Extensions OPTIONAL
}

AttCertVersion ::= INTEGER { v2(1) }

ACTarget

The intent of this class is to represent a single target.

ACTargets

The intent of this class is to represent the ACTargets extension which may be present in the AC.

ACValidator

Validator class capable of validating an Attribute Certificate and verify its signature against a trust store of Attribute Authority certificates.
13.11 INTERFACE ACTrustStore

NOTE: This interface is deprecated. Use VOMSTrustStore instead.

13.11.1 DECLARATION

public interface ACTrustStore

13.11.2 ALL KNOWN SUBINTERFACES

BasicVOMSTrustStore (in 13.1, page 61)

13.11.3 ALL CLASSES KNOWN TO IMPLEMENT INTERFACE

BasicVOMSTrustStore (in 13.1, page 61)

13.11.4 METHOD SUMMARY

getAACandidate(X500Principal) Returns an array of issuer candidates, by performing a name comparison of the AC’s issuer and the subject names of the certificates in the trust store.
13.11.5 METHODS

- getAACandidate
  java.security.cert.X509Certificate[] getAACandidate( javax.security.auth.x500.X500Principal issuer )

  - Description
    Returns an array of issuer candidates, by performing a name comparison of the AC’s issuer
    and the subject names of the certificates in the trust store.
    NOTE: No actual verification or validation of signature takes place in this function.

  - Parameters
    * issuer – the principal to find an issuer for. If null, all known AAs will be returned.

  - Returns
    - an array of issuer candidates, or null in case of an error.

13.12 INTERFACE VOMSTrustStore

13.12.1 DECLARATION

public interface VOMSTrustStore

13.12.2 ALL KNOWN SUBINTERFACES

PKIStore (in 13.4, page 65)

13.12.3 ALL CLASSES KNOWN TO IMPLEMENT INTERFACE

PKIStore (in 13.4, page 65)

13.12.4 METHOD SUMMARY

  getAACandidate(X500Principal, String) Returns candidates to the role of signer of an AC
  with he given issuer and of the give VO.
  getLSC(String, String) Returns the LSCFile corresponding to the VO and Host specified.
  stopRefresh() Stops refreshing the store.

13.12.5 METHODS

- getAACandidate
  java.security.cert.X509Certificate[] getAACandidate( javax.security.auth.x500.X500Principal issuer, java.lang.String voName )

  - Description
    Returns candidates to the role of signer of an AC with he given issuer and of the give VO.

  - Parameters
    * issuer – the DN of the signer.
    * voName – the VO to which he signer belongs.

  - Returns
    - an array of issuer candidates, or null if none is found.
• getLSC
   org.glite.security.voms.LSCFile getLSC( java.lang.String voName, java.lang.String hostName )
   
   – Description
      Returns the LSCFile corresponding to the VO and Host specified.
   
   – Parameters
      * voName – the name of the VO.
      * hostName – the name of the issuing host.
   
   – Returns
      the LSCfile, or null if none is found.

• stopRefresh
   void stopRefresh()
   
   – Description
      Stops refreshing the store. This method MUST be called prior to disposing of the store.

13.13 CLASS ACCerts

This class represents the ACCerts extension which may be present in the AC.

13.13.1 DECLARATION

public class ACCerts
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.13.2 CONSTRUCTOR SUMMARY

   ACCerts() Creates an empty ACCerts object.
   ACCerts(ASN1Sequence) Creates an ACCerts starting from a sequence.

13.13.3 METHOD SUMMARY

   addCert(X509CertificateStructure) Manually adds a certificate to the list.
   getCert() Gets the certificates.
   getDERObject() Makes a DERObject representation.
   getInstance(ASN1Sequence) static variant of the constructor.

13.13.4 CONSTRUCTORS

• ACCerts
   public ACCerts()
   
   – Description
      Creates an empty ACCerts object.

• ACCerts
   public ACCerts( org.bouncycastle.asn1.ASN1Sequence seq )
13.13.5 METHODS

- **addCert**
  
  ```java
  public void addCert( org.bouncycastle.asn1.x509.X509CertificateStructure cert )
  ```

  **Description**
  
  Manually adds a certificate to the list.

  **Parameters**
  
  * cert – The certificate to add.

- **getCerts**
  
  ```java
  public java.util.List getCerts( )
  ```

  **Description**
  
  Gets the certificates.

  **Returns** – the list of certificates.

- **getDERObject**
  
  ```java
  public org.bouncycastle.asn1.DERObject getDERObject( )
  ```

  **Description**
  
  Makes a DERObject representation.

  **Returns** – the DERObject

- **getInstance**
  
  ```java
  public static ACCerts getInstance( org.bouncycastle.asn1.ASN1Sequence seq )
  ```

  **Description**
  
  static variant of the constructor.

  **See also**
  
  * ACCerts(org.bouncycastle.asn1.ASN1Sequence) (in 13.13.4, page 94)

13.14 CLASS ACGENERATOR

```
AttributeCertificateInfo ::= SEQUENCE {
  version AttCertVersion -- version is v2,
  holder Holder,
  issuer AttCertIssuer,
```
signature AlgorithmIdentifier,
serialNumber CertificateSerialNumber,
attrCertValidityPeriod AttCertValidityPeriod,
attributes SEQUENCE OF Attribute,
issuerUniqueID UniqueIdentifier OPTIONAL,
extensions Extensions OPTIONAL

AttCertVersion ::= INTEGER { v2(1) }

13.14.1 DECLARATION

public class ACGenerator
extends java.lang.Object

13.14.2 CONSTRUCTOR SUMMARY

ACGenerator()

13.14.3 METHOD SUMMARY

addAttribute(String, String, String)
addAttributes(String, String, List)
generateACInfo()
setExtensions(Vector)
setHolderIssuer(X500Principal)
setHolderSerial(BigInteger)
setIssuer(X500Principal)
setNotAfter(Date)
setNotBefore(Date)
sign(PrivateKey)

13.14.4 CONSTRUCTORS

• ACGenerator
  public ACGenerator()  

13.14.5 METHODS

• addAttribute
  public void addAttribute( java.lang.String oid, java.lang.String policyAuthority, 
                        java.lang.String value )
• addAttributes
  public void addAttributes( java.lang.String oid, java.lang.String policyAuthority, java.util.List values )

  – Parameters
    * oid –
    * policyAuthority –
    * values –

• generateACInfo
  public AttributeCertificateInfo generateACInfo( )

• setExtensions
  public void setExtensions( java.util.Vector vector )

  – Parameters
    * vector –

• setHolderIssuer
  public void setHolderIssuer( javax.security.auth.x500.X500Principal principal )

  – Parameters
    * principal –

• setHolderSerial
  public void setHolderSerial( java.math.BigInteger integer )

  – Parameters
    * integer –

• setIssuer
  public void setIssuer( javax.security.auth.x500.X500Principal principal )

  – Parameters
    * principal –

• setNotAfter
  public void setNotAfter( java.util.Date date )

  – Parameters
    * date –

• setNotBefore
  public void setNotBefore( java.util.Date date )

  – Parameters
    * date –

• sign
  public void sign( java.security.PrivateKey key )

13.15 Class ACTarget

The intent of this class is to represent a single target.
13.15.1 Declaration

public class ACTarget
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.15.2 Constructor Summary

ACTarget() empty constructor
ACTarget(ASN1Sequence) Creates an ACTarget from a sequence

13.15.3 Method Summary

getDERObject() Makes a DERObject representation.
getGroup() Gets the group element
getInstance(ASN1Sequence) Static variant of the constructor.
getIssuerSerial() Gets the IssuerSerial
getIssuerSerialString() Gets the IssuerSerial element
getName() Gets the name element.
setGroup(GeneralName) Sets the group.
setGroup(String) Sets the group
setIssuerSerial(IssuerSerial) Sets the IssuerSerial
setIssuerSerial(String) Sets the IssuerSerial
setName(GeneralName) Sets the name
setName(String) Sets the name
toString() Creates a string representation of the target.

13.15.4 Constructors

• ACTarget
  public ACTarget()

  – Description
  empty constructor

• ACTarget
  public ACTarget( org.bouncycastle.asn1.ASN1Sequence seq )

  – Description
  Creates an ACTarget from a sequence

  – Parameters
  * seq – the Sequence

  – Throws
  * java.lang.IllegalArgumentException – if there are parsing problems.

13.15.5 Methods

• getDERObject
  public org.bouncycastle.asn1.DERObject getDERObject()
- **Description**  
  Makes a DERObject representation.

- **Returns**  
  the DERObject

- **getGroup**  
  public java.lang.String getGroup()  
  
  - **Description**  
    Gets the group element
  
  - **Returns**  
    the group.

- **getInstance**  
  public static ACTarget getInstance( org.bouncycastle.asn1.ASN1Sequence seq )  
  
  - **Description**  
    Static variant of the constructor.
  
  - **See also**  
    ∗ ACTarget(org.bouncycastle.asn1.ASN1Sequence) (in 13.15.4, page 98)

- **getIssuerSerial**  
  public org.bouncycastle.asn1.x509.IssuerSerial getIssuerSerial()  
  
  - **Description**  
    Gets the IssuerSerial
  
  - **Returns**  
    the IssuerSerial

- **getIssuerSerialString**  
  public java.lang.String getIssuerSerialString()  
  
  - **Description**  
    Gets the IssuerSerial element
  
  - **Returns**  
    the IssuerSerial as String.

- **getName**  
  public java.lang.String getName()  
  
  - **Description**  
    Gets the name element.
  
  - **Returns**  
    the name.

- **setGroup**  
  public void setGroup( org.bouncycastle.asn1.x509.GeneralName g )  
  
  - **Description**  
    Sets the group.
  
  - **Parameters**  
    ∗ g – the group

- **setGroup**  
  public void setGroup( java.lang.String s )
– **Description**  
Sets the group

– **Parameters**
  * s – the group name.

• **setIssuerSerial**
  
  public void setIssuerSerial( org.bouncycastle.asn1.x509.IssuerSerial is )

  – **Description**  
  Sets the IssuerSerial

  – **Parameters**
  * is – the IssuerSerial

• **setIssuerSerial**
  
  public void setIssuerSerial( java.lang.String s )

  – **Description**  
  Sets the IssuerSerial

  – **Parameters**
  * s – a textual representation of the IssuerSerial, in the from subject:serial

• **setName**
  
  public void setName( org.bouncycastle.asn1.x509.GeneralName n )

  – **Description**  
  Sets the name

  – **Parameters**
  * n – the name

• **setName**
  
  public void setName( java.lang.String s )

  – **Description**  
  Sets the name

  – **Parameters**
  * s – the name.

• **toString**
  
  public java.lang.String toString( )

  – **Description**  
  Creates a string representation of the target.

  – **Returns** – the string, or null if there were problems.

### 13.16 Class ACTargets

The intent of this class is to represent the ACTargets extension which may be present in the AC.
13.16.1 Declaration

public class ACTargets
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.16.2 Constructor Summary

ACTargets() Empty constructor.
ACTargets(ASN1Sequence) Creates an ACTargets from a sequence.

13.16.3 Method Summary

AddTarget(ACTarget) Manually add a target.
addTarget(String) Manually add a target.
getDERObject() Makes a DERObject representation.
getInstance(ASN1Sequence) Static variant of the constructor.
getTargets() Gets the list of targets.

13.16.4 Constructors

- ACTargets
  public ACTargets()
  - Description
    Empty constructor.

- ACTargets
  public ACTargets( org.bouncycastle.asn1.ASN1Sequence seq )
  - Description
    Creates an ACTargets from a sequence.
  - Parameters
    * seq – the sequence.
  - Throws
    * java.lang.IllegalArgumentException – if there are parsing errors.

13.16.5 Methods

- AddTarget
  public void AddTarget( ACTarget act )
  - Description
    Manually add a target.
  - Parameters
    * act – the target.
  - See also
    * ACTarget (in 13.15, page 97)
• **addTarget**
  
  public void addTarget(java.lang.String s)
  
  - **Description**
    Manually add a target.
  
  - **Parameters**
    
    - *s* – the target.

• **getDERObject**
  
  public org.bouncycastle.asn1.DERObject getDERObject()
  
  - **Description**
    Makes a DERObject representation.
  
  - **Returns** – the DERObject

• **getInstance**
  
  public static ACTargets getInstance(ASN1Sequence seq)
  
  - **Description**
    Static variant of the constructor.
  
  - **See also**
    
    - ACTargets(ASN1Sequence) (in 13.16.4, page 101)

• **getTargets**
  
  public java.util.List getTargets()
  
  - **Description**
    Gets the list of targets.
  
  - **Returns** – a List containing the targets, expressed as String.

### 13.17 **CLASS ACVALIDATOR**

Validator class capable of validating an Attribute Certificate and verify its signature against a trust store of Attribute Authority certificates.

#### 13.17.1 **DECLARATION**

public class ACValidator
extends java.lang.Object

#### 13.17.2 **FIELD SUMMARY**

- log
- myTrustStore
- myVOMSSStore
- theVerifier
13.17.3 Constructor summary

ACValidator(ACTrustStore)
ACValidator(VOMSTrustStore)

13.17.4 Method summary

cleanup()
getInstance()
getInstance(ACTrustStore)
getInstance(VOMSTrustStore)
validate(AttributeCertificate)

13.17.5 Fields

- protected static org.apache.log4j.Logger log
- protected ACTrustStore myTrustStore
- protected VOMSTrustStore myVOMSStore
- protected org.glite.security.voms.PKIVerifier theVerifier

13.17.6 Constructors

- ACValidator
  public ACValidator(ACTrustStore trustStore)

- ACValidator
  public ACValidator(VOMSTrustStore theStore)

13.17.7 Methods

- cleanup
  public void cleanup()

- getInstance
  public static ACValidator getInstance()

- getInstance
  public static ACValidator getInstance(ACTrustStore trustStore) throws java.lang.IllegalArgumentException

- getInstance
  public static ACValidator getInstance(VOMSTrustStore trustStore) throws java.lang.IllegalArgumentException

- validate
  public boolean validate(AttributeCertificate ac)
13.18 Class AttCertIssuer

Shadow implementation of AttributeCertificateInfo from BouncyCastle

13.18.1 Declaration

public class AttCertIssuer
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.18.2 Constructor Summary

AttCertIssuer(DEREncodable)
AttCertIssuer(GeneralNames)
AttCertIssuer(V2Form)

13.18.3 Method Summary

getDERObject() Produce an object suitable for an ASN1OutputStream.
getIssuerName()

13.18.4 Constructors

• AttCertIssuer
  public AttCertIssuer( org.bouncycastle.asn1.DEREncodable obj )

• AttCertIssuer
  public AttCertIssuer( org.bouncycastle.asn1.x509.GeneralNames v1FormIn )

• AttCertIssuer
  public AttCertIssuer( V2Form v2FormIn )

13.18.5 Methods

• getDERObject
  public org.bouncycastle.asn1.DERObject getDERObject() 

  – Description
  Produce an object suitable for an ASN1OutputStream.

  AttCertIssuer ::= CHOICE {
    v1Form GeneralNames, -- MUST NOT be used in this
    -- profile
    v2Form [0] V2Form -- v2 only
  }

• getIssuerName
  public org.bouncycastle.asn1.x509.GeneralNames getIssuerName() 

13.19 Class AttributeCertificate

A shadow implementation of the non-working BouncyCastle implementation of X.509 Attribute Certificates

13.19.1 Declaration

public class AttributeCertificate
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.19.2 Field summary

logger

13.19.3 Constructor summary

AttributeCertificate(ASN1Sequence)

13.19.4 Method summary

getAcinfo()
getAttributes()
getAttributes(String) Returns a list of the attributes matching the provided OID.
getCertList()
getDERObject() Produce an object suitable for an ASN1OutputStream.
getExtensions()
getFullAttributes()
getFullyQualifiedAttributes()
getHolder()
getHolderX509()
getHost()
getHostPort()
getInstance(InputStream) Create an Attribute Certificate from a input stream containing

    DER-encoded data
getIssuer()
getIssuerX509()
getListOfFQAN()
getNotAfter()
getNotBefore()
getPort()
getSerialNumber()
getSignature()
getSignatureAlgorithm()
getSignatureValue()
getTargets()
getVO()
isValid() Synonym for validAt(null)
validAt(Date) Checks if the AC was valid at the provided timestamp.
verify(PublicKey) Verifies the signature of the AC using the provided signature key
verifyCert(X509Certificate)

13.19.5 FIELDS

- protected static org.apache.log4j.Logger logger

13.19.6 CONSTRUCTORS

- AttributeCertificate
  public AttributeCertificate( org.bouncycastle.asn1.ASN1Sequence seq ) throws java.io.IOException

13.19.7 METHODS

- getAcinfo
  public AttributeCertificateInfo getAcinfo( )

- getAttributes
  public org.bouncycastle.asn1.ASN1Sequence getAttributes( )

  - See also
    * AttributeCertificateInfo.getAttributes() (in 13.20.7, page 110)

- getAttributes
  public java.util.List getAttributes( java.lang.String oid )

  - Description
    Returns a list of the attributes matching the provided OID.
  - Parameters
    * oid – Object Identifier, on the form ”1.2.3.4”
  - Returns – List of ASN.1 objects representing the OID type in question

- getCertList
  public ACCerts getCertList( )

- getDERObject
  public org.bouncycastle.asn1.DERObject getDERObject( )

  - Description
    Produce an object suitable for an ASN1OutputStream.
    AttributeCertificate ::= SEQUENCE {
      acinfo AttributeCertificateInfo,
      signatureAlgorithm AlgorithmIdentifier,
      signatureValue BIT STRING
    }

- getExtensions
  public org.bouncycastle.asn1.x509.X509Extensions getExtensions( )
• **getFullAttributes**
  
  public FullAttributes **getFullAttributes**() 

• **getFullyQualifiedAttributes**
  
  public java.util.List **getFullyQualifiedAttributes**() 

    – **Returns** – List of String of the VOMS fully qualified attributes names (FQANs):
      
      vo[/group[/group2...]]/[Role=[role]][/Capability=capability]

• **getHolder**
  
  public Holder **getHolder**() 

• **getHolderX509**
  
  public java.lang.String **getHolderX509**() 

• **getHost**
  
  public java.lang.String **getHost**() 

• **getHostPort**
  
  public java.lang.String **getHostPort**() 

• **getInstance**
  
  public static AttributeCertificate **getInstance**( java.io.InputStream **in** ) throws java.io.IOException 

    – **Description**
      
      Create an Attribute Certificate from a input stream containing DER-encoded data 

    – **Parameters**
      
      * **in** – 

    – **Returns** – the Attribute Certificate 

    – **Throws**
      
      * java.io.IOException –

• **getIssuer**
  
  public javax.security.auth.x500.X500Principal **getIssuer**() 

• **getIssuerX509**
  
  public org.bouncycastle.jce.X509Principal **getIssuerX509**() 

• **getListOfFQAN**
  
  public java.util.List **getListOfFQAN**() 

    – **Returns** – List of FQAN of the VOMS fully qualified attributes names (FQANs) 

    – **See also**
      
      * org.glite.security.voms.FQAN (in 13.2, page 63) 

• **getNotAfter**
  
  public java.util.Date **getNotAfter**() throws java.text.ParseException 

• **getNotBefore**
  
  public java.util.Date **getNotBefore**() throws java.text.ParseException 

• **getPort**
  
  public int **getPort**()
- **getSerialNumber**
  public org.bouncycastle.asn1.DERInteger getSerialNumber()

- **getSignature**
  public byte[] getSignature()

- **getSignatureAlgorithm**
  public org.bouncycastle.asn1.x509.AlgorithmIdentifier getSignatureAlgorithm()

- **getSignatureValue**
  public org.bouncycastle.asn1.DERBitString getSignatureValue()

- **getTargets**
  public ACTargets getTargets()

- **getVO**
  public java.lang.String getVO()

- **isValid**
  public boolean isValid()
  - Description
    Synonym for validAt(null)
  - Returns – true if currently valid

- **validAt**
  public boolean validAt(java.util.Date date)
  - Description
    Checks if the AC was valid at the provided timestamp.
  - Parameters
    * date – if null, current time is used
  - Returns – true if the AC was valid at the time in question.

- **verify**
  public boolean verify(java.security.PublicKey key)
  - Description
    Verifies the signature of the AC using the provided signature key
  - Parameters
    * key – The (RSA) public key to verify the signature with
  - Returns – true if success, false otherwise

- **verifyCert**
  public boolean verifyCert(java.security.cert.X509Certificate cert)

### 13.20 Class AttributeCertificateInfo

Shadow implementation of AttributeCertificateInfo from BouncyCastle
13.20.1 Declaration

public class AttributeCertificateInfo
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.20.2 Field Summary

AC_CERTS_OID
AC_FULL_ATTRIBUTES_OID
AC_TARGET_OID
VOMS_ATTR_OID
VOMS_EXT_OID

13.20.3 Constructor Summary

AttributeCertificateInfo(ASN1Sequence)

13.20.4 Method Summary

getAttCertVersion()
getAttrCertValidityPeriod()
getAttributes()
getCertList()
getDERObject() Produce an object suitable for an ASN1OutputStream.
getExtensions()
getFullAttributes()
getFullyQualifiedAttributes()
getHolder()
getHost()
getHostPort()
getInstance(ASN1Sequence)
getIssuer()
getIssuerUniqueID()
getListOfFQAN()
getPort()
getSerialNumber()
getSignature()
getTargets()
getVO()

13.20.5 Fields

• public static final java.lang.String AC_TARGET_OID
• public static final java.lang.String AC_CERTS_OID
• public static final java.lang.String AC_FULL_ATTRIBUTES_OID
• public static final java.lang.String VOMS_EXT_OID
• public static final java.lang.String VOMS_ATTR_OID
13.20.6 Constructors

- AttributeCertificateInfo
  public AttributeCertificateInfo( org.bouncycastle.asn1.ASN1Sequence seq )

13.20.7 Methods

- getAttCertVersion
  public org.bouncycastle.asn1.DERInteger getAttCertVersion( )

- getAttrCertValidityPeriod
  public org.bouncycastle.asn1.x509.AttCertValidityPeriod getAttrCertValidityPeriod( )

- getAttributes
  public org.bouncycastle.asn1.ASN1Sequence getAttributes( )

- getCertList
  public ACCerts getCertList( )

- getDERObject
  public org.bouncycastle.asn1.DERObject getDERObject( )

  – Description
  Produce an object suitable for an ASN1OutputStream.

  AttributeCertificateInfo ::= SEQUENCE {
    version AttCertVersion -- version is v2,
    holder Holder,
    issuer AttCertIssuer,
    signature AlgorithmIdentifier,
    serialNumber CertificateSerialNumber,
    attrCertValidityPeriod AttCertValidityPeriod,
    attributes SEQUENCE OF Attribute,
    issuerUniqueID UniqueIdentifier OPTIONAL,
    extensions Extensions OPTIONAL
  }

  AttCertVersion ::= INTEGER { v2(1) }

- getExtensions
  public org.bouncycastle.asn1.x509.X509Extensions getExtensions( )

- getFullAttributes
  public FullAttributes getFullAttributes( )
- **getFullyQualifiedAttributes**
  
  ```java
  public java.util.List getFullyQualifiedAttributes()
  ```
  
  - **Returns** – List of String of the VOMS fully qualified attributes names (FQANs):
    
    ```
    vo[/group[/group2...][/Role=[role][/Capability=capability]
    ```

- **getHolder**
  
  ```java
  public Holder getHolder()
  ```

- **getHost**
  
  ```java
  public java.lang.String getHost()
  ```

- **getHostPort**
  
  ```java
  public java.lang.String getHostPort()
  ```

- **getInstance**
  
  ```java
  public static AttributeCertificateInfo getInstance( org.bouncycastle.asn1.ASN1Sequence seq )
  ```

- **getIssuer**
  
  ```java
  public AttCertIssuer getIssuer()
  ```

- **getIssuerUniqueID**
  
  ```java
  public org.bouncycastle.asn1.DERBitString getIssuerUniqueID()
  ```

- **getListOfFQAN**
  
  ```java
  public java.util.List getListOfFQAN()
  ```
  
  - **Returns** – List of FQAN of the VOMS fully qualified attributes names (FQANs)
  
  - **See also**
    
    ```
    * org.glite.security.voms.FQAN (in 13.2, page 63)
    ```

- **getPort**
  
  ```java
  public int getPort()
  ```

- **getSerialNumber**
  
  ```java
  public org.bouncycastle.asn1.DERInteger getSerialNumber()
  ```

- **getSignature**
  
  ```java
  public org.bouncycastle.asn1.x509.AlgorithmIdentifier getSignature()
  ```

- **getTargets**
  
  ```java
  public ACTargets getTargets()
  ```

- **getVO**
  
  ```java
  public java.lang.String getVO()
  ```

### 13.21 Class AttributeHolder

This class represents an Attribute Holder object.
13.21.1 **Declaration**

```java
public class AttributeHolder
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable
```

13.21.2 **Constructor Summary**

- `AttributeHolder()` Empty constructor.
- `AttributeHolder(ASN1Sequence)` Creates an AttributeHolder object from a Sequence.

13.21.3 **Method Summary**

- `getAttributes()` Gets a list of Generic Attributes.
- `getDERObject()` Makes a DERObject representation.
- `getGrantor()` Gets the Grantor of these attributes.
- `getInstance(ASN1Sequence)` Static variant of the constructor.

13.21.4 **Constructors**

- **AttributeHolder**
  ```java
  public AttributeHolder()
  ```
  Description
  Empty constructor.

- **AttributeHolder**
  ```java
  public AttributeHolder(org.bouncycastle.asn1.ASN1Sequence seq)
  ```
  Description
  Creates an AttributeHolder object from a Sequence.
  
  Parameters
  * `seq` – the Sequence
  
  Throws
  * `java.lang.IllegalArgumentException` – if there are parsing problems.

13.21.5 **Methods**

- **getAttributes**
  ```java
  public java.util.List getAttributes()
  ```
  Description
  Gets a list of Generic Attributes.
  
  Returns
  – the list or null if none was loaded.

- **getDERObject**
  ```java
  public org.bouncycastle.asn1.DERObject getDERObject()
  ```
  Description
  Makes a DERObject representation.
• Returns – the DERObject

- getGrantor
  public java.lang.String getGrantor()
  
  – Description
  Gets the Grantor of these attributes.

  – Returns – the grantor.

- getInstance
  public static AttributeHolder getInstance( org.bouncycastle.asn1.ASN1Sequence seq )

  – Description
  Static variant of the constructor.

  – See also
  * AttributeHolder(org.bouncycastle.asn1.ASN1Sequence) (in 13.21.4, page 112)

13.22 CLASS FullAttributes

This class represents the GenericAttributes extension which may be found in the AC.

13.22.1 DECLARATION

public class FullAttributes
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.22.2 CONSTRUCTOR SUMMARY

  FullAttributes() Empty constructor
  FullAttributes(ASN1Sequence) Creates a FullAttributes object from a sequence.

13.22.3 METHOD SUMMARY

  getAttributeHolders() Returns a list of the AttributeHolders.
  getDERObject() Makes a DERObject representation.
  getInstance(ASN1Sequence) Static variant of the constructor.

13.22.4 CONSTRUCTORS

- FullAttributes
  public FullAttributes( )

  – Description
  Empty constructor

- FullAttributes
  public FullAttributes( org.bouncycastle.asn1.ASN1Sequence seq )
13.22.5 METHODS

- **getAttributeHolders**
  ```java
  public java.util.List getAttributeHolders()
  ```

  - **Description**
    Returns a list of the AttributeHolders.
  - **Returns**
    the list or null if none was there.

- **getDERObject**
  ```java
  public org.bouncycastle.asn1.DERObject getDERObject()
  ```

  - **Description**
    Makes a DERObject representation.
  - **Returns**
    the DERObject

- **getInstance**
  ```java
  public static FullAttributes getInstance( org.bouncycastle.asn1.ASN1Sequence seq)
  ```

  - **Description**
    Static variant of the constructor.
  - **See also**
    * `getInstance(org.bouncycastle.asn1.ASN1Sequence)` in 13.22.4, page 113

### 13.23 CLASS GENERICATTRIBUTE

This class represents the single Generic Attribute.

#### 13.23.1 DECLARATION

```
public class GenericAttribute
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable
```

#### 13.23.2 CONSTRUCTOR SUMMARY

- **GenericAttribute()** Empty constructor
- **GenericAttribute(ASN1Sequence)** Creates a GenericAttributes object from a sequence.
13.23.3 **METHOD SUMMARY**

- `getDERObject()` Makes a DERObject representation.
- `getInstance(ASN1Sequence)` Static variant of the constructor.
- `getName()` Gets the name of the attribute
- `getQualifier()` Gets the qualifier of the attribute
- `getValue()` Gets the value of the attribute

13.23.4 **CONSTRUCTORS**

- **GenericAttribute**
  
  ```java
  public GenericAttribute()
  ```
  
  - **Description**
  
    Empty constructor
  
- **GenericAttribute**
  
  ```java
  public GenericAttribute(ASN1Sequence seq)
  ```
  
  - **Description**
  
    Creates a GenericAttributes object from a sequence.
  
  - **Parameters**
  
    - `seq` – the Sequence
  
  - **Throws**
  
    - `java.lang.IllegalArgumentException` – if there are parsing problems.

13.23.5 **METHODS**

- **getDERObject**
  
  ```java
  public org.bouncycastle.asn1.DERObject getDERObject()
  ```
  
  - **Description**
  
    Makes a DERObject representation.
  
  - **Returns**
  
    – the DERObject

- **getInstance**
  
  ```java
  public static GenericAttribute getInstance(ASN1Sequence seq)
  ```
  
  - **Description**
  
    Static variant of the constructor.
  
  - **See also**
  
    - `GenericAttribute(ASN1Sequence)` (in 13.23.4, page 115)

- **getName**
  
  ```java
  public String getName()
  ```
  
  - **Description**
  
    Gets the name of the attribute
  
  - **Returns**
  
    – the name
• **getQualifier**
  
  public java.lang.String getQualifier()  

  - **Description**
  
  Gets the qualifier of the attribute  

  - **Returns** – the qualifier

• **getValue**
  
  public java.lang.String getValue()  

  - **Description**
  
  Gets the value of the attribute  

  - **Returns** – the value

### 13.24 Class Holder

The Holder object.

```
Holder ::= SEQUENCE {
    baseCertificateID [0] IssuerSerial OPTIONAL,
      -- the issuer and serial number of
      -- the holder’s Public Key Certificate
    entityName [1] GeneralNames OPTIONAL,
      -- the name of the claimant or role
    objectDigestInfo [2] ObjectDigestInfo OPTIONAL
      -- used to directly authenticate the holder,
      -- for example, an executable
}
```

### 13.24.1 Declaration

public class Holder  
extends java.lang.Object  
implements org.bouncycastle.asn1.DEREncodable

### 13.24.2 Constructor Summary

- **Holder(ASN1Sequence)**
- **Holder(X500Principal, BigInteger)**
- **Holder(X509Certificate)**

### 13.24.3 Method Summary

- **getDERObject()**
- **getIssuer()**
- **isHolder(X509Certificate)**
- **matchesDN(X500Principal, GeneralNames)**
13.24.4 Constructors

- **Holder**
  
  ```java
  public Holder( org.bouncycastle.asn1.ASN1Sequence seq )
  ```

- **Holder**
  
  ```java
  public Holder( javax.security.auth.x500.X500Principal issuer, java.math.BigInteger serial )
  ```

- **Holder**
  
  ```java
  public Holder( java.security.cert.X509Certificate cert )
  ```

13.24.5 Methods

- **getDERObject**
  
  ```java
  org.bouncycastle.asn1.DERObject getDERObject( )
  ```

- **getIssuer**
  
  ```java
  public org.bouncycastle.asn1.x509.GeneralNames getIssuer( )
  ```

- **isHolder**
  
  ```java
  public boolean isHolder( java.security.cert.X509Certificate cert )
  ```

- **matchesDN**
  
  ```java
  protected static boolean matchesDN( javax.security.auth.x500.X500Principal subject, org.bouncycastle.asn1.x509.GeneralNames targets )
  ```

13.25 Class IetfAttrSyntax

Implementation of IetfAttrSyntax as specified by RFC3281.

```latex
IetfAttrSyntax ::= SEQUENCE {
  policyAuthority [0] GeneralNames OPTIONAL,
  values SEQUENCE OF CHOICE {
    octets OCTET STRING,
    oid OBJECT IDENTIFIER,
    string UTF8String
  }
}
```

13.25.1 Declaration

```java
public class IetfAttrSyntax
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable
```

13.25.2 Field Summary

- VALUE_OCTETS
- VALUE_OID
- VALUE_UTF8
13.25.3 Constructor summary

IetfAttrSyntax(ASN1Sequence)

13.25.4 Method summary

getDERObject()
getPolicyAuthority()
getValues()
getValueType()

13.25.5 Fields

- public static final int VALUE_OCTETS
- public static final int VALUE_OID
- public static final int VALUE_UTF8

13.25.6 Constructors

- IetfAttrSyntax
  public IetfAttrSyntax( org.bouncycastle.asn1.ASN1Sequence seq )

13.25.7 Methods

- getDERObject
  org.bouncycastle.asn1.DERObject getDERObject( )

- getPolicyAuthority
  public org.bouncycastle.asn1.x509.GeneralNames getPolicyAuthority( )

- getValues
  public java.util.List getValues( )

- getValueType
  public int getValueType( )

13.26 Class ObjectDigestInfo

13.26.1 Declaration

public class ObjectDigestInfo
extends java.lang.Object
implements org.bouncycastle.asn1.DEREncodable

13.26.2 Constructor summary

ObjectDigestInfo(ASN1Sequence)
13.26.3  **METHOD SUMMARY**

- `getDERObject()` Produce an object suitable for an ASN1OutputStream.
- `getDigestAlgorithm()`
- `getDigestedObjectType()`
- `getObjectDigest()`
- `getOtherObjectTypeID()`

13.26.4  **CONSTRUCTORS**

- **ObjectDigestInfo**
  ```java
  public ObjectDigestInfo( org.bouncycastle.asn1.ASN1Sequence seq )
  ```

13.26.5  **METHODS**

- **getDERObject**
  ```java
  public org.bouncycastle.asn1.DERObject getDERObject( )
  ```

  **Description**
  Produce an object suitable for an ASN1OutputStream.

  ```
  ObjectDigestInfo ::= SEQUENCE {
    digestedObjectType ENUMERATED {
      publicKey (0),
      publicKeyCert (1),
      otherObjectTypes (2) },
    -- otherObjectTypes MUST NOT
    -- be used in this profile
    otherObjectTypeID OBJECT IDENTIFIER OPTIONAL,
    digestAlgorithm AlgorithmIdentifier,
    objectDigest BIT STRING
  }
  ```

- **getDigestAlgorithm**
  ```java
  public org.bouncycastle.asn1.x509.AlgorithmIdentifier getDigestAlgorithm( )
  ```

- **getDigestedObjectType**
  ```java
  public org.bouncycastle.asn1.DEREnumerated getDigestedObjectType( )
  ```

- **getObjectDigest**
  ```java
  public org.bouncycastle.asn1.DERBitString getObjectDigest( )
  ```

- **getOtherObjectTypeID**
  ```java
  public org.bouncycastle.asn1.DERObjectIdentifier getOtherObjectTypeID( )
  ```

13.27  **CLASS Util**

13.27.1  **DECLARATION**

```java
public class Util 
extends java.lang.Object
```
13.27.2 Constructor summary

Util()

13.27.3 Method summary

generalNameToGeneralNames(GeneralName)
generalNameToX500Name(GeneralName)
generalNameToX509Name(GeneralName)
x500nameToGeneralName(byte[])
x500nameToGeneralNames(X500Principal)

13.27.4 Constructors

- Util
  public Util()

13.27.5 Methods

- generalNameToGeneralNames
  public static org.bouncycastle.asn1.x509.GeneralNames generalNameToGeneralNames(org.bouncycastle.asn1.x509.GeneralName name)

- generalNameToX500Name
  public static javax.security.auth.x500.X500Principal generalNameToX500Name(org.bouncycastle.asn1.x509.GeneralName name)

- generalNameToX509Name
  public static org.bouncycastle.jce.X509Principal generalNameToX509Name(org.bouncycastle.asn1.x509.GeneralName name)

- x500nameToGeneralName
  public static org.bouncycastle.asn1.x509.GeneralName x500nameToGeneralName(byte[] encodedName)

- x500nameToGeneralNames
  public static org.bouncycastle.asn1.x509.GeneralNames x500nameToGeneralNames(javax.security.auth.x500.X500Principal name)

13.28 Class V2Form

13.28.1 Declaration

class V2Form
  extends java.lang.Object
  implements org.bouncycastle.asn1.DEREncodable

13.28.2 Constructor summary

V2Form(ASN1Sequence)
V2Form(GeneralNames)
13.28.3 Method Summary

getBaseCertificateID()
getDERObject() Produce an object suitable for an ASN1OutputStream.
getIssuerName()  
getObjectDigestInfo()  

13.28.4 Constructors

• V2Form
  public V2Form( org.bouncycastle.asn1.ASN1Sequence seq )

• V2Form
  public V2Form( org.bouncycastle.asn1.x509.GeneralNames issuerName )

13.28.5 Methods

• getBaseCertificateID
  public org.bouncycastle.asn1.x509.IssuerSerial getBaseCertificateID( )

• getDERObject
  public org.bouncycastle.asn1.DERObject getDERObject( )
  – Description
  Produce an object suitable for an ASN1OutputStream.
  V2Form ::= SEQUENCE {
    issuerName GeneralNames OPTIONAL,
    baseCertificateID [0] IssuerSerial OPTIONAL,
    objectDigestInfo [1] ObjectDigestInfo OPTIONAL
  -- issuerName MUST be present in this profile
  -- baseCertificateID and objectDigestInfo MUST NOT
  -- be present in this profile
  }

• getIssuerName
  public org.bouncycastle.asn1.x509.GeneralNames getIssuerName( )

• getObjectDigestInfo
  public ObjectDigestInfo getObjectDigestInfo( )

14 Known Problems and Caveats

There is a set of known problems that may appear during the use of VOMS, but most of them can be eliminated by paying attention to some simple rule.

1. Remember that to obtain a VOMS-enabled proxy you must specify the --voms <vo> option. Without it voms-proxy-init generates a completely standard Globus proxy.

2. Due to the needs of GSI authentication, a maximum of 5 minutes of time shift are allowed among the VOMS server and all its clients. Any greater amount will result in a failed authentication and consequently in an error message.
REFERENCES


