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 {
  char *group;
  char *role;
  char *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 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.

2.1.2 role

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

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:

```
#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.

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.

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

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, while *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(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;
}
```

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 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.

2.2.12 custom

This field contains the data returned bu 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. 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.

2.3.1 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.

2.3.2 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.

2.3.3 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.

2.3.4 cdir, vdir

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

2.4 Functions

2.4.1 Generalities

Most of these functions share two parameters, `struct vomsdata *vd`, and `int *error`. To avoid repetition, these two parameters are described here.
error This field contains the error code returned by one of the methods. Please note that the value of this field is only significant if the *last* method called returns an error value. Also, the value of this field is subject to change without notice during method executions, regardless of whether an error effectively occurred.

The possible values returned are: VERR_NONE, VERR_NOSOCKET, VERR_NOIDENT, VERR_COMM, VERR_PARAM, VERR_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.

2.4.2 struct contactdata **VOMS_FindByAlias(struct vomsdata *vd, char *alias, char *system, char *user, int *error)

```
struct contactdata { /*!< You must never allocate directly this structure.
```

```
                        subject to change without notice. The only supported way to
                        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! */
  int   version;   /*!< The version of Globus on which this server runs. */
};
```

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 `/opt/edg/etc/vomses`.

user The directory where the user-specific files are stored. If empty its defaul is `$VOMS_USERCONF`. If this is also empty, then the default becomse `$HOME/.edg/vomses`.

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.

2.4.3 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 `/opt/edg/etc/cvomses` 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 becomse `\$HOME/.edg/vomses`.

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.

2.4.4 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 behaviour.

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.

2.4.5 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 behaviour, 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 `/etc/grid-security/vomsdir`.
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 `/etc/grid-security/certificates`.
RETURNS
A pointer to a properly initialized `vomsdata` structure, or NULL if something went wrong. This is the only case in which an error code would no be associated to the function.

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

2.4.6 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.
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.

2.4.7 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.

2.4.8 void VOMS_Delete(strcut 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.

2.4.9 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.

2.4.10 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.

2.4.11 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.

2.4.12 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 ca 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.

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

This funxtion 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.

2.4.14 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.

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

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

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

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

order The name of an attribute, in 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.

2.4.16 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.

2.4.17 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 existance 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' 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 receied at all. (Usually as a consequence of either a server error or not being recognized by the server as a valid user.)
VERR_ORDER	If the attribute that the client requested, via the `VOMS_Ordering()` function, to be first in the list of attributes received is not first in the attributes returned by the server. This particular code means that the data has been correctly interpreted and is available in the vomsdata structure if you want to use it.
VERR_SERVERCODE	Some strange error occured in the server.

2.4.18 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 receied at all. (Usually as a consequence of either a server error or not being recognized by the server as a valid user.)
VERR_ORDER	If the attribute that the client requested, via the `VOMS_Ordering()` function, to be first in the list of attributes received is not first in the attributes returned by the server. This particular code means that the data has been correctly interpreted and is available in the vomsdata structure if you want to use it.
VERR_SERVERCODE	Some strange error occured in the server.

2.4.19 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 certifiate, 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.

2.4.20 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.

2.4.21 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.

2.4.22 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.

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

This function gives a textual description of the *last* encountered error.
error The error returned by the previous function.
buffer A pointer to a buffer that will hold the error message. If this is NULL, then it will be allocated by the function (and must be released by the caller).
len The length of the buffer pointed to by the previous parameter.
RESULTS

NULL The buffer passed was not long enough, or there is not enough memory to allocate a buffer or the vomsdata structure was improperly initialized.

<>**NULL** A pointer to a buffer containig the error message. If *buffer* was not null, then this is *buffer*, else it is a newly allocated chunk of memory that should be free()ed by the caller.

ERRORS

VERR_NOPARAM	The `vomsdata` structure was not properly initialized.

2.4.24 int VOMS_RetrieveEXT(X509_EXTENSION *ext, struct vomsdata *vd, int *error)

This function retrieves VOMS information from the given extension. Due to the lack of a holder certificate, all checks regarding holder information will be skipped.
ext The extension to parse.
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
Check the description of the the VOMS_Retrieve() function for a description of the errors.

2.4.25 int VOMS_RetrieveFromCtx(gss_ctx_id_t ctx, int how, struct vomsdata *vd, int *error)

This function retrieves VOMS information from the given Globus context.
ctx The context from which to retrieve the certificate to parse.
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
Check the description of the the VOMS_Retrieve() function for a description of the errors.

2.4.26 int VOMS_RetrieveFromCred(gss_cred_id_t cred, int how, struct vomsdata *vd, int *error)

This function retrieves VOMS information from the given Globus credential.
cred The credential from which to retrieve the certificate to parse.
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
Check the description of the the VOMS_Retrieve() function for a description of the errors.

2.4.27 int VOMS_RetrieveFromProxy(int how, struct vomsdata *vd, int *error)

This function retrieves VOMS information from an existing Globus proxy certificate.

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

Check the description of the the VOMS_Retrieve() function for a description of the errors.