



UVOS CLIENT MANUAL

UNICORE Team

Document Version:	1.4.1
Component Version:	1.4.1
Date:	01 05 2011

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

This work was co-funded by the EC Chemomentum project under the FP6 Grant Agreement Nr. IST-033437.

Contents

1	Introduction	1
2	Installation	1
2.1	Installation from the RPM package (RedHat distributions)	1
2.2	Installation from the archive	2
3	Using the command line UVOS client	2
4	Using other available clients	2
5	Developing with uvos-client	3
6	Client configuration	3
7	Commands reference	5

UNICORE VO Service (UVOS) is a client-server system, developed to be used as an additional tool for large distributed systems, providing a solution for grid users management. Grid systems, especially UNICORE grid middleware, are the mainspring of the UVOS system. UVOS can be used with different systems, however is designed primarily to support UNICORE grid middleware.

For more information about UVOS visit <http://uvos.chemomentum.org>.

1 Introduction

This package called *uvos-client* contains client side libraries and a command line application (CLC) for accessing UVOS Server.

For a general introduction of UVOS please refer to its website: <http://uvos.chemomentum.org>

A complete overview of the UVOS is available in the initial chapters of the "UVOS Server manual" available there.

2 Installation

UVOS client is distributed either as a platform independent archive or as an installable, platform dependent package such as RPM. Depending on the installation source used, installation method and paths after installation are different.

2.1 Installation from the RPM package (RedHat distributions)

If installing using distribution-specific package as RPM the following instruction shall be used: Download RPM UVOS client package from UNICORE download site and install it using rpm command as the root user:

```
$> rpm -i unicore-uvos-clc-VERSION.noarch.rpm
```

You can also (what is suggested) use yum to install and subsequently update the component. The yum installation may be performed as follows (note that first command is needed only if you have not yet installed the EMI yum repository):

```
$> wget --no-check-certificate \
https://twiki.cern.ch/twiki/pub/EMI/EMI-1/rc1.repo \
-O /etc/yum.repos.d/emi-1.repo
$> yum install unicore-uvos-clc
```

The default configuration of the client is placed in `/etc/unicore/uvos-clc/uvosClient.conf`. Upon a first start of the client it will be copied to `.uvos-clc/` folder in your home directory. You should edit it there (see [Configuration Section 6](#) for details). The executable program is called `uvos-clc`.

2.2 Installation from the archive

If installing using a portable archive:

1. Download the `uvos-client` archive from the UNICORE download site and unpack it.
2. The default configuration of the client is placed in `conf/uvosClient.conf`. You should edit it there (see [Configuration Section 6](#) for details). The program can be found in the `bin/` subdirectory and is called `uvoscmd.sh`.

3 Using the command line UVOS client

UVOS Command Line Client can operate in batch or interactive mode. Invoking it without arguments provides you with usage instructions. When invoking UVOS CLC user must select one of the available commands and provide arguments for it.

UVOS CLC offers a built in help system: it allows you to list all commands and get help on each of them. The following command outputs help for the operation `addIdentity` (assuming you are invoked UVOS CLC in interactive mode, similarly you can invoke it in batch mode):

```
help addIdentity
```

Note that in interactive mode in order to pass arguments which contain spaces you should surround them with double quotes "like this".

The command line client uses a configuration file to get information about:

- UVOS server address.
- How to authenticate to the server.
- Trusted certificates for TLS connection.

See [Configuration Section 6](#) for details.

4 Using other available clients

Other clients which are available in this package are usually not useful for UVOS users. The sole exception is `SAMLSelfClient`. This program gets all attributes which are defined for the identity making the call (i.e. this one which is set in client's configuration file).

The `WebClient` program serves as an example for developers only so you can ignore it.

5 Developing with uvos-client

If you intend to use this package as a library please refer to the JavaDocs documentation (it is available from the download site, and from the documentation site which is preferred, the most accurate source). Remember that there are two VO APIs implemented by this library:

- **SAML API** It uses SAML 2.0 and so is standard-complaint, can be used to access any version of UVOS server and possibly other SAML services. However it offers only a read-only access. If you can then use use this API. It is included in the package: `pl.edu.icm.unicore.uvos.wsclient.samlapi`
- **UVOS API** It uses an own UVOS web services interface. It is not guaranteed that using this API will work perfectly with older UVOS server releases (however most of fundamental functions should work). Also this API in not standards complaint. On the other hand it allows you to use all UVOS features. It is included in the package: `pl.edu.icm.unicore.uvos.wsclient.api`

Also note that if intend to simply add UVOS support for your XFire or WSRFLite or UAS service then you should use `xfire-voutils` or `uas-authz` packages which provides a ready to use solution for such cases.

For additional information please contact UVOS or UNICORE developers at: unicore-devel@lists.sourceforge.net

6 Client configuration

The client uses the configuration file to get information about:

- UVOS server address,
- clients identity which is used to authenticate the client to the UVOS server,
- trusted certificates for TLS connection.

The file location can be chosen via `-c file` argument or by setting the `UVOSCLC_CONFIG` shell variable. The Java Properties format is used. See:

[http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load\(java.io.InputStream\)](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream))

for a formal discussion, however it is enough to know that:

- comments are started with `#`
- on each line (non empty and not commented out) there is one property and its value defined. . .
- . . . with the syntax: `propertyName=property value`

UVOS can be contacted using insecure HTTP or secure HTTPS. Except closed test environments it is always strongly suggested to use secure HTTPS.

Clients can authenticate to the UVOS using two mechanisms:

- using X.509 certificate - it is taken from SSL session so to use this authentication you must use *https* protocol.
- using email and password - then any protocol can be used.

Below the valid properties are presented along with description in comments (so it can be copied and used as an example configuration).

```
# The following three mandatory properties define UVOS server address. ←
# Possible values 'http' or 'https'
protocol=https
# Port number, integer 1-65535
port=2443
# Host address or IP address
host=localhost

# Those settings are used only when protocol is 'https'.
# Do we authenticate ourself to the server using our TLS certificate? ←
SSLauthenticate=true
# If authenticate is false keystore* properties are not needed. ←
# Otherwise those are
# properties define a keystore to read client's certificate from. ←
# It will be used
# for authenticating it to the UVOS server.
# Keystore path
keystore=conf/certs/dummyClient.jks
# Keystore password
keystorePasswd=the!client
# Keystore type (JKS or PKCS12)
keystoreType=JKS
# Keystore alias with private key
keystoreAlias=mykey
# Keystore key's password. Defaults to keystorePassword so can be ←
# undefined.
#keyPasswd=

# However truststore is always needed for https - this file ←
# contains all
# trusted certificates. You can use the same file for keystore and ←
# truststore
# Keystore path
truststore=conf/certs/dummyClientTruststore.jks
# Keystore password
truststorePasswd=the!client
# Keystore type (JKS or PKCS12)
truststoreType=JKS
```

```
# Do we authenticate with HTTP Basic authN (i.e. username & passwd) ←
?
HTTPauthenticate=true
# The following two options are only relevant if HTTPauthenticate ←
is true
# email address of identity (present in UVOS)
HTTPpasswd=
# ... and its password
HTTPuser=voadmin@localhost
```

7 Commands reference

This section provides reference documentation about all commands which are available in UVOS Command Line Client. This reference can be generated by the command *helpAll*. Note that information generated by help system of the Command Line Client is always up to date.

Command: addEquivalentIdentity

Creates a new identity, which is alias of (is equivalent to) ←
 another, existing identity. Syntax:
 addEquivalentIdentity <dn|x509|email> <newIdentity> <dn|x509| ←
 email> <equivalent>
 The first argument is an identity type. Next is value, which is a ←
 simple string in case of 'dn' or
 'email' types and a file name with X509 certificate in case of ' ←
 x509' type.
 It is the value of added identity. Next come parameters of ←
 equivalent identity.
 Note that label of the new identity will be the same as of it's ←
 equivalent.

Example:

```
addEquivalentIdentity email john@example.com email john@example. ←
org
```

~~~~~  
 Command: addGroup

Adds a new group. Syntax:

```
addGroup <parentGroupPath> <newGroupName>
```

Separate the parent group path elements with '/'. Example:

```
addGroup /parentGroup newGroup
```

~~~~~  
 Command: addIdentity

Creates a new identity. Syntax:
 addIdentity <dn|x509|email> <value> [label]
 or: addIdentity email <value> passwd <password> [label]
 The first argument is an identity type. Next is value, which is a ←
 simple string in case of 'dn' or 'email'
 types and a file name with certificate in case of 'x509' type. 3rd ←
 (optional) argument is a friendly
 (unique) name for the identity. It is common for all equivalent ←
 identities. When adding email type identity
 it is also possible to set it's password.

Example:

```
addIdentity email test@example.com Johnny
```

~~~~~

Command: addNotification

Adds a new notification definition for the given action. Optionally ←  
 you can register it only in the context of a group

Syntax:

```
addNotification <action> <recipients> [groupFilter]
```

Example:

```
addNotification addGroup receiver@example.com /Math-VO
```

~~~~~

Command: addToGroup

Adds an identity to a group. Syntax:

```
addToGroup <dn|x509|email> <identity> <groupPath>
```

The first argument is an identity type. Next is value, which is a ←
 simple string in case of 'dn' or 'email'

types and a file name with X509 certificate in case of 'x509' type. ←
 It is the value of added identity.

The last argument is the group path.

Example:

```
addToGroup email john@example.com /group/subgroup
```

~~~~~

Command: areEquivalent

Checks if two given identities are equivalent.

Syntax:

```
areEquivalent <dn|x509|email> <identity1> <dn|x509|email> < ←  

  identity2>
```

Example:

```
areEquivalent email ann@example.com email ann@example.org
```

```
~~~~~
Command: changeLabel

Changes unique name of identity, which is common for all equivalent ←
identities. Syntax:
 changeLabel <dn|x509|email> <identity> <newLabel>
Example:
 changeLabel email ann@example.com Ann

~~~~~
Command: changePasswd

Changes the given identity password (or deletes it).
Syntax:
  changePasswd email <identity> [newPasswd]
If new password is not given then the password is removed. Note that ←
it is technically possible to set also
password for other types of identity but it makes no sense. Example:
  changePasswd email ann@example.com Secret

~~~~~
Command: copyGroup

Copies or moves a group to a different parent group. Syntax:
 copyGroup <groupToBeCopiedPath> <newParentGroupPath> <doMove> [←
 newName]
Example:
 copyGroup /group/subgroup /parent true movedG
will create group /parent/movedG with the same contents as /group/ ←
subgroup has, then /group/subgroup will be
deleted.

~~~~~
Command: disableAttribute

Disables an attribute. Syntax:
  disableAttribute <global> <dn|x509|email> <identity> < ←
  AttributeName> [value]
  disableAttribute <group> <groupPath> <AttributeName> [value]
  disableAttribute <ig> <dn|x509|email> <identity> <groupPath>< ←
  AttributeName> [value]
The first argument specifies what kind of attribute will be changed ←
: group attribute, identity global
attribute, or identity attribute valid only in particular group. If ←
the optional 'value' argument is set
then only this value of attribute will be disabled.
```

```
Example:
  disableAttribute ig email ann@example.com /group urn:unicore: ←
    attrType:user:profession scientist

~~~~~
Command: enableAttribute

Enables an attribute. Syntax:
 enableAttribute <global> <dn|x509|email> <identity> < ←
 AttributeName> [value]
 enableAttribute <group> <groupPath> <AttributeName> [value]
 enableAttribute <ig> <dn|x509|email> <identity> <groupPath>< ←
 AttributeName> [value]
The first argument specifies what kind of attribute will be changed ←
: group attribute, identity global
attribute, or identity attribute valid only in particular group. If ←
the optional 'value' argument is set
then only this value of attribute will be disabled.
Example:
 enableAttribute ig email ann@example.com /group urn:unicore: ←
 attrType:user:profession scientist

~~~~~
Command: exit

Exits application

~~~~~
Command: exit

Exits application

~~~~~
Command: getATypes

Lists all known attribute types. Syntax:
  getATypes

~~~~~
Command: getATypes

Lists all known attribute types. Syntax:
 getATypes

~~~~~
Command: getAllEquivalentents

Retrieves a list of all identities that are equivalent to the given ←
```

```
one.
Syntax:
  getAllEquivalents <dn|x509|email> <identity1>
Example:
  getAllEquivalents email ann@example.com

~~~~~
Command: getAllGroups

Retrieves a list of all groups the given identity is a member of.
Syntax:
 getAllGroups <dn|x509|email> <identity> [implied]
If the last arg is given then also groups implied are returned, i.e ←
 . if user is a member of group
/A/B then /A will be returned then too. Example:
 getAllGroups email ann@example.com

~~~~~
Command: getAllIdentities

Retrieves a list of all identities
Syntax:
  getAllIdentities

~~~~~
Command: getApplication

Get an ID of an application submitted by a given identity. Prints -1 ←
 when no application is found.
Warning - note that more than one application may be submitted by a ←
 single identity.
Syntax:
 getApplication <dn|x509|email> <identity>

~~~~~
Command: getApplicationForms

Lists available application forms.
Syntax:
  getApplicationForms

~~~~~
Command: getApplications

Lists application actions. Arguments filter the response. Negative ←
 formId means any formId.
Syntax:
```

```

getApplications [formId] [status]

~~~~~
Command: getApplications

Lists application actions. Arguments filter the response. Negative ↵
formId means any formId.
Syntax:
  getApplications [formId] [status]

~~~~~
Command: getAttribute

Returns value(s) of the attribute of the given element.
Syntax:
 getAttribute <global> <dn|x509|email> <identity> <attribute> [↵
 allScopes] [includeImpliedGroups]
 getAttribute <group> <groupPath> <attribute>
 getAttribute <ig> <dn|x509|email> <identity> <groupPath> < ↵
 attribute>
The first argument specifies what kind of attribute will be listed: ↵
group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
 getAttribute ig email ann@example.com /group urn:unicore:attrType ↵
 :user:profession

~~~~~
Command: getAttributes

Returns all attributes of the given element.
Syntax:
  getAttributes <global> <dn|x509|email> <identity> [allScopes] [ ↵
    includeImpliedGroups]
  getAttributes <group> <groupPath>
  getAttributes <ig> <dn|x509|email> <identity> <groupPath>
The first argument specifies what kind of attribute will be listed: ↵
group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  getAttributes ig email ann@example.com /group

~~~~~
Command: getAuthz

Lists authorization policy for VO service access
Syntax:
 getAuthz global

```

```

 getAuthz group <group>
Example:
 getAuthz group /some/group m
~~~~~
Command: getContent

Returns the content (subgroups and identities) of the given group.
Syntax:
    getContent <groupPath>
Example:
    getContent /group

~~~~~
Command: getDisabledAttributes

Lists all disabled attributes. Only disabled values are presented. ←
Syntax:
 getDisabledAttributes <global> <dn|x509|email> <identity>
 getDisabledAttributes <group> <groupPath>
 getDisabledAttributes <ig> <dn|x509|email> <identity> <groupPath ←
 >
The first argument specifies what kind of attribute will be listed: ←
 group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
 getDisabledAttributes ig email ann@example.com /group

~~~~~
Command: getDisabledAttributes

Lists all disabled attributes. Only disabled values are presented. ←
Syntax:
    getDisabledAttributes <global> <dn|x509|email> <identity>
    getDisabledAttributes <group> <groupPath>
    getDisabledAttributes <ig> <dn|x509|email> <identity> <groupPath ←
    >
The first argument specifies what kind of attribute will be listed: ←
    group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
    getDisabledAttributes ig email ann@example.com /group

~~~~~
Command: getEvents

Displays all events (i.e. contents modification) which occurred in ←

```

```
the specified period of time. Syntax:
 getEvents [from <yyyy-mm-dd> <hh:mm:ss>] [to <yyyy-mm-dd> <hh:mm: ←
 ss>]
If any of range bounds is not specified then it is unlimited.
Example:
 getEvents from 2007-03-28 21:49:00
It will display all events from the specified date till now.

~~~~~
Command: getIITypes

Lists all known identity types. Syntax:
  getIITypes

~~~~~
Command: getIITypes

Lists all known identity types. Syntax:
 getIITypes

~~~~~
Command: getMyIds

Lists all identities of the current user
Syntax:
  getMyIds

~~~~~
Command: getNotifications

Lists notifications. Optionally you can query for notifications of ←
 a particular action.
Syntax:
 getNotifications [action] [groupFilter]
Example:
 getNotifications addGroup /Math-VO

~~~~~
Command: getNotifications

Lists notifications. Optionally you can query for notifications of ←
  a particular action.
Syntax:
  getNotifications [action] [groupFilter]
Example:
  getNotifications addGroup /Math-VO

~~~~~
Command: getPerms
```

Lists all permissions of specified user

Syntax:

```
getPerms global <dn|x509|email> <identity>
getPerms group <group> <dn|x509|email> <identity>
```

Example:

```
getPerms group /some/group email ann@example.com
```

~~~~~

Command: help

Provides help. Use without arguments to get generic help or with ↵  
command name as parameter to get help  
on the specified command usage.

~~~~~

Command: helpAll

Provides full help for all commands.

~~~~~

Command: isMember

Checks if the given identity is a member of the given group.

Syntax:

```
isMember <dn|x509|email> <identity> <groupPath>
```

Example:

```
isMember email ann@example.com /group
```

~~~~~

Command: processApplication

Process an application. This method doesn't fulfill any of extra ↵  
requests attached to the application.

Syntax:

```
processApplication <appId> <action> <sendEmail:true|false> [↵
additionalNotes]
```

Valid actions are REJECT, ACCEPT or REMOVE

~~~~~

Command: purgeHistory

Deletes all historical content of database which is older than ↵  
requested. Syntax:

```
purgeHistory <yyyy-mm-dd> <hh:mm:ss>
```

Example:

```
purgeHistory 2007-03-28 21:49:00
```

```

~~~~~
Command: removeAType

Deletes an existing attribute type, if it is unused (must be also ←
unused in history!)
Syntax:
  removeAType <name>
Example:
  removeAType urn:someTypeTo:Remove

~~~~~
Command: removeApplicationForm

Remove an application form.
Syntax:
 removeApplicationForm <id>

~~~~~
Command: removeAttribute

Removes an attribute. Syntax:
  removeAttribute <global> <dn|x509|email> <identity> < ←
  AttributeName>
  removeAttribute <group> <groupPath> <AttributeName>
  removeAttribute <ig> <dn|x509|email> <identity> <groupPath>< ←
  AttributeName>
The first argument specifies what kind of attribute will be removed ←
: group attribute, identity global
attribute, or identity attribute valid only in particular group.
Example:
  removeAttribute ig email ann@example.com /group urn:unicore: ←
  attrType:user:profession

~~~~~
Command: removeAuthz

Removes authorization policy for VO service access
Syntax:
 removeAuthz <group> a <AttributeName>
 removeAuthz <group> <o|m>
Use group '/' to modify global policy. Permissions can removed ←
either from the bearer of an attribute or
from the accessed resource owner ('o' case) or from the group ←
members ('m' case')
Example:
 removeAuthz /some/group m

~~~~~

```

Command: removeFromGroup

Removes an identity from a group. Syntax:

```
removeFromGroup <dn|x509|email> <identity> <groupPath>
```

The first argument is an identity type. Next is value, which is a ←  
simple string in case of 'dn' or 'email'

types and a file name with X509 certificate in case of 'x509' type. ←

It is the value of added identity.

The last argument is the group path.

Example:

```
removeFromGroup email john@example.com /group/subgroup
```

~~~~~

Command: removeGroup

Removes a group. Syntax:

```
removeGroup <groupPath> [true|false]
```

Separate the parent group path elements with '/'. Optional 2nd ←

argument specifies if the behaviour should

be recursive (default is false).

Example:

```
removeGroup /gl/subg/groupToRemove true
```

~~~~~

Command: removeIdentity

Removes a identity. Syntax:

```
removeIdentity <dn|x509|email> <value>
```

The first argument is an identity type. Next is value, which is a ←

simple string in case of 'dn' or 'email'

types and a file name with certificate in case of 'x509' type.

Example:

```
removeIdentity email test@example.com
```

~~~~~

Command: removeNotification

Removes an existing notification.

Syntax:

```
removeNotification <id>
```

Example:

```
removeNotification 3
```

~~~~~

Command: setApplicationForm

Adds or updates application form. Definition is read form XML file

Syntax:

```

    setApplicationForm <update:true|false> <fileWithDefinition <
    >
Example:
    setApplicationForm true appForm3.xml
~~~~~
Command: setAttribute

Adds or changes an attribute. Syntax:
 setAttribute <global> <dn|x509|email> <identity> <true|false> <
 attributeTypeAndNameAttributeName> [value1 value2 ...]
 setAttribute <group> <groupPath> <true|false> <AttributeName> [
 value1 value2 ...]
 setAttribute <ig> <dn|x509|email> <identity> <groupPath> <true|
 false> <AttributeName> [value1 value2 ...]
The first argument specifies what kind of attribute will be changed <
: group attribute, identity global
attribute, or identity attribute valid only in particular group. <
 The boolean argument specifies if existing
attribute with the same name should be updated.
Example:
 setAttribute ig email ann@example.com /group true urn:unicore: <
 attrType:user:profession scientist
~~~~~
Command: setAuthz

Modifies authorization policy for VO service access
Syntax:
    setAuthz <group> a <perm> <AttributeName> [value]]
    setAuthz <group> <o|m> <perm>
Use group '/' to modify global policy. Permissions can assigned <
either to the bearer of attribute or to
the accessed resource owner ('o' case) or to the group members ('m' <
case')
In any case permissions are specified as string with syntax:
    <r|-><f|-><i|-><w|->
where 'r' is read permission, 'f' is full read permission, 'i' is <
identityCtl permission and 'w' is write
permission.
Example:
    setAuthz /some/group m -f---
It will assign members of /some/group (and it's subgroups) the <
fullRead permission and remove all other
permissions (if were set).
~~~~~
Command: setIdentityStatus

```

Changes the given identity status to disable or enable it.

Syntax:

```
setIdentityStatus <dn|x509|email> <identity> <true|false>
```

The last argument equal to 'true' will make the identity enabled ( ← active) and 'false' will disable it.

Example:

```
setIdentityStatus email ann@example.com false
```

~~~~~

Command: setTime

Set time in the past for the subsequent queries. After setting the ← time all \*query\* operations will query the past, historical contents of the service.

Useful only in interactive mode

Syntax:

```
setTime [yyyy-mm-dd hh:mm:ss]
```

Without the arguments it will return to normal operation on current ← contents. Example:

```
setTime 2007-01-01 23:56:00
```

~~~~~

Command: submitApplication

Submits a new application. It is read from XML file.

Syntax:

```
submitApplication <fileWithDefinition>
```

Example:

```
setApplicationForm appl.xml
```

~~~~~

Command: updateAType

Adds or updates existing attribute type

Syntax:

```
updateAType <name> [shortDescription] [longDescription]
```

Example:

```
updateAType urn:newType 'Dummy type' 'Dummy type used to express ← foo with bar values'
```

~~~~~

Command: updateCSRApplication

Updates an application containing CSR only, when CSR is processed ( ← i.e. accepted or rejected) by a CA.

Syntax:

```
updateCSRApplication <fileWithCSR> <accepted:true|false> < ← sendEmail:true|false> [<fileWithSignedCert>]
```

File with signed certificate is needed if accepted is true.

