UNICORE COMMANDLINE CLIENT: USER MANUAL

UNICORE Team

Document Version:	1.0.0
Component Version:	6.4.2
Date:	04 11 2011

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

Contents

1 Overview

The UNICORE Commandline client (UCC) is a full-featured client for the UNICORE Grid middleware. UCC has client commands for all the UNICORE basic services, the OGSA-BES interface and the UNICORE workflow system.

It offers the following functions

- Job submission and management for both UNICORE native and OGSA-BES interfaces
- Batch mode job submission and processing with many performance tuning options
- Data movement (upload, download, server-to-server copy, etc) using the UNICORE storage management functions and available data transfer protocols
- Storage functions (ls, mkdir, ...) including creation of storage instances via storage factories
- Full UNICORE workflow system support, including the possibility to run single jobs through the resource brokering system
- Support for the UNICORE metadata system
- Information about the Grid is provided via the "system-info" and "query-cip" commands
- Various utilities like a "shell" mode, the ability to generate SAML trust delegations, low-level WSRF operations and others
- Extensibility through custom commands and the possibility to run scripts written in the Groovy programming language
- Built-in help

For more information about UNICORE visit http://www.unicore.eu.

2 Installation and configuration

2.1 Prerequisites

To run UCC, you need the Sun/Oracle or OpenJDK Java 6 JRE or SDK (or later). If not installed on your system, you can download it from this download page.

2.2 Download

You can get the latest version from the SourceForge UNICORE download page.

2.3 Installation and configuration

To install, unpack the distribution in a directory of your choice. It's a good idea to add the bin/ directory to your PATH variable,

```
export PATH=$PATH:<UCC_HOME>/bin
```

where UCC_HOME is the directory you installed UCC in.

Note
Windows only Please do not install UCC into a directory containing spaces such as "Program files".
Setting environment variables can be done (as administrator) using the Control panel→System→Extras panel.

Though you can specify your keystore location and other parameters on the commandline, it is easiest to place this information in a file, so that you do not have to key in this information repeatedly.

2.4 Preferences file

UCC checks by default whether the file <userhome>/.ucc/preferences exists, and reads it. To specify keystore, password and your preferred UNICORE 6 registry URL, the file should look as follows.

```
keystore=<your keystore>
password=<your password>
registry=<your registry URL>
```

Note
If you are worried about security, and do not want specify the password: UCC will ask for it if it is not given in the preferences or on the commandline.

Note
Windows only The preferences are usually searched in the "c:\Documents and Settings\<user_name>\.ucc" folder.
To create the .ucc folder, you might have to use the command prompt "mkdir" command.
When specifying paths in the preferences file, the backslash \ character needs to be written using an extra backslash \\

For example, if you are using a local UNICORE 6 installation for testing, you could use

```
registry=https://localhost:8080/DEMO-SITE/services/Registry?res= ↩
    default_registry
```

Note
If you wish to change the default property file location, you can set a Java VM property in the UCC start script, for example by editing the command that starts UCC

```
java .... -Ducc.preferences=<preferences location> ....
```

2.5 Security

You need a keystore in jks or pkcs12 format, that contains your user certificate and the certificate(s) of the certificate authorities that you trust. The trusted certificates can also reside in a separate file in jks format. Here is a full configuration example.

```
keystore=<your keystore>
password=<your password>
storetype=<jks or pkcs12>

#optional: configure separate truststore (must be JKS)
truststore=<your truststore>
truststorePassword=<your password>
```

2.6 Logging

UCC writes some messages to the console, more if you choose the verbose mode (-v option). If you need real logging (e.g. when using the batch mode), you can edit the <UCC_HOME>/conf/logging.properties file, which configures the Log4J logging infrastructure used in UNICORE.

2.7 Installing UCC extensions

UCC can be extended with additional commands. It is enough to copy the libraries (.jar files) of the extension into a directory that is scanned by UCC: in general these are the UCC `lib`

2.8 Testing the installation

To test your UCC installation and to get information about the resources on the Grid you're connecting to, do

```
  ucc system-info -l -v
```

3 Getting started with UCC

Assuming you have successfully installed UCC, this section shows how to get going quickly.

3.1 Getting help

Calling UCC with the "-h" option will show the available options. To get a list of available commands, type

```
ucc -h
```

```
To get help on a specific command, type
```

```
ucc <command> -h
```

See also here for a list of common options.

3.2 Connecting

First, contact UNICORE and make sure you have access to some target systems.

```
ucc connect [options]
```

3.3 List available sites

Then, list the sites available to you using

```
ucc list-sites [options]
```

3.4 Running your first job

The UCC distribution contains samples that you can run. Let's run the "date" sample. The "-v" switch prints more info so you can see what's going on.

```
ucc run [options] -v [UCC_HOME]/samples/date.u
```

Note
Look for UCC samples in the /usr/share/doc/unicore/ucc/samples directory,

---

This will run "date" on a randomly chosen site, and retrieve the output. To run on a particular site, use the "-s" option to specify a particular target system.

3.5 Listing your jobs

The command

```
ucc list-jobs [options]
```

will print a list of jobs (actually their addresses) with their respective status (RUNNING, SUCCESSFUL, etc)

4 Common options to UCC

The following table lists the options understood by all UCC commands. Most commands have additional options. You can get a summary of all available options by calling a UCC command with the "-h" option.

Table 1: Common options for the UCC

option (short and long form)	**description**
-c,--configuration <Properties_file>	Properties file containing your preferences. By default, a file *userhome/.ucc/preferences* is checked.
-k,--keystore <Keystore_file>	Keystore containing your user credential and trusted certificates
-n,--alias <Alias>	Key entry alias (by default, the first key entry is used)
-p,--password <Password>	Keystore password
-x,--storetype <jks/pkcs12>	Keystore type (default is "jks")
-T,--truststore <Truststore_file>	(optional) Truststore in JKS format containing your trusted certificates
-Y,--truststorePassword <Password>	Truststore password
-o,--output <Output_dir>	Directory for any output produced (default is the current directory)
-r,--registry <List_of_Registry_URLs>	The comma-separated list of URLs of UNICORE registries
-v,--verbose	Verbose mode
-h,--help	Print help message
-y,--with-timing	Timing mode (may not be supported by all commands)

4.1 Configuration file

By default, UCC checks for the existence of a file <userhome/.ucc/preferences> and reads default settings from there. As shown above, you can use a different file by specifiying it on the commandline using the "-c" option.

The configuration file can contain default settings for many commandline options, which are given in the form <option name>=<value> where <option name> is the long form of the option.

For example, to set your keystore and registry, the file can contain the following settings

```
keystore=user-keystore.p12
storetype=pkcs12
password=XXXXXXX
registry=https://localhost:8080/XNJS/services/Registry?res= ↩
    default_registry
```

Note
To protect your passwords, you should make the file non-readable by others, for example on Unix using a command such as *chmod 600 preferences*

4.2 Communications options

The configuration file may also contain low-level options, for example if you need to specify connection timeouts, http proxies etc.

Table 2: HTTP options for the UCC

option	**description**
http.proxyHost	HTTP(s) proxy to use
http.proxyPort	Port of the HTTP(s) proxy to use
http.nonProxyHosts	Space separated list of host name fragments for which NOT to go via the proxy. If the target URL contains such a fragment, it is accessed directly
http.connection.timeout	Timeout to use when establishing a HTTP connection
http.sockettimeout	Timeout to use when reading/writing from/to HTTP connection

For example, to set the timeout when establishing a connection to 5 seconds, you would use

```
http.connection.timeout=5000
```

5 Running jobs

5.1 Introduction

The UCC can run jobs specified in a simple job description Section 6 format. In the following it is assumed that you have installed Section 2 UCC and tried the quick start Section 3 examples.

For example, assume the file "myjob.u" looks as follows

```
{
 ApplicationName="Date", ApplicationVersion="1.0"
}
```

To run this through UCC, issue the following command

```
ucc run myjob.u
```

This will submit the job, wait for completion, download the stdout and stderr files, and place them in your default output directory. The run command has many options, to see all the possibilities use the built-in help:

```
ucc run -h
```

5.1.1 Controlling the output location and file names

Output files will be placed in the directory given by the "-o" option, if not given, the current directory is used. Also, file names will be prefixed with the job id, to assure unique file names. This behaviour can be changed using the "-b" option. When "-b" is given on the command line, files will keep their original names as specified in the job description file.

5.1.2 Specifying the site

In the example above, a random site will be chosen to execute the job. To control it, you can use the "-s" option. This will accept the name of a target system. The target systems available to you can be listed by

```
ucc list-sites
```

5.2 Options overview

The following options are available when running jobs (see also the general options overview Section 4.

Table 3: General options for UCC

Option	Short and long form	Description
-s,--sitename SITE	Site where the job shall be run	-o,--output <Output_dir>
Directory for any output produced (default is the current directory)	-b,--brief	Do not prefix the output files with the job id
-O,--stdout	specify a name for the exported standard out (by default: *stdout*)	-E,--stderr
specify a name for the exported standard error (by default: *stderr*)	-a,--asynchronous	Run asynchronously

5.3 Processing jobs asynchronously

In case of long-running jobs, you will want to run the job asynchronously, i.e. just submit the job, stage in any files and start it, in order to get the results later. UCC supports this, of course. The basic idea is that when submitting a job in asynchronous mode, a job descriptor file is written that contains the job's address, and any information about export files.

5.3.1 Asynchronous submission

Use the "-a" flag when submitting a job

```
ucc run -a <job file>
```

This will submit the job, stage-in any local files, start the job and exit. A job descriptor file (ending in ".job") will be written to your configured output directory.

5.3.2 Get the status of a particular job

The command

```
ucc get-status <job_desc>
```

will retrieve the job status. If not given on the command line, the job id will be read from the console.

5.3.3 Download results

To get stdout, stderr and other files you have marked for export in your job description, do

```
ucc get-output -o <outdir> <job_desc>
```

Here, the option "-o" specifies the directory where to put the output, by default the current directory is used. As before, the job address can also be read from the console.

5.3.4 Referencing a job by its EPR (Endpoint reference)

In case you want to check on a job not submitted through UCC, or in case you do not have the job descriptor file any more, you can also refer to a job given its EPR. For example, the "list-jobs" command will produce a list of all job EPRs that you can access.

Note that in this case UCC will only retrieve stdout and stderr files. To download other result files, you'll have to use the {{{datamovement.html}data movement}} functions.

5.3.5 Uploading and executing an executable

To upload and execute a file on a remote server, you'll need a small helper script to make the uploaded file executable and run it:

```
#!/bin/sh
chmod +x myapp
./myapp
```

Your ucc job description would then look as follows

```
{
  Executable: "/bin/sh",
  Arguments: ["helper.sh"],
  Imports: [
    {From: "helper.sh", To: "helper.sh"},
    {From: "myapp", To: "myapp"},
  ],
}
```

5.3.6 Scheduling job submission to the batch system

Sometimes a user wishes to control the time when a job is submitted to the batch queue, for example because she knows that a certain queue will be empty at that time.

Note
This feature only works with server release 6.4.0 or higher.

To schedule a job, you can either use the "-S" option to the ucc "run" command:

```
ucc run -S "12:24" ...
```

Alternatively, you can specify the start time in your job file using the "Not before" key word

```
{

 Not before: "12:30",

}
```

In both cases, the specified start time can be given in the brief "HH:mm" (hours and minutes) format shown above, or in the full ISO 8601 format including year, date, time and time zone:

```
{

 Not before: "2011-12-24T12:30:00+0200",

}
```

6 Job description format

UCC uses a simple format that allows you to specify the application or executable you want to run, arguments and environment settings, any files to stage in from remote servers or the local machine and any result files to stage out.

A number of sample files can be found in the "samples" directory of your UCC distribution. (on Linux, check also /usr/share/unicore/ucc/samples)

The format used is called JSON, and contains comma-separated key-value mappings, where the values can be simple strings, or lists of values, or maps. String values should be placed in "quotes". Comments are (inofficially) possible using the "#" hash character, as in Unix shell scrips.

Each JSON file must begin and end with curly braces "{ ... }". Several complete job samples can be found in the "samples" directory of the distribution.

Note

Note: quotes "" are needed around the keys and values in case special characters (like *:* or */*") appear, if in doubt use quotes!

To view an example job showing all available options, simply run

```
ucc run -H
```

(most of the options shown are not mandatory, of course)

Note
You may alternatively specify jobs in the JSDL format that is used internally in UNICORE 6. To do this, run UCC with the "-j" option.

6.1 Site name

You can (optionally) specify on which site (if available) the job should be run.

```
  Site: "DEMO-SITE",
```

If you do not specifiy anything UCC will select a site that will match your requirements (at least those that UCC checks for).

6.2 Specifying the application or executable

You can specify a UNICORE application by name and version, or using a (machine dependent) path to an executable file.

```
#using application name and version
{
   ApplicationName: "Date",
   ApplicationVersion: "1.0",
}
```

Note the comma-separation and the curly braces. To call an executable,

```
#using an executable

{
   Executable: "/bin/date",
}
```

6.3 Arguments and Environment settings

Arguments and environment settings are specified using a list of String values. Here is an example.

```
{

   Executable: "/bin/ls",

   Arguments: ["-l", "-t"],
```

```
   Environment: ["PATH=/bin", "FOO=bar"],

}
```

6.4 Application parameters

In UNICORE, parameters for applications are often transferred in the form of environment variables. For example, the POVRay application has a large set of parameters to specify image width, height and many more. In UCC, you can specify these parameters in a very simple way using the "Parameters" keyword:

```
{
  ApplicationName: POVRay,

  Parameters{
   WIDTH: 640,
   HEIGHT: 480,
   DEBUG: "",
  },

}
```

Note that an "empty" parameter (which does not have a value) needs to be written with an explicit empty string due to the limitations of the JSON syntax.

6.5 Job data management

6.5.1 Importing files into the job workspace

To import files from your local computer or from remote sites to the job's working directory on the remote UNICORE server, there's the "Imports" keyword. Here is an example Import section that specifies three imports:

```
{

   Imports: [

   # import a local file
    { From: "/work/data/fileName", To: "uspaceFileName" },


   # import a set of PDF files into the Uspace
    { From: "/work/data/pdf/*.pdf", To: "/" },

   # import a remote file from a UNICORE storage
```

```
    { From: "u6://DEMO-SITE/Home/testfile", To: "otherUspaceFile"  ↩
        },
   ]

}
```

If for some reason it may happen that the local file does not exist, and you want the job to run anyway, there is a flag "FailOnError" that can be set to "false" :

```
   Imports: [
   # do not fail on errors for this import:
    { From: "/work/data/fileName", To: "uspaceFileName",  ↩
        FailOnError: "false", },
   ]
```

Note
UCC supports simple wild cards ("*" and "?") for importing exporting LOCAL files, i.e. currently wildcards do not work for server-to-server imports and exports.

6.5.2 Exporting result files from the job workspace

To export files from the job's working directory to your local machine or to some remote storage, use the "Exports" keyword. Here is an example Exports section that specifies two exports:

```
{

   Exports: [
    #this exports all png files to a local directory
    { From: "*.png", To: "/home/me/images/" },

    #this exports a single file to a to local directory
    #failure of this data transfer will be ignored
    { From: "error.log", To: "/home/me/logs/error.log", FailOnError ↩
        : "false", },

    #this exports to a UNICORE storage
    { From: "stdout", To: "u6://DEMO-SITE/Home/results/myjob/stdout ↩
        " },

   ]

}
```

As a special case, UCC also supports downloading files from other UNICORE storages using the Exports keyword:

```
{
   Exports: [
    #this exports a file from a UNICORE storage
    { From: "u6://DEMO-SITE/Work/somefile", To: "/home/me/somefile" ↩
        },
   ]
}
```

The remote location can be given as a full UNICORE 6 URI, or using the more user friendly (but slower) "u6://" notation. Read more on remote locations Section 7.

Local files can be given as an absolute or relative path; in the latter case the configured output directory will be used as base directory.

The protocol to be used for imports and exports can be chosen using the "Preferred Protocols" entry, containing a space-separated list of protocols:

```
{

   Preferred protocols: "BFT RBYTEIO",

}
```

If not specified, BFT will be used.

6.5.3 Specifying credentials for data staging

Some data staging protocols supported by UNICORE require credentials such as username and password. Currently, these are "ftp" and "scp". In case you want to give username and password, the syntax is as follows

```
{
   Imports: [
     { From: "ftp://someserver:25/some/file", To: "input_data"
       Credentials: { Username: "myname", Password: "mypassword" },
     },
   ]
}
```

and similarly for exports.

6.5.4 Redirecting standard input

If you want to have your application or executable read its standard input from a file, you can use the following

```
  Stdin: filename,
```

then the standard input will come from the file named "filename" in the job working directory.

6.6 Resources

A job definition can have a Resources section specifying the resources to request on the remote system.

```
Resources: {

  #memory per CPU (bytes, you may use the common "K","M" or "G")
  Memory: 268435456 ,

  #time per CPU (seconds, use "min", "h", or "d" for other units)
  Runtime: 86400 ,

  #Total number of requested CPUs
  CPUs: 64 ,

  #you may optionally give the number of nodes
  #Nodes: 2 ,
  #together with the CPUs per node
  #CPUsPerNode: 32,

  #Custom resources (site-dependent!)
  StackLimitPerThread : 262144,

  #Operating system
  Operating system: LINUX,   #MACOS, WINNT, ...

  #Resource reservation reference
  Reservation: job1234,

}
```

Note that you can also specify a reservation reference if your batch system supports this and you have made a resource reservation.

6.7 Execution environments

To run a job in a special execution environments (as supported by the server), you can use the following syntax.

```
Execution environment: {
    Name: ...,
    Arguments: {
      ArgName1: "value1", ArgName2: "value2", ...
    },
    Options: [ ... ],
    Precommands: [ ... ],
    Postcommands: [ ... ],
    User precommand: "..." ,
```

```
      User postcommand: "..." ,
    },
```

6.8 Miscellaneous options

6.8.1 Selecting the remote login and/or group

In case you have multiple logins or Unix groups on the remote site mapped to the same credential, you can select the user name and/or group to use as follows

```
  User name: yourlogin,
  Group: yourgroup,
```

Hint: you can get a list of your logins/groups on the site by executing

```
  ucc list-sites -s SITENAME -l
```

6.8.2 Specifying a project

If the system you're submitting to does accounting, you can specify the account (or project) you want to charge the job to using the "Project" tag:

```
Project: "my_project",
```

6.8.3 Specifying the user email for batch system notifications

Some batch systems support sending email upon completion of jobs. To specify your email, use

```
  User email: foo@bar.org
```

Hint: if you want to explicitely switch off the email notification, use "NONE" as email value. This might be necessary because older UNICORE server versions try to use the email address from your certificate (if present).

6.8.4 Specifying the job name

The job name can be set simply by

```
  Name: Test job
```

6.8.5 Specifying the status check interval for batch mode

Once a job is started, it is often not useful to check its status every few seconds, because the job might be running several minutes or more. Especially in batch mode it can reduce the load on the servers if the update interval is chosen longer. This can be achieved by using the following setting (this only affects batch mode!):

```
  Update interval: 60,   #only check once a minute (default is one  ↩
      second)
```

6.8.6 Specifying the "lifetime" of the job

If you want to specify a lifetime of the job, and not rely on the server default, you can use the lifetime attribute:

```
  Lifetime: 12h,   #sec, min, h, d
```

7 Data management functions

UCC offers access to all the data management functions in UNICORE. You can upload or download data from a remote server, initiate a server-to-server transfer, create directories and so on.

7.1 Specifying remote locations

Remote locations can be specified in two ways. The first way is to use a URI that includes protocol, storage server and filename, for example

```
BFT:https://mygateway:8080/SITE/services/StorageManagement?res= ↩
    default_storage#/file
```

which specifies a file named "/file" on the storage instance "https://mygateway:8080/SITE/services/StorageManagement?res
using the BFT protocol.

Paths are relative to the storage root, not the root of the actual file system.

This explicit format is sometimes inconvenient, so you can use a shorter, more intuitive format. This is also a URI, but you need to know only the name of the virtual site (target system), and the storage or job id. For example

```
unicore6://SITE/Home/file?protocol=PROTOCOL
```

or shorter

```
u6://SITE/Home/file?protocol=PROTOCOL
```

This will resolve the current user's "Home" storage at the target system named "SITE". Note that if you do not specify the protocol, the BFT protocol will be used as default.

You can also refer to a job Uspace (the job's working directory) on a given site. For this, you will need the unique ID of that job, which you can get for example using the *list-jobs* command. For example,

```
u6://SITE/1f3bc2e2-d814-406e-811d-e533f8f7a93b/outfile
```

refers to the file "outfile" in the working directory of the given job on the "SITE" target system.

It is also possible to refer to storage services that are registered in the registry using their name, for example

```
u6://SHARE/myfiles/a_file
```

can be used to refer to the shared storage named "SHARE" if it is registered in the registry.

Though convenient, the method using "unicore6://" is much slower, and will generate some network traffic. If you do a lot of operations on the same resource, you should use the *resolve* command to find out the URI of the resource, and use that later.

7.1.1 The resolve command

This will figure out the "real" address for a "unicore6://" URL as defined above.

```
ucc resolve u6://SHARE/
```

7.2 Data movement

7.2.1 get-file

Use *get-file* to download remote files to your local machine.

Example

```
ucc get-file -s u6://DEMO-SITE/Home/test.txt -t my_test.txt
```

The "-s" (source) and "-t" (target) options are used to denote the source file(s) and the target file or directory. Wild card characters * and *?* are supported. For example,

```
ucc get-file -s u6://DEMO-SITE/Home/*.pdf -t pdfs/
```

will download all *.pdf files and write them to the "pdfs" directory (which must exist).

7.2.2 put-file

Use *put-file* to upload a local file to a remote location.

Example:

```
ucc put-file -s test.txt -t u6://DEMO-SITE/Home/test.txt
```

If you specify the "-a" option, data will be appended to an existing file.

7.2.3 copy-file

This will initiate a server-to-server data transfer. Use the "-a" option to run asynchronously, i.e. ucc will not wait for the transfer to complete. Instead, a file containing the transfer reference will be written, which can be passed to the *copy-file-status* command for status checking later.

In case the source and target file are on the same storage resource, UCC will issue the remote copy command and return immediately, as there is no need for an asynchronous mode.

Example:

```
ucc copy-file -s u6://OTHER-SITE/Home/test.txt -t u6://DEMO-SITE/ ↩
    Home/test.txt
```

Sometimes a user wishes to schedule the time when a server-to-server transfer is executed, for example because she knows that more network bandwith will be available at that time.

Note
This feature only works with server release 6.4.0 or higher.

To schedule the file transfer, you can use the "-S" option to the ucc "copy-file" command:

```
ucc copy-file -S "12:30" ...
```

The format is simply "HH:mm" (hours and minutes). Alternatively you can give the time in the full ISO 8601 format including year, date, time and time zone:

```
ucc copy-file -S "2011-12-24T12:30:00+0200" ...
```

7.2.4 copy-file-status

This will print the status of the given data transfer. As argument, it expects a file name containing the transfer reference, or directly the reference.

Example (for Unix) which captures the reference into a shell variable:

```
export ID=$(ucc copy-file -a -s u6://OTHER-SITE/Home/test.txt -t u6 ↩
    ://DEMO-SITE/Home/test.txt)
ucc copy-file-status $ID

==== Specifying the file transfer protocol

To use a different protocol from the default BFT, you can use the  ↩
    "-P" option to specify a list
of preferred protocols. UCC will try to match them with the  ↩
    capabilities of the storage and use
the first match. Your preferred protocols can also be listed in  ↩
    your preferences file using the
"protocols" key:
```

protocols=UFTP BFT

```
[NOTE]
================
If necessary, you can specify additional filetransfer options in  ↩
    your preferences file as well.
For example, to use the UFTP protocol you may want to specify the  ↩
    client host address
and the number of parallel streams explicitely:
```

uftp.client.host=your_client_hostname uftp.streams=2 #encrypt data (at the cost of performance) uftp.encryption=true

```
You can even override the UFTP server host, which can be useful in  ↩
    case the UFTP server is accessible
via multiple network interfaces:
```

uftp.server.host=myhost.com

```
UCC will try to use reasonable defaults for any missing parameters.
================
```

7.3 Handling directories

7.3.1 mkdir

This will create a directory (including required parent directories) remotely.

Example

```
ucc mkdir u6://DEMO-SITE/Home/testdirectory/data/pdfs
```

7.3.2 rm

This will remove a file or directory remotely. By default, UCC will ask for a confirmation. Use the "--quiet" or "-q" option to disable this confirmation (e.g. when using this command in scripts).

Example

```
ucc rm u6://DEMO-SITE/Home/testdirectory/data/pdfs
```

7.4 Finding data

7.4.1 ls

This will list a remote directory. Useful options are: "-l" (detailed output), "-H" (human-friendly) and "-R" (recurse). Example:

```
ucc ls u6://DEMO-SITE/Home -l -H
```

If the storage supports metadata, you can get the metadata of a single file using "ls -l -m":

```
ucc ls u6://DEMO-SITE/Home/.bashrc -l -m
```

7.4.2 find

This command is a similar to the well-known Unix utility, however much less powerful. It allows to do recursive listings and retrieve files matching certain conditions. Currently only "name match" is available. For example to get all PDF files on a storage,

```
ucc find -r -l u6://DEMO-SITE/Home/ -N .pdf
```

The *find* command is currently implemented synchronously, and may thus run into a network timeout when it takes too long. This limitation will be overcome in future versions of this command.

8 Metadata management functions

UCC offers a simple interface to access the metadata management service in UNICORE.

8.1 Basics

The metadata functions are all accessed via a single UCC command `metadata`. The actual operation to be performed is given with the "-C" (i.e. "command") option.

The storage to be operated upon is given using the "-s" option, alternatively the "-m" option can be used to directly give the metadata service URL.

In addition to the URL, the name of the target file on the storage is required.

Metadata is represented in JSON format. The metadata operations usually read metadata from a file (or write results to file), which is specified using the "-f" option.

In the following examples, `<STORAGE>` denotes the URL of a storage capable of handling metadata.

8.2 Available commands

8.2.1 creating metadata

To create metadata, a file in JSON format is required containing key-value pairs. For example, edit the file "meta.json" to contain:

```
{
 foo: bar
}
```

Say we have a file "test" on our storage, then you can create metadata as follows

```
ucc metadata -C create -f meta.json -s <STORAGE> /test
```

If you now look at the file with "ls -l -m",

```
ucc ls -l -m  <STORAGE>#/test
```

you should get something like this:

```
-rw-           3344 2011-06-27 22:32 /test
{
  "foo": "bar",
  "resourceName": "/test"
}
```

8.2.2 reading metadata

Apart from the "ls -l -m" used above, there is also an explicit "read" command, which can write the metadata to a file as well.

```
ucc metadata -C read -s <STORAGE> /test -f out.json
```

The "-f" option is optional.

8.2.3 updating metadata

Using update, the given metadata is merged with any existing metadata. Say we have a file x.json containing:

```
{
 x: y
}
```

we can append this to the existing metadata

```
ucc metadata -C update -s <STORAGE> /test -f x.json
```

Check that the metadata has indeed been appended.

8.2.4 deleting metadata

Explicitely deleting is also possible:

```
ucc metadata -C delete -s <STORAGE> /test
```

Check that the metadata has indeed been deleted.

8.2.5 searching

Searching requires a search string (according to the rules of Apache Lucene), and is triggered by the "search" command:

```
ucc metadata -C search -q "foo" -s <STORAGE> /
```

8.2.6 triggering metadata extraction

To trigger the extraction of metadata on the server, use the "start-extract" command:

```
ucc metadata -C start-extract -s <STORAGE> /
```

In this case the "/" denotes the base path from which to start the extraction process. The extraction process is asynchronous, so a "Task" service address will be returned which can be used to monitor the extraction process using the "wsrf getproperties" command.

9 Workflow extensions

9.1 Introduction

UCC supports the UNICORE workflow system and allows to submit workflows to the workflow engine or single jobs to the service orchestrator (broker).

The workflows are executed server-side, and UCC is used only for submitting, managing data and getting results.

9.2 Command overview

The following commands are provided. More details and examples follow below.

- `workflow-submit` : submit a workflow file
- `workflow-info` : list information about workflows
- `workflow-trace` : gather performance data from the workflow execution

9.3 Basic use

To check the availability of workflow services, issue the following command

```
ucc system-info -l
```

This should show at least an accessible workflow engine and service orchestrator.

The distribution contains some example workflow files in the <[?]> directory that you can edit and submit.

```
ucc workflow-submit yourworkflow.swf
```

which will submit the workflow and print the address of the workflow to standard output. To get the workflow status,

```
ucc workflow-info <workflow_address>
```

To list all your workflows, you can use the <[?]> command without an explicit workflow address

```
ucc workflow-info -l
```

9.4 Managing workflow data

During workflow execution, data files will be produced that the workflow system will move to a location on the Grid that is accessible for the individual jobs. Usually this location is created automatically by UCC before the workflow is submitted, using a special UNICORE service called a storage factory. If you want to influence this decision, UCC allows to select the storage factory to be used via the "-f" option to the workflow-submit command:

```
ucc workflow-submit -f <factory-url> <workflow-file>
```

You can check the available factories with the system-info command. If not specified, UCC will use the fist storage factory it finds in the registry.

In general, storages that are dynamically created will be deleted when the workflow is deleted. To persistently store data, you need to make sure to export important result files to a persistent location (e.g. your Home on some Grid site, or a persistent storage).

Alternatively you can directly specify a storage URL, either using the convenient "u6://..." notation, or as a real network URL:

```
ucc workflow-submit -S u6://MY-SITE/Home <workflow-file>

ucc workflow-submit -S https://my-gateway/SITE/services/ ↩
    StorageManagement?res=myuser-Home <workflow-file>
```

9.4.1 Importing local data for use by a workflow

If you have local files that need to be imported before starting the workflow, you have to specify this using a normal UCC job file that contains only an "Imports" section:

```
{
  #stage-in specification

  Imports: [
    {From: local-file.sh, To: "c9m:${WORKFLOW_ID}/input.sh"}
  ],
}
```

When submitting the workflow, add the "-u <filename>" option to specify the imports file.

This will cause UCC to copy the local file "local-file.sh" to the workflow storage space. You can refer to this file in your workflow using the "global" name "c9m:...", say in a script activity:

```
....
        <jsdl:DataStaging>
        <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
        <jsdl:FileName>input.sh</jsdl:FileName>
          <jsdl:Source>
            <jsdl:URI>c9m:${WORKFLOW_ID}/input.sh</jsdl:URI>
```

```
          </jsdl:Source>
        </jsdl:DataStaging>

....
```

The workflow system will resolve the name at runtime and your file will be used. This allows you to group your files by workflow ID.

9.4.2 Downloading output files

You can use the usual `get-file` command to download files using the "global IDs" used by the workflow engine. Hint: the `workflow-info` command will list the files that are produced by the workflow.

9.5 More

9.5.1 Tracing

The trace functionality of the workflow engine allows to retrieve some performance data, try

```
ucc workflow-trace <your_workflow_address>
```

10 Batch processing

The *batch* command allows you to run many jobs without having to start UCC each time. You can control how many jobs should go to which site. This allows efficient job processing, while putting some load on the client machine. If you need to take the client offline, you should consider using the workflow system instead, which also allows efficient high-throughput processing.

Assume you have a bunch of jobs in UCC's xref:ucc_jobdescription{job description] stored in a directory *jobs*. The output should go to a directory *out*. You can run them all through UCC using a single invocation as follows:

```
ucc batch -i jobs -o out
```

As job files, UCC will accept files ending in ".u", ".jsdl" or ".xml".

10.1 Options

You can run in "follow" mode, where UCC will watch the input directory, and will process new files as they arrive.

```
ucc batch -f -i jobs -o out
```

UCC can also process JSDL files, to batch-process these, use the "-j" option:

```
ucc batch -j -i jobs -o out
```

10.2 Performance tuning options

Getting the most performance out of UCC and your Grid installation can be a challenging task. Sending too many jobs to a site might decrease throughput, sometimes the client machine can be the limiting factor, etc.

You should experiment a bit to get the best performance for your specific setup. UCC has many options available for tuning. Here is an overview.

Table 4: Tuning options for the UCC batch mode

option (short and long form)	**description**
-K,--keep	Do not delete finished jobs on the server. By default, finished jobs are destroyed.
-m,--max <MaxRunningJobs>	Limit on jobs submitted by UCC at one time (default: 100)
-t,--threads <NumThreads>	Number of threads to be used for processing (default: 4)
-u,--update <UpdateInterval>	Minimum time in milliseconds between status requests on a single job (Default: 1000)
-R,--noResourceCheck	Do not check if the necessary application is available on the target system (will increase performance a bit)
-X,--noFetchOutcome	Do not fetch standard output and error
-S,--submitOnly	Only submit the jobs, do not wait for them to finish
-M,--maxNewJobs	Limit the number of job submissions (default: 100)
-s,--sitename	Specify which site to use
-W,--siteWeights	Specify a file containing site weights
-j,--jsdl	Assume jobs are in JSDL format instead of the default JSON .u files

10.3 Resource selection in batch mode

By default, the UCC batch mode will select a random site for running a job. You can modify the selection in different ways.

- using the "-s" option or a "Site: <sitename>," entry in the job file, you can specify the site directly
- use the "-W" option to specify a file containing site weights.

Say you have two sites where one site is a big cluster and the other a small cluster. To send more jobs to the big cluster, you can use the site weights file,

```
#example site weights file for use with "ucc batch -W ..."

BIG-CLUSTER = 100
SMALL-CLUSTER = 10

#send no jobs to this site
BAD-CLUSTER = 0

# set default weight (for any sites not specified here)
UCC_DEFAULT_SITE_WEIGHT = 10
```

This would tell UCC to send 10 times more jobs to the "BIG-CLUSTER" site, and send no jobs´to the "BAD-CLUSTER". All other sites would get weight "10", i.e. the same as "SMALL-CLUSTER".

11 OGSA-BES functions

Assuming you have successfully installed Section 2 UCC, this section shows you how to manage and monitor jobs on OGSA-BES services using UCC. The set of commands not only supports the UNICORE implementation, but may also work with implementations in other Grid middlewares compliant with OGF's OGSA-BES specification.

11.1 OGSA-BES Setup

In UNICORE style, users are required to provide a Registry URL inside the preferences file. For BES users it is not always the case that an endpoint is advertised via a UNICORE Registry. Therefore, the configuration options allow user to modify this behaviour.

```
contact-registry=[true|false]
```

Users who whish to disable UCC calling the registry can set the "contact-registry" option to false. By default the "contact-registry" option is true.

When setting "contact-registry" to false, OGSA-BES users must provide at least one BESFactory URL using the following format.

```
bes.1=https://site1.com/services/BESFactory
bes.2=https://site2.com/services/BESFactory
bes.3=https://site3.com/services/BESFactory
bes.4=file:///tmp/bes-jugene.xml
bes.5=/tmp/bes-juropa.xml
...
```

If the "contact-registry" option is set to false and no OGSA-BES URL is specified, UCC will report an error. To use an XML endpoint reference (EPR) read from a file for contacting a BESFactory service, the contents of a EPR file must validate against the WS-Addressing's endpoint reference schema. See below the contents of the sample endpoint reference file,

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/ ↩
    addressing">
  <wsa:Address>https://localhost:8080/DEMO-SITE/services/BESFactory ↩
     ?res=default_bes_factory</wsa:Address>
</wsa:EndpointReference>
```

In the above XML snippet, under the "Address" tag, you must specify the URL of a target BESFactory service.

For the sake of convenience, here is an XML infoset representation taken from the WS-Addressing specification:

```
<EndpointReference>
    <Address>xs:anyURI</Address>
    <ReferenceParameters>xs:any*</ReferenceParameters> ?
    <Metadata>xs:any*</Metadata>?
</EndpointReference>
```

11.2 Running and monitoring OGSA-BES jobs

UCC provides an easy to use command for submitting jobs on OGSA-BES complaint endpoints. To send a job read from a JSDL file,

```
ucc bes-submit-job -j hellompi.xml -s bes.3 -v
```

Alternatively, the job can be submitted using a BESFactory URL or endpoint reference file path.

```
ucc bes-submit-job -j hellompi.xml -s https://example3.com/services ↩
    /BESFactory -v
```

or

```
ucc bes-submit-job -j hellompi.xml -s file:///tmp/bes-jugene.xml -v
```

The JSON job description Section 6 can also be used, although only a subset of JSON constructs are supported for the OGSA-BES extensions.

Users can fetch the job status by specifying the descriptor (.job) file. This file is automatically generated after a successful execution of "bes-submit-job" command. Example:

```
ucc bes-job-status jobid.job
```

Job can be terminated using a job descriptor file:

```
ucc bes-terminate-job jobid.job
```

To list BESFactory properties:

```
ucc bes-list-att -s bes.1
```

The above command will result in BESFactory's properties without jobs information. To see the list of the user's jobs on a BESFactory

```
ucc bes-list-job -s bes.1
```

11.3 Enabling username/password authentication

Some BES implementations support authentication using username and password. To add username and password to the messages sent by UCC, the UCC preferences file must contain the following settings

```
#
# setup username/password
#
uas.security.out.handler.classname=de.fzj.unicore.bes.security. ↩
    UsernameOutHandler
de.fzj.unicore.bes.security.UsernameOutHandler.wsUserName=< ↩
    your_username>
de.fzj.unicore.bes.security.UsernameOutHandler.wsPassword=< ↩
    your_password>
de.fzj.unicore.bes.security.UsernameOutHandler. ↩
    wsActivateUsernameProfile=true
```

12 The UCC shell

If you want to run a larger number of UCC commands, the overhead of starting the Java VM or checking the registry may become annoying. For this scenario, UCC offers a "shell" that allows the user to enter UCC commands interactively.

It is usually started by

```
ucc shell
```

If you want to process a list of commands from a file instead of typing them, you can start the shell like this

```
ucc shell -f commandsfile
```

or on Unix you can use the redirection features

```
ucc shell < commandsfile
```

12.1 Exiting the shell

To exit, type `exit` or press CTRL-D

12.2 Changing property settings

To change a property setting in shell mode, you can use the *set* command. Without additional arguments, current properties are listed:

```
ucc>set
registry=https://...
output=/tmp
 ...
```

To set one or more properties, add space separated `key=value` strings:

```
ucc>set output=/work registry=https://....
```

13 Admin use of UCC

You can use UCC to keep track of your jobs, or, with appropriate permissions, to keep track of all the resources on a site. UCC allows to list jobs, Grid sites, and applications, including full details. Using the scripting possibilities Section 14, UCC can be extended to other administrative tasks as well.

13.1 Security considerations

Usually, each UNICORE user has only access to his or her own resources (such as jobs). For administrative use, you will need to aquire administrator privileges. There are two ways to achieve this.

- create a dedicated certificate and map it to role "admin" (in the XUUDB, or whatever attribute source you are using). This method is recommended if you want to remotely administrate UNICORE/X.
- use the server keystore (of the UNICORE/X server you want to administrate) as UCC keystore. This will also give you administrator privileges. For this you will need to be logged on to the UNICORE/X server.

13.2 Filtering lists

The UCC commands that list server-side things (list-jobs etc) accept a filtering option, that can be used to limit the results of the operation. Filtering works on the XML resource properties of the resource in question.

Filtering is enabled by the "-f" or "--filter" option of the form

```
-f XMLNAME OPERATOR VALUE
```

where XMLNAME is the name of an XML Element from the WSRF resource properties document.

For example, to list all your running jobs:

```
ucc list-jobs -f Status equals RUNNING
```

To list all jobs submitted on Nov 13, 2007:

```
ucc list-jobs -f SubmissionTime contains 2007-11-13
```

etc.

Table 5: Filtering options

operator (long and short form)	**description**
equals, eq	String equality (ignoring case)
notequals, neq	String inequality (ignoring case)
contains, c	Substring match
notcontains, nc	substring non-match
greaterthan, gt	Lexical comparison
lessthan, lt	Lexical comparison

13.3 WSRF commands

UCC supports several low-level WSRF operations using the "wsrf" command.

To destroy a resource,

```
ucc wsrf destroy <Address>
```

To get a property listing (i.e. print the XML resource property document)

```
ucc wsrf getproperties <Address>
```

To extend the lifetime of a resource

```
ucc wsrf extend <Address> <Days>
```

These commands can be abbreviated, e.g. + ucc wsrf d <Address>

14 Scripting

UCC can execute Groovy scripts. Groovy (http://groovy.codehaus.org) is a dynamic scripting language similar to Python or Ruby, but very closely integrated with Java. The scripting facility can be used for automation tasks or implementation of custom commands, but it needs a bit of insight into how UNICORE 6 and UCC work.

14.1 Script context

Your Groovy scripts can access some predefined variables that are summarized in the following table

Table 6: Variables accessible for scripts

variable	**description**	**Java type**
registry	A preconfigured client for accessing the registry	de.fzj.unicore.uas.client.IRegistryQuery
securityProperties	Security configuration (keystore, etc)	eu.unicore.security.util.client.IClientProperties
registryURL	the URL of the registry	java.lang.String
messageWriter	for writing messages to the user	de.fzj.unicore.ucc.MessageWriter
commandLine	the command line	org.apache.commons.cli.CommandLine
properties	defaults from the user's properties file	java.util.Properties

14.2 Examples

Some example Groovy scripts can be found in the samples/ directory of the UCC distribution.

Here is a script that will delete all your jobs (use at your own risk):

Groovy example: delete all your jobs

```
/*
* remove all jobs
*/

//import UNICORE/X client classes
import de.fzj.unicore.uas.client.*;

//iterate over TSSs and remove all jobs

def lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister( ↩
    registry,securityProperties,messageWriter)

lister.each {
    it.jobs.each{
         messageWriter.message "Job at "+it.address.stringValue
         new JobClient(it.address.stringValue, it,  ↩
             securityProperties).destroy()
    }
}
```

Groovy example: list available storages

```
/*
* list available storages
*/
import de.fzj.unicore.uas.client.*
import javax.xml.namespace.QName

//porttype of storage service
def SMSPORT=new QName("http://unigrids.org/2006/04/services/sms"," ↩
    StorageManagement")

//method to extract storage name from a storage client
def findName(epr){
  sms=new StorageClient(epr.address.stringValue, epr,  ↩
      securityProperties)
  return sms.resourcePropertiesDocument.storageProperties. ↩
      fileSystem.name
}

//list storages from registry
registry.listAccessibleServices(SMSPORT).each {
    name=findName(it)
```

```
    messageWriter.message "Storage <"+name+"> at "+it.address. ↩
        stringValue
}

//list storages attached to target systems
def lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister( ↩
    registry,securityProperties,messageWriter)

lister.each {
    it.storages.each{
        name=findName(it)
            messageWriter.message "Storage <"+name+"> at "+it. ↩
                address.stringValue
    }
}
```

15 Frequently asked questions

15.1 Configuration

15.1.1 Do I really have to store my password in the preferences file? Isn't this insecure?

Putting the password in a file or giving it as a commandline parameter can be considered insecure. The file could be read by others, and the commandline parameters may be visible in for example in the output of the *ps* command. Thus, UCC will simply ask for the password in case you did not specify it.

15.1.2 How can I enable more detailed logging?

UCC uses log4j, by default the configuration is done in <UCC_HOME>/conf/logging.properties You can edit this file and increase the logging levels, choose to log to a file or to the console, etc.

15.1.3 How can I set the HTTP connection timeout?

In your properties file, set

```
http.connection.timeout=<timeout in milliseconds>
```

15.1.4 How can I log the SOAP messages sent and received by UCC?

In your properties file, set

```
#log outgoing  messages
log.outgoing=true
#log incoming messages
log.incoming=true
```

which will log the messages on INFO level.

15.1.5 How can I generate a proxy cert and add it to my message in order to use e.g. GridFTP?

In your properties file, add

```
#enable proxy cert out handler
uas.security.out.handler.classname=de.fzj.unicore.uas.security. ↩
    ProxyCertOutHandler
```

which will add a handler that creates a proxy cert and adds it to the message.

15.2 Usage

15.2.1 Can I use multiple registries with UCC?

Yes. Simply use a comma-separated list of URLs for the "-c" option. However, you may only use a single key/truststore, so all registries (and sites listed in them) must accept the same security credentials.

15.2.2 Can I upload and execute my own executable?

Yes. Check the jobs section Section 5.

15.2.3 Can I use UCC to list the contents of the registry?

Using the *wsrf* command, and the UNIX grep utility, this is very easy, for example

```
ucc wsrf getproperties https://localhost:8080/DEMO-SITE/services/ ↩
    Registry?res=default_registry | grep Address
```

will list the addresses of all services registered in the registry.

15.2.4 How can I use plain JSDL files instead of a .u JSON file for job submission?

Add the "-j" option when submitting a job.

15.2.5 I get strange errors related to security

Make sure to set and use an alias for your key entry, otherwise you might experience strange errors, especially when using PKCS12 keystores. Also read the general UNICORE FAQ on www.unicore.eu[the UNICORE website] which contains descriptions of many common errors.

15.2.6 Are the JSDL documents and workflow documents validated?

The JSDL documents passed to the run command and to submit-workflow are validated, and any errors are logged. If you wish UCC to stop in case of validation errors, you need to set a property

```
ucc.validation.fail_on_errors=true
```