The VOMS C++ API
A Developer's Guide

Vincenzo Ciaschini

December 20, 2005

Contents

Chapter 1

Introduction

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 mantain 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.

Chapter 2

The API.

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

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 {
  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_/]*`.

2.1.1 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.

2.1.2 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".

2.1.3 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.

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:

```
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.

2.2.1 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

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.

Update: With software version 1.6.0 and onwards, support for version 0 has been dropped.

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.

2.2.2 siglen

The length of the data signature.

2.2.3 user

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

2.2.4 userca

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

2.2.5 server

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

2.2.6 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.

2.2.7 voname

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

2.2.8 uri

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

fqdn:*port*

where *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.

2.2.9 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, 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;
}
```

2.2.10 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.

2.2.11 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.

2.2.12 custom

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

2.2.13 fqan

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

2.3 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:
  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:
  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;

  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);

private:
  /* not relevant: removed from this listing. */
};
```

Let us see the fields in detail.

2.3.1 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.

2.3.2 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.

2.4 Methods

2.4.1 voms

2.4.2 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.

2.4.3 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.

2.4.4 voms::operator=(const voms &)

This defines an assignment operator between two different `voms` structures.

2.5 vomsdata

2.5.1 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.

2.5.2 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/edg/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.

2.5.3 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 `/.edg/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.

2.5.4 std::vector<contactdata> vomsdata::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 `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.

2.5.5 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 exemine 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

2.5.6 void vomsdata::ResetOrder(void)

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

2.5.7 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 funciton 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

2.5.8 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

2.5.9 void vomsdata::ResetTargets(void)

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

2.5.10 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

2.5.11 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
};
```

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 ca be dismissed.

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

2.5.12 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.

2.5.13 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 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.

SEE ALSO
SetVerificationType()

2.5.14 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' 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 obtaind 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.

2.5.15 bool vomsdata::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' 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 obtaind 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.

2.5.16 bool vomsdata::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()

2.5.17 bool vomsdata::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 is 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.

2.5.18 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.

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

This function returns a textual description for the error encountered by the other functions. This cannot fail.

2.5.20 bool vomsdata::RetrieveFromCtx(gss_ctx_id_t context, recurse_type how)

This function is capable of retrieving VOMS AC information from a GSS context.
context The context from which to obtain the certificate.
how What to do with the certificate chain. See the documentation of Retrieve (2.5.13) for possible values.

Return and error values are the same as Retrieve. Again, see (2.5.13) for possible values.

2.5.21 bool vomsdata::RetrieveFromCred(gss_cred_id_t credential, recurse_type how)

This function is capable of retrieving VOMS AC information from a GSS credential.

credential The credential from which to obtain the certificate.
how What to do with the certificate chain. See the documentation of Retrieve (2.5.13) for possible values.

Return and error values are the same as Retrieve. Again, see (2.5.13) for possible values.

2.5.22 bool vomsdata::Retrieve(X509_EXTENSION *ext)

Tihs function retrieves the VOMS AC extension from the passed extension. Please note that the unavailability of the holder certificate means that checks related to the holder of the AC will not be done.
ext The extension to evaluate.

Return and error values are the same as Retrieve. Again, see (2.5.13) for possible values.

2.5.23 bool vomsdata::RetrieveFromProxy(recurse_type how)

This function is capable of retrieving VOMS AC information from an existing proxy.
how What to do with the certificate chain. See the documentation of Retrieve (2.5.13) for possible values.

Return and error values are the same as Retrieve. Again, see (2.5.13) for possible values.