

EGEE

EGEE User's Guide

VOMS CORE SERVICES

Document identifier:	EGEE-JRA1-TEC-571991
Date:	March 29, 2005
Activity:	JRA1: Middleware Engineering and Integration
Lead Partner:	
Document status:	DRAFT
Document link:	https://edms.cern.ch/document/571991/1

Abstract: This user's guide explains how to use Service VOMS Core Services.

Delivery Slip

	Name	Partner	Date	Signature
From	Vincenzo Ciaschini et al.	INFN	Mar 9, 2005	
Reviewed by	Anthony Wilson	RAL	Mar 11, 2005	
Approved by				

Document Change Log

Issue	Date	Comment	Author
Initial version	Mar 9, 2005		V. Ciaschini
Reviewer's comments	Mar 29, 2005	comments reflected	V. Ciaschini

Document Change Record

Issue	Item	Reason for Change

Copyright ©Members of the EGEE Collaboration. 2004. See <http://eu-egEE.org/partners> for details on the copyright holders.

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

1	INTRODUCTION	4
1.1	SERVICE ARCHITECTURE	4
1.2	INTERACTIONS WITH OTHER SERVICES	4
2	QUICKSTART GUIDE	4
2.1	INSTALLING AND RUNNING A VOMS SERVER	4
2.2	USING THE CLIENT COMMAND	5
3	REFERENCE GUIDE	6
3.1	COMMANDLINE INTERFACES	6
3.1.1	VOMS	6
3.1.2	VOMS-PROXY-INIT	7
3.1.3	VOMS-PROXY-INFO	10
3.1.4	VOMS-PROXY-DESTROY	11
3.2	APPLICATION PROGRAM INTERFACES	12
3.2.1	C++ API	12
3.2.2	C API	25
4	AC FORMAT	38
4.1	INTRODUCTION	38
4.2	FQAN	38
4.3	VOMS ATTRIBUTE CERTIFICATE PROFILE	39
4.3.1	HOLDER	41
4.3.2	ATTCERTISSUER	41
4.3.3	V2FORM	41
4.4	ATTRIBUTES	41
4.4.1	EXTENSIONS	42
4.4.2	ATTRIBUTES	42
4.5	ATTRIBUTE CERTIFICATE VALIDATION	42
5	KNOWN PROBLEMS AND CAVEATS	43

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 and document the format of the Attribute Certificate (AC) that gets inserted into a voms-enabled proxy certificate.

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). 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 VOs 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 QUICKSTART GUIDE

2.1 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.
- db** This is the name of the database that will contain the information about the VO. Its default name is "voms", and you need to specify this information if and only if you are installing multiple servers on the same machine. Otherwise the default is perfectly fine.
- port** This is the port number where the VOMS server will be listening. There is no default value for this option, although 15000 is the recommended choice for the first server installed on a host, and additional servers may use 15001, 15002, etc...
- voms-vo** This is the name of the VO to which the VOMS server belongs. Its default value is "unspecified" which is *not* a valid value for a VO name. For this reason, this option should be *always* specified.

- mysql-admin** This is the name of the MySQL root user. 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.
- mysql-pwd** This is the password of the MySQL account specified by the previous option. Its default value is "", meaning that there is no password. This is *not* advisable. The root account of a MySQL server hosting a VOMS DB *must* be protected by a password.
- 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", 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. It does have a default value, but this should never be used. You should always specify a new value.
- code** This is a unique code for each server installed on the same host. It is a value between 0 and 65535, and its default value is 0.

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

```
voms-install-db -port 15000 -vo-name my-vo -mysql-pwd 'some' -voms-pwd 'thing'
```

For a second VO on the same host,

```
voms-install-db -db new-vo -port 15001 -vo-name new-vo -mysql-pwd 'some' -voms-name 'voms2'
-voms-pwd 'thing' -code 1
```

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.

2.2 USING THE CLIENT COMMAND

Be sure in `"/etc/grid-security/vomsdir"` or wherever the `X509_VOMS_DIR` environment variable points you have a copy of the host certificates of all the VOMS servers that could be contacted.

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 easier way to comply to both previous points is to install the VO config RPM that should be distributed by the VOMS servers themselves.

To create a proxy containing attributes retrieved from a VOMS server use

```
voms-proxy-init -voms voname
```

Be aware that invoking `voms-proxy-init` without specifying the `-voms` option create a proxy without attributes (equal to one issued by `grid-proxy-init`).

3 REFERENCE GUIDE

3.1 COMMANDLINE INTERFACES

3.1.1 VOMS

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.

-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".
-logfile	The location of the log file. The default location is "/opt/glite/var/log/<voname>.lg"
-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.
-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 <i>not</i> use it.
-foreground	OBSOLETE. This option will be removed in the future. Do <i>not</i> 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 <hostname>:<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. <i>Never</i> 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	This is a unique numeric code, between 0 and 65535, used to identify different servers installed on the same machine. Its default value is 0.

3.1.2 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.

First of all, in "/etc/grid-security/vomsdir" you should include a copy of the host certificates of all the VOMS servers that could be contacted by the users.

Second, in "/opt/glite/etc/vomses" You should 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 easier way to comply to both previous points is to install the VO config RPM that should be distributed by the VOMS servers themselves.

This is all the configuration that should be done for the use of this command.

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 `:R<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.
- 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 the generated proxy, 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"`.
- 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"`.
- 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 significant 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.
- vomslife** This option lets you specify the validity, in seconds, 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 option."
- 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.
- 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.

- separate** This option save the ACs in a separate file, instead than including them into a proxy certificate.
- 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.
- confile** Specifies a non-standard location of the system-wide vomses files. The default value for this option is “/opt/glite/etc/vomses”.
- userconf** Specifies a non-standard location of the user-specific config files. The default value for this option is “\$HOME/.glite/vomses”.
- 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.

3.1.3 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 subject information.
-issuer	Prints the issuer information.
-type	Prints the proxy's type.
-strength	Prints the length (in bits) of the private key.
-valid	Prints the start and end validity times.
-time	Prints the end validity as a number of seconds for which the object will still be valid.
-info	Lets "-subject", "-issuer", "-valid" and "-time" also apply to ACs, and prints attributes values.
-extra	Prints extra informations that were included in the proxy.
-all	Prints everything. (Implies all other options.)
-fqan	Specifies that attributes should be printed in the FQAN format. (default)
-extended	Specifies that attributes should be printed in the extended format.
-exists	Activates the "-hours" and "-bits" options.
-hours	Verifies that the proxy, and the ACs if "-info" was specified, will be valid for at least <H> hours.
-bits	Verifies that the proxy key has at least bits.

3.1.4 VOMS-PROXY-DESTROY

This command destroys an already existing VOMS proxy.

No configuration is needed.

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 <i>always</i> 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".
-quiet	Prints 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.

3.2 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 argument should be considered a private detail and subject to change without notice.

3.2.1 C++ API

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

3.2.1.1 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:

```
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_/*]`.

3.2.1.1.1 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 not acceptable as part of a group name.

This field **MUST** always be filled.

3.2.1.1.2 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".

3.2.1.1.3 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.

3.2.1.2 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:

```

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:
    AC *ac;
    X509 *holder;
public:
    voms(const voms &);
    voms();
    voms &operator=(const voms &);
    ~voms();
};
  
```

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.

3.2.1.2.4 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

Support for version 0 is going to be phased out of the code base in roughly 6 months (late June - start of July). When that happens, version 0 structures will not be readable anymore. Until then, support for it is being kept as a transition measure.

Please do note that modifying the fields of a version 0 structure associated with a `versiondata` object invalidates the result of the `Export` method on that object.

3.2.1.2.5 `siglen`

The length of the data signature.

3.2.1.2.6 `user`

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

3.2.1.2.7 `userca`

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

3.2.1.2.8 `server`

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

3.2.1.2.9 `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.

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

3.2.1.2.11 `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.

3.2.1.2.12 `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, with a little twist: dates coming from an AC are in GeneralizedTime format, while dates coming from the old version data are in UtcTime format.

Here follows a code example doing that conversion:

```
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 10:
    t->type = V_ASN1_UTCTIME;
    break;
case 15:
    t->type = V_ASN1_GENERALIZEDTIME;
    break;
default:
    ASN1_TIME_free(t);
    return NULL;
}
return t;
}

```

3.2.1.2.13 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 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.

3.2.1.2.14 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.

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

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

3.2.1.2.17 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:
        class Initializer {
        public:
            Initializer();
        private:
            Initializer(Initializer &);

```

```

};

private:
static Initializer init;
std::string ca_cert_dir;
std::string voms_cert_dir;
int duration;
std::string ordering;
std::vector<contactdata> servers;
std::vector<std::string> targets;

public:
error_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:
bool loadfile(std::string, uid_t uid, gid_t gid);
bool loadfile0(std::string, uid_t uid, gid_t gid);
bool verifydata(std::string &message, std::string subject, std::string ca,
                X509 *holder, voms &v);
bool verifyold(std::string &message, std::string subject, std::string ca,

```

```

        X509 *holder, voms &v);
bool verifynew(std::string &message, std::string subject, std::string ca,
               X509 *holder, voms &v);
X509 *check(check_sig f, void *data);
bool check_cert(X509 *cert);
bool retrieve(X509 *cert, STACK_OF(X509) *chain, recurse_type how,
              std::string &bufferold, AC_SEQ **listnew, BIGNUM **bn,
              std::string &subject, std::string &ca, X509 **holder);
verify_type ver_type;
std::string servererrors;
};

```

Let us see the fields in detail.

3.2.1.2.18 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
};

```

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 what errors its execution may generate and on which conditions.

3.2.1.2.19 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.

3.2.1.2.20 Methods

3.2.1.2.21 voms

3.2.1.2.22 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.

3.2.1.2.23 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.

3.2.1.2.24 voms::operator=(const voms &)

This defines an assignment operator between two different voms structures.

3.2.1.2.25 vomsdata

3.2.1.2.26 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' 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 hard about the loss of configurability that it would entail.

3.2.1.2.27 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 as 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 correct permissions are 644 for files and 755 for directories.

VERR_FORMAT The file was not in the expected format.

3.2.1.2.28 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 as 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 644 for files and 755 for directories.

VERR_FORMAT The file was not in the expected format.

3.2.1.2.29 **std::vector<contactdata> vomldata::FindByAlias(std::string alias)**

```
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 `vomldata::LoadSystemContacts()` and `vomldata::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.

3.2.1.2.30 **void vomldata::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

For the moment, this is the only place where the FQAN format for attribute names is not yet fully supported. The attribute field will so have to be specified in the `<group name>:<role name>` format. This situation will be corrected sometime in the 1.2.x series.

SEE ALSO

ResetOrder

3.2.1.2.31 void vomsdata::ResetOrder(void)

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

3.2.1.2.32 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

3.2.1.2.33 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

3.2.1.2.34 void vomsdata::ResetTargets(void)

This function clears the list of targets for an AC. **SEE ALSO** AddTarget, ListTargets

3.2.1.2.35 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

3.2.1.2.36 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      = 0 x f f f f f f f f ,
    VERIFY_NONE     = 0 x 0 0 0 0 0 0 0 0 ,
    VERIFY_DATE     = 0 x 0 0 0 0 0 0 0 1 ,
    VERIFY_TARGET   = 0 x 0 0 0 0 0 0 0 2 ,
    VERIFY_KEY      = 0 x 0 0 0 0 0 0 0 4 ,
    VERIFY_SIGN     = 0 x 0 0 0 0 0 0 0 8 ,
    VERIFY_ORDER    = 0 x 0 0 0 0 0 0 1 0 ,
    VERIFY_ID       = 0 x 0 0 0 0 0 0 2 0
};
```

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_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.

3.2.1.2.37 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.

3.2.1.2.38 bool vomsdata::Retrieve(X509 *cert, STACK_OF(X509) *chain, recurse_type how = RECURSE_CHAIN)

This function retrieves a VOMS AC from a VOMS-enabled proxy certificate, executes the verifications requested by the SetVerificationType() function and interprets the data.

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.

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 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.
VERR_NOINIT	The vomldata object hasn't been properly initialized. Most likely the vom_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.

SEE ALSO

SetVerificationType()

3.2.1.2.39 bool 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 vomldata 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' 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.
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.

3.2.1.2.40 bool Contact(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 <code>ServerError()</code> 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 <code>vomsdata</code> object hasn't been properly initialized. Most likely the <code>voms_dir</code> and <code>ca_dir</code> parameters are empty.

3.2.1.2.41 `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()`

3.2.1.2.42 `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.

3.2.1.2.43 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).
------------	--

3.2.2 C API

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

3.2.2.1 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:

```
struct data {
    char *group;
    char *role;
    char *cap;
};
```

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

3.2.2.1.44 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.

3.2.2.1.45 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".

3.2.2.1.46 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.

3.2.2.2 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:

```
#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;
    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.

3.2.2.2.47 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

Support for version 0 is going to be phased out of the code base in roughly 6 months (late June - start of July). When that happens, version 0 structures will not be readable anymore. Until then, support for it is being kept as a transition measure.

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.

3.2.2.2.48 siglen The length of the data signature.

3.2.2.2.49 user This field contains the subject of the holder's certificate in slash-separated format.

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

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

3.2.2.2.52 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.

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

3.2.2.2.54 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, while *port* is the port at which the AA can be contacted on that server.

3.2.2.2.55 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, with a little twist: dates coming from an AC are in GeneralizedTime format, while dates coming from the old version data are in UtcTime 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 10:
                t->type = V_ASN1_UTCTIME;
                break;
            case 15:
                t->type = V_ASN1_GENERALIZEDTIME;
                break;
            default:
                ASN1_TIME_free(t);
                return NULL;
        }
    }
    return t;
}

```

```

    }
    return NULL;
}

```

3.2.2.2.56 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 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.

3.2.2.2.57 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.

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.

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

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

3.2.2.2.60 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.

```

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.

3.2.2.2.61 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.

3.2.2.2.62 workvo, volen

Compatibility

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

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

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

3.2.2.3 Functions 3.2.2.3.65 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_NOCKET`, `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`

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.

3.2.2.3.66 struct contactdata **VOMS.FindByAlias(struct vomsdata *vd, char *alias, char *system, char *user, int *error)

```
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 */
    char *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! */
};
```

This function looks in the `vomses` files installed in both the system-wide and user-specific directories for servers that have been registered with a particular alias.

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

system The directory where the system-wide files are located. If empty then its default is `.`

user The directory where the user-specific files are stored. If empty its default is `.` If this is also empty, then the default become `.` **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.

3.2.2.3.67 struct contactdata **VOMS_FindByVO(struct vomsdata *vd, char *vo, char *system, char *user, int *error)

This function looks in the vomses files installed in both the system-wide and user-specific directories for servers that have been registered as serving a particular alias.

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

system The directory where the system-wide files are located. If this field is NULL then the default of vomses is used.

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 VO. This may be NULL if there was an error or no server was found registered with the specified VO.

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.

3.2.2.3.68 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.

3.2.2.3.69 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: `"/opt/glite/etc/vomses"`.

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.

3.2.2.3.70 `struct voms *VOMS_Copy(struct voms *, int *error)`

This function duplicates an existing `voms` structure. It is the only 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.

3.2.2.3.71 `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.

3.2.2.3.72 `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 structure with a NULL pointer.

RESULTS

None.

3.2.2.3.73 `int VOMS_AddTarget(struct vomsdaa *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.

3.2.2.3.74 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.

3.2.2.3.75 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.

3.2.2.3.76 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
```

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_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.

RESULTS

0 If there is an error.

<> **0** otherwise.

VERR_NOINIT The vomsdata structure was not properly initialized.

3.2.2.3.77 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.

3.2.2.3.78 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.

3.2.2.3.79 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 the 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 the <group>[:<role>]: format.

Compatibility

This is the only point where the FQAN format is not yet fully supported. Expect this to change in future revisions.

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.

3.2.2.3.80 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.

3.2.2.3.81 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 `FindByAlias()` or `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 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 <code>VOMS_Ordering()</code> 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.

3.2.2.3.82 `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' 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

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 <code>VOMS_Ordering()</code> 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.

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

This function is used to extract from a proxy certificate the VOMS-specific extension, to parse them and to insert the results into the `vomsdata` structure.

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.

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_NOINIT	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.

3.2.2.3.84 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.

3.2.2.3.85 int VOMS_Export(char **buffer, int *buflen, 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.

buflen 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.

3.2.2.3.86 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.

4 AC FORMAT

4.1 INTRODUCTION

X.509 Attribute Certificates (ACs) [1] are used to bind a set of attributes, like group membership, role, security clearance, etc... with an AC holder. Their well-defined, standardized format and easy extensibility make them a premium way to distribute those informations in large system, and in particular in environments where authentication is done via X.509 Certificates [2]. This is the reason why ACs are the format chosen by the VOMS server [5] to encode authorization data.

However, the reference documentation about ACs leaves a huge amount of freedom regarding exactly how ACs should be encoded. The scope of this paper is to document the particular vernacular of ACs used by VOMS, and how the data they contain is supposed to be encoded. This format is in any case fully compatible with what described in [1], and should any incompatibility be found between what is described here and what is described in [1], the latter is the authoritative source.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [3].

4.2 FQAN

The FQAN (short form for Fully Qualified Attribute Name) is what VOMS ACs use in place of the Group/Role attributes. It is better described in [4], although a brief summary will be given in the following paragraphs. It has been developed because of two perceived problems with the standard-defined [1] Group and Role attributes:

1. The Group and Role attributes are completely independent of each other; in particular, Roles are meant to be global, associated directly to the AC holder, regardless of group membership. On the other hand, besides this behaviour VOMS also allows groups and roles to be bound together, using one as a qualifier of the other. While it is indeed possible to encode groups and roles inside the standard attributes in a format that could represent this information, there is no way to have the same format also be readable by other AC users without risking misunderstandings.
2. Also, practical use of group/role attributes in defining ACLs has showed that having them separate is inconvenient, and it is much simpler to have them all expressed together.

For these reasons, a new format has been devised, as documented in [4]. However, here follows a copy of the relevant informations.

Group membership, Role holding and Capabilities may be expressed in a format that bounds them together in the following way:

```
<group name>/Role=[<role name>][/Capability=<capability name>]
```

where the elements between [] are optional.

This format specifies that the AC holder is a member of group <group name>, and in this group he holds the role <role name> while having the capability <capability name>.

<group name>, <role name> and <capability name> are described by the following grammar:

```
group name      ::= entity
                | groupname '/' entity
role name       ::= entity
capability name ::= entity

entity          ::= [a-zA-Z0-9 _]*
```

It can be noted that while role and capability names have a flat structure, group name can be expressed as a series of identifiers separated by the “/” character. This happens because groups are a structured entities, where a group can have subgroups, that can have subgroups, ad libitum. They are represented in the same format as Unix path names, where the first directory name corresponds to the VO name, the second one to a group, the third one to a subgroup of the preceding group, etc. . .

4.3 VOMS ATTRIBUTE CERTIFICATE PROFILE

This is the general format of an AC as defined by [1]. Customizations used by VOMS will be discussed in individual subsections. Everything not specifically mentioned here is intended to be in accordance with [1].

```
AttributeCertificate ::= SEQUENCE {
    acinfo          AttributeCertificateInfo,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue   BIT STRING
}

AttributeCertificateInfo ::= SEQUENCE {
    version          AttCertVersion,
```

```

holder          Holder,
issuer          AttCertIssuer,
signature       AlgorithmIdentifier,
serialNumber    CertificateSerialNumber,
attrCertValidityPeriod AttCertValidityPeriod,
attributes      SEQUENCE OF Attribute,
issuerUniqueID  UniqueIdentifier OPTIONAL,
extensions      Extensions OPTIONAL
}

AttCertVersion ::= INTEGER { v2(1) }

Holder ::= SEQUENCE {
  baseCertificateID      [0] IssuerSerial OPTIONAL,
}

AttCertIssuer ::= CHOICE {
  v2Form      [0] V2Form
}

V2Form ::= SEQUENCE {
  issuerName      GeneralNames OPTIONAL,
  baseCertificateID [0] IssuerSerial OPTIONAL,
  objectDigestInfo [1] ObjectDigestInfo OPTIONAL
}

IssuerSerial ::= SEQUENCE {
  issuer      GeneralNames,
  serial      CertificateSerialNumber,
  issuerUID   UniqueIdentifier OPTIONAL
}

AttCertValidityPeriod ::= SEQUENCE {
  notBeforeTime GeneralizedTime,
  notAfterTime  GeneralizedTime
}

Attribute ::= SEQUENCE {
  type      AttributeType,
  values    SET OF AttributeValue
  -- at least one value is required
}

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY DEFINED BY AttributeType
  
```

4.3.1 HOLDER

The holder of a VOMS AC **MUST** always be an X.509 PKC. As a consequence of this, in VOMS ACs the only admissible choice for the field is the baseCertificateID, hence the absence in the above description, of the other two choices from this SEQUENCE. The issuerUID field in this case **MUST** be present if and only if it is also present in the holder's PKC, and in this case they **MUST** have the same value. Note that [2] says that conforming implementations of PKCs **SHOULD NOT** use this field, but that implementations **SHOULD** be capable to handle it.

4.3.2 ATTCERTISSUER

The AttCertIssuer field **MUST** always be encoded using the V2Form data format.

4.3.3 V2FORM

Conforming ACs **MUST NOT** use either the baseCertificateID or the objectDigestInfo fields. They **MUST** use the issuerName field, which **MUST** contain one and only one distinguished name belonging to the certificate that the AC issuer will use to sign the AC. This in particular means that this subject **MUST NOT** be empty.

4.4 ATTRIBUTES

The attributes field contains informations about the AC holder. At least one attribute **MUST** always be present.

Attributes types use the format defined in [1], repeated here for convenience:

```

IetfAttrSyntax ::= SEQUENCE {
  policyAuthority [0] GeneralNames OPTIONAL,
  values          SEQUENCE OF CHOICE {
    octets      OCTET STRING,
    oid         OBJECT IDENTIFIER,
    string      UTF8String
  }
}
  
```

The attributes Group and Role, defined in [1] are not used by VOMS AC, and **SHOULD NOT** be present in conforming ACs. Instead, it defines a new attribute, FQAN, which holds informations about both, and in fact also binds them together.

```

name          : voms-attribute
OID           : { voms 4 }
syntax       : IetfAttrSyntax
values       : Multiple allowed
  
```

where "voms" is the OID 1.3.6.1.5.3004.100.100 and has been registered for VOMS.

The policyAuthority field of the IetfAttrSyntax **MUST** contain an encoding of both the VO to which the AC issuer belongs and the server which generated this particular attribute, in the following format:

<vo name>://<fqhn>:<port>

all of this component should be omitted, and the IA5STRING choice of the GeneralName type should be used.

On the same way, the octets choice of the values field should be used to encode the FQANs.

4.4.1 EXTENSIONS

In the current version, only a specific subset of the extensions specified in [1] is used and they are described here, along with any specifics points that were originally only loosely defined. A VOMS-compliant AC is allowed to use extensions other than those indicated here, on the condition that they should not be critical.

4.4.1.1 AC Target

This extension MAY be present. If it is present, then then targetName option MUST be used, with the FQDNs of the hosts which the AC is targeted to. Compliant implementation MUST honor this extension. Also, they MUST be capable of understanding at least the targetName option.

4.4.1.2 No Revocation Available

This extension MUST be used in the current version of VOMS ACs.

4.4.2 ATTRIBUTES

While in principle any attribute may be used here, this section will specify what attributes are included in the current version of ACs and which are expected to be recognized by conforming implementations.

4.4.2.1 Fully Qualified Attribute Name (FQAN)

This attribute is used to express user membership in groups and ownership of roles in an integrated way that makes easier to express relations between the two elements. It is fully documented in [FQAN], and MUST be included in any and all VOMS ACs.

4.4.2.2 Group and Role

This two attributes are not used in current version, but they MAY be present. However, in this case their content should be consistent with the content of the FQAN attribute. The suggested way to ensure this is the following:

1. Role and Group have the same number of elements as FQAN.
2. If the n-th element of FQAN denote membership in group G and ownership of role R, then those are the values of the n-th Group and the n-th Role. If no role R is specified in an element of the FQAN attribute, then the corresponding element in the Role attribute is the empty string.

Conforming implementations MAY recognize this two attributes, but if they do they SHOULD the verify correspondence between their values and the content of the FQAN attributes. Should there be a discrepancy, the normative data should be that included in the FQAN element. It is up to the implementation whether to consider a discrepancy enough cause for an error or to settle for a warning.

4.5 ATTRIBUTE CERTIFICATE VALIDATION

All mechanisms described by [1] are kept as they are with only the following change:

It is not required at any time during signature verification that:

- The AC issuer certificate has the signing bit set, or that

- any proxy certificate or user certificate as the signing bit set.

It is although preferred for AC issuer certificate that the signing bit is set.

5 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.
3. In case multiple servers are installed on the same hosts, be certain to specify the `--code` option, giving a different value for each server. Failure to do so will result in the generation of Attribute Certificates that are NOT in accordance with the standard.

REFERENCES

- [1] S. Farrell, R. Housley, RFC 3281: An Internet Attribute Certificate Profile for Authorization.
- [2] R. Housley, W. Polk, W. Ford, D. Solo, RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.
- [3] S. Bradner, RFC 2119: Key words for use in RFCs to Indicate Requirement Levels.
- [4] V. Ciaschini, A. Frohner, Voms Credential Format, <http://edg-wp2.web.cern.ch/edg-wp2/security/voms/edg-voms-credential.pdf>
- [5] R. Alfieri, R. Cecchini, V. Ciaschini, L. Dell'Agnello, A. Frohner, A. Gianoli, L. Karoly, F. Spataro, An Authorization System for Virtual Organizations, Forthcoming in Proceedings of the 1st European Across Grids Conference.