jLite User Manual

jLite is a Java library providing simple API for accessing a gLite based grid infrastructure. It is intended for Java developers who struggle with gLite middleware and want to reduce time and effort needed to build a cross-platform grid application on top of the EGEE grid infrastructure. jLite also includes a command-line interface which can be used as a simple cross-platform alternative to gLite User Interface on Windows and other operating systems.

Existing Java APIs for gLite are scattered among several packages and expose mostly low-level service operations. Available API usage examples often imply the presence of gLite User Interface environment installed on Scientific Linux. This complicates the use of these APIs for development of cross-platform grid applications. jLite is addressing these problems by providing a high-level Java API with functionality similar to gLite shell commands. Current implementation supports complete gLite job management lifecycle including VOMS proxy creation and delegation, transfer of job input files, job submission, job status monitoring and retrieval of job output files. Normal, collection and parametric gLite jobs are supported. The API hides complexity of underlying middleware and its configuration. jLite is easy to install because it includes all external dependencies and does not require installation of gLite User Interface. The library is pure Java and can be used on any Java-capable platform including Windows.

1 Requirements

In order to use jLite you will need a Java JRE or JDK 1.5 or higher.

Sun's original JRE/JDK is highly recommended since it contains all the needed security classes.

2 Installation

The installation of jLite is done by unpacking the distribution to any directory (referred further as JLITE_HOME).

Not that much, now to the hardest part...

3 Configuration

Configuration has always been one of the trickiest parts of grid middleware. jLite tries to make this task as easy as possible. First of all, it respects the default locations of credentials and configuration files used in Globus and gLite. So you can copy jLite to gLite UI machine and start to use it immediately, without any additional configuration (skip this chapter then). But, since in most cases you'll want to use jLite on machine without gLite installation, jLite has also its own default locations for configuration files under "JLITE_HOME/etc" directory. These locations can also be used to keep your settings separate from existing gLite installation.

Both jLite API and CLI use the same default configuration paths as described below. For jLite API, you can override these locations via methods of GridSessionConfig class. For jLite CLI, in some cases it is possible to specify non-standard locations via provided options.

Note1. The environment variables are referred further as $VARIABLE. The variable $JLITE_HOME can also be set as a Java system property. First jLite checks the environment variable, then the system property.

Note2. If you want to use jLite with GILDA VO download the archive with GILDA settings from jLite site and follow instructions inside the archive.

User Credentials

jLite uses default locations to search for user credentials as described in gLite documentation:

The user has to possess a valid X.509 certificate on the submitting machine, consisting of two files: the certificate file and the private key file. The location of the two mentioned files is assumed to be either pointed to respectively "$X509_USER_CERT" and "$X509_USER_KEY" or by "$HOME/.globus/usercert.pem" and "$HOME/.globus/userkey.pem" if the X509 environment variables are not set.

The easiest way to start on a new machine without gLite is to put your certificate and private key files to "$HOME/.globus/" directory. On Windows $HOME corresponds to "C:\Documents and Settings\

%USERNAME%".

Note that user certificate and private key files are not mandatory on the client machine, since they are only needed for the creation of the proxy certificate. As long as you have a valid proxy certificate (say, copied it from gLite UI machine) you can use jLite without installing user credentials.

You can override the default locations of user credentials via methods setUserCertPath() and setUserKeyPath() of GridSessionConfig class.

Since version 0.2 jLite also supports passing user credentials as a GlobusCredential instance via setUserCredentials() method of GridSessionConfig class.

VOMS User Proxy Certificate

jLite searches for user proxy certificate in the location pointed to by "$X509_USER_PROXY" environment variable or in "$(java.io.tmpdir)/x509up_u_$(user.name)" file if the X509 environment variable is not set.

- $(java.io.tmpdir) corresponds to "/tmp" on Linux and "C:\DOCUME~1\%USERNAME %\LOCALS~1\Temp" on Windows.
- $(user.name) is the current user name.

Note that this slightly differs from default proxy location in gLite: "/tmp/x509up_u<UID>", where <UID> is the identifier of the current Linux user. Make sure to rename a file respectively if you want to use proxy certificate from gLite UI.

You can override the default location of proxy certificate via method setProxyPath() of GridSessionConfig class.

Since version 0.2 jLite also supports passing VOMS user proxy certificate as a GlobusCredential instance via setProxy() method of GridSessionConfig class.

CA Certificates

jLite checks the following default locations of the trusted certificates directory:

- $JLITE_HOME/etc/certs/ca
- $X509_CERT_DIR
- /etc/grid-security/certificates

The check is performed in the specified order and the first existing directory/variable is chosen.

There are several ways to obtain CA certificates such as:

- Downloading certificate distribution from one of CA repositories (see http://www.igtf.net/);
- Copying certificates from "/etc/grid-security/certificates" directory on gLite UI machine to "$JLITE_HOME/etc/certs/ca".

Important! Starting with release 0.2 jLite requires signing policies (“*.signing_policy” files) to be present in the CA directory along with certificates (“*.0” files).

Note that jLite doesn't require CRL files (“*.r0”). You can use CRL files (“*.r0”), but then make sure you update them regularly (see below).

While it doesn't happen very often, CA certificates may expire. CRL files (if you have them in certificates directory) tend to expire more often. This can make jLite to fail with strange errors, so make sure you regularly update certificates and CRLs. Unfortunately there's no automatic update tool.

You can override the default location of trusted certificates directory via method setCertDir() of GridSessionConfig class.

VOMS Configuration Files

jLite checks the following default locations of the VOMS configuration files:

- $JLITE_HOME/etc/vomses
- $VOMS_USERCONF

- $HOME/.glite/vomses
- $GLITE_LOCATION/etc/vomses
- /opt/glite/etc/vomses

The check is performed in the specified order and the first existing directory/variable is chosen.

The easiest way to start on a new machine without gLite is to copy the "vomses" directory from gLite UI machine to "$JLITE_HOME/etc/vomses". You can also create VOMS configuration files manually by copying a configuration string for each VO you need from VOMS web interface and saving this string in a file named as VO.

You can override the default location of VOMS configuration files via method setVOMSDir() of GridSessionConfig class. An alternative way, in case you don't want to mess with configuration files, is to specify configuration of each VOMS server via method addVOMSServer() of GridSessionConfig class.

VOMS Certificates

jLite checks the following default locations of the VOMS certificates directory:

- $JLITE_HOME/etc/certs/voms
- $X509_VOMS_DIR
- /etc/grid-security/vomsdir

The check is performed in the specified order and the first existing directory/variable is chosen.

The easiest way to start on a new machine without gLite is to copy the "vomsdir" directory from gLite UI machine to "$JLITE_HOME/etc/certs/voms". You can also get the certificate of VOMS via its web interface by using Firefox or other browser that can save site certificate in *.pem format.

Note that VOMS certificates may expire. This can make jLite to fail with strange errors, so make sure you regularly update certificates. Unfortunately there's no automatic update tool.

You can override the default location of VOMS certificates directory via method setVOMSCertDir() of GridSessionConfig class.

WMProxy Client Configuration Files

jLite checks the following default locations of the WMProxy client configuration files:

- $JLITE_HOME/etc/wms/<VO name-lowercase>/glite_wmsclient.conf
- $HOME/.glite/<VO name-lowercase>/glite_wmsclient.conf (or glite_wms.conf)
- $GLITE_WMS_LOCATION/etc/<VO name-lowercase>/glite_wmsclient.conf (or glite_wms.conf)
- $GLITE_LOCATION/etc/<VO name-lowercase>/glite_wmsclient.conf (or glite_wms.conf)
- /opt/glite/etc/<VO name-lowercase>/glite_wmsclient.conf (or glite_wms.conf)

The check is performed in the specified order for each VO and the first existing configuration file is chosen.

The easiest way to start on a new machine without gLite is to copy "glite_wmsclient.conf" or "glite_wms.conf" files from gLite UI machine to "$JLITE_HOME/etc/wms/<VO name-lowercase>/glite_wmsclient.conf". You can also create these files manually: for each VO you need to have a "glite_wms.conf" file with the following format:

```
[
   WMProxyEndPoints = "WMPROXY_SERVICE_URI";
]
```

Note that currently jLite doesn't use any configuration parameters except WMProxyEndPoints and doesn't support multiple WMProxy endpoints per VO (the first endpoint from the list is used).

You can override the default location of WMProxy client configuration files (the parent directory with <VO name-lowercase> subdirectories inside) via method setWMSDir() of GridSessionConfig class. An alternative way, in case you don't want to mess with configuration files, is to specify WMProxy endpoints via method addWMProxy() of GridSessionConfig class.

4 Getting Started with jLite API

jLite API is pretty straightforward. Check out the Javadoc in "JLITE_HOME/doc" directory. In order to use the API in your application add all jars from “JLITE_HOME/lib” to your classpath .

The example in "JLITE_HOME/src/jlite/examples/Demo.java" will help you to get started. This example implements a typical scenario of running a grid job which includes the following steps:

1. Configure and create grid session;
2. Create user proxy;
3. Delegate user proxy to WMProxy server;
4. Load job description;
5. List matched resources;
6. Submit job to Grid;
7. Monitor job status;
8. Download job output.

Step 1 is to set configuration parameters and instantiate a grid session. Grid session is configured by a GridSessionConfig object and instantiated via a GridSessionFactory as exemplified below:

```
GridSessionConfig config = new GridSessionConfig();

// user's private key passphrase
config.setUserKeyPass("*********");

// path to CA certificates
config.setCertDir("etc/certs/ca");

// paths to VOMS configuration files and certificates
config.setVOMSDir("etc/vomses");
config.setVOMSCertDir("etc/certs/voms");

// path to WMProxy configuration files
config.setWMSDir("etc/wms");

// create Grid session
GridSession session = GridSessionFactory.create(config);
```

Step 2 is to create a VOMS user proxy by specifying VO name and proxy lifetime in seconds as exemplified below:

```
GlobusCredential proxy = session.createProxy("gilda", 43200);
```

Step 3, required before any job submissions can be made, is to delegate proxy credentials to the WMProxy server by specifying a delegation identifier as follows:

```
session.delegateProxy("myId");
```

The delegation identifier can be an arbitrary string. jLite grid session remembers the specified identifier and automatically uses it for subsequent calls to the WMProxy server.

Step 4 is to load job description in JDL format from a file. This task can be accomplished, for example, with gLite API from org.glite.jdl package:

```
JobAd jad = new JobAd();
jad.fromFile("test/normal/hello/hello.jdl");
String jdl = jad.toString();
```

The optional step 5 is to match available grid resources (computing elements, CE) to requirements specified in the job description. This can be done to assure that required resources exist and provide a list of matching resources to a user as exemplified below:

```
List<MatchedCE> ces = session.listMatchedCE(jdl);
System.out.println("Matched Computing Elements:");
for (MatchedCE ce : ces) {
    System.out.println("\t" + ce.getId());
}
```

The MatchedCE structure contains a CE identifier and its rank for the given job. The matched CEs list is sorted in decreasing rank order.

Step 6 is the job submission which is done by specifying the job description and a path to job input files (if the files are specified with relative paths in JDL):

```
String jobId = session.submitJob(jdl, "test/normal/hello");
```

Inside the submitJob() method jLite performs all the low-level steps needed in order to submit a job: registers job in WMProxy server, gets input sandbox destination URI, uploads input directory to sandbox via GridFTP, and finally starts the job. The method returns a job identifier which is used for monitoring job status, downloading job output, canceling job, etc.

Note: Current implementation supports normal, collection and parametric gLite jobs. Support for other job types is in the todo list.

Step 7 is monitoring job status until the job is done or aborted as exemplified below:

```
String jobState = "";
do {
    Thread.sleep(10000);
    jobState = session.getJobState(jobId);
    System.out.println(jobState);
} while ( !jobState.equals("DONE") &&
                !jobState.equals("ABORTED") );
```

The final step 8 is to download the job output to a specified local directory as follows:

```
if (jobState.equals("DONE")) {
    String outputDir = "test/normal/hello/" + Util.getShortJobId(jobId);
    session.getJobOutput(jobId, outputDir, true);
}
```

The third argument of the getJobOutput() method tells jLite to purge job output data from the server which puts the job in the CLEARED state.

5 jLite CLI

The jLite distribution includes a command line interface (CLI) similar to gLite UI commands. The semantics of these commands maps directly to jLite API methods so the CLI implementation is rather straightforward. Despite that, jLite CLI is not a mere demo but a real application which can be used as a cross-platform alternative to gLite UI on any Java capable operating system including Windows.

jLite CLI commands are located in "JLITE_HOME/cli" directory ("COMMAND.bat" files are for Windows, "COMMAND.sh" files are for Linux).

proxy-init

Creates a VOMS proxy. More specifically it generates a Grid proxy, contacts one or more VOMS servers,

retrieves the requested user attributes and includes them in the proxy . The command is similar to “voms-proxy-init” command on gLite UI, except that the former has less options and omits “-voms” option before required VOs/attributes.

Usage:

```
proxy-init [options] <voms>[:<command>] ...
```

options:

-debug	enables extra debug output
-help	displays usage
-limited	creates a limited proxy
-out <proxyfile>	non-standard location of new proxy cert
-proxyver <version>	version of proxy certificate {2,3,4} (default is 2)
-rfc	creates RFC 3820 compliant proxy (synonomous with -proxyver 4)
-valid <h:m>	proxy and AC are valid for h hours and m minutes (defaults to 12:00)
-vomses <path>	non-standard location of VOMS configuration files

To create a basic VOMS proxy, without requiring any special role or primary group, use:

```
$ proxy-init(.sh/.bat) <vo>
```

To create a proxy with a given role (e.g. production) and primary group (e.g. /cms/HeavyIons), the syntax is:

```
$ proxy-init(.sh/.bat) <alias>:<group name>[Role=<role name>]
```

where alias specifies the server to be contacted (see below), and usually is the name of the VO. For example:

```
$ proxy-init(.sh/.bat) cms:/cms/HeavyIons/Role=production
```

proxy-info

Prints information about an existing VOMS proxy. The command output is similar to “voms-proxy-info -all” output on gLite UI.

Usage:

```
proxy-info [options]
```

options:

-file <proxyfile>	non-standard location of proxy
-help	displays usage

proxy-delegate

Delegates a VOMS proxy to WMProxy service. The command is similar to “glite-wms-job-delegate-proxy” command on gLite UI, except the former automatically uses user name as new delegation id if not specified explicitly. If not specified, the WMProxy service endpoint is determined by the default VO of the proxy certificate.

Usage:

```
proxy-delegate [options]
```

where options are:

-d <id_string>	delegation id (default is user name)
-e <service_URL>	WMProxy service endpoint
-help	displays usage

proxy-destroy

Destroys a VOMS proxy. The command is similar to “voms-proxy-destroy” command on gLite UI.

Usage:

```
proxy-destroy [options]
```

where options are:

-file <proxyfile>	non-standard location of proxy
-help	displays usage

job-match

Prints a list of available computing elements that satisfy requirements of a given JDL file. The list of matched CEs is sorted in decreasing rank order. The command is similar to “glite-wms-job-list-match --rank” command on gLite UI, except the former automatically uses user name as existing delegation id if not specified explicitly. If not specified, the WMProxy service endpoint is determined by the default VO of the proxy certificate.

Usage:

```
job-match [options] <jdl_file>
```

where options are:

-a	automatic proxy delegation
-d <id_string>	delegation id (default is user name)
-e <service_URL>	WMProxy service endpoint
-help	displays usage

job-submit

Submits a job via WMProxy. The command is similar to “glite-wms-job-submit” command on gLite UI, except that the former has less options, introduces a new option “-in” and automatically uses user name as existing delegation id if not specified explicitly. If not specified, the WMProxy service endpoint is determined by the default VO of the proxy certificate.

Note: Current implementation supports normal, collection and parametric gLite jobs. Support for other job types is in the todo list.

Usage:

```
job-submit [options] <jdl_file>
```

options:

-a	automatic proxy delegation
-d <id_string>	delegation id (default is user name)
-debug	enables extra debug output
-e <service_URL>	WMProxy service endpoint
-help	displays usage
-in <dir_path>	search input files in the specified directory (default is current directory)
-o <file_path>	write job id to specified file
-r <ce_id>	send job to specified computing element

job-status

Retrieves the status of a job. The command is similar to “glite-wms-job-status” command on gLite UI.

Usage:

```
job-status [options] <jobId> ...
```

options:

-help	displays usage
-i <file_path>	select JobId(s) from the specified file

job-output

Retrieves the output of a job. The command is similar to “glite-wms-job-output” command on gLite UI. If not specified, the WMProxy service endpoint is determined by the default VO of the proxy certificate.

Usage:

```
job-output [options] <jobId> ...
```

options:

-dir <dir_path>	store output files in the specified directory (default is CURRENT_DIR/JOB_ID)
-e <service_URL>	WMProxy service endpoint
-help	displays usage
-i <file_path>	select JobId(s) from the specified file
-list	do not retrieve output files, just print their URIs
-nopurge	do not purge job output after retrieval

job-cancel

Cancels a job. The command is similar to “glite-wms-job-cancel” command on gLite UI. If not specified, the WMProxy service endpoint is determined by the default VO of the proxy certificate.

Usage:

```
job-cancel [options] <jobId> ...
```

options:

-e <service_URL>	WMProxy service endpoint
-help	displays usage
-i <file_path>	select JobId(s) from the specified file

6 Test Jobs

You might find useful job examples used for testing jLite located in JLITE_HOME/test:

- /collection – example of collection job
- /normal – examples of normal job
  - hello – prints “Hello Grid!” and hostname of worker node via custom shell script
  - hostname – prints hostname of worker node, has no input files
- /paramteric – example of parametric job

7 Troubleshooting

When something goes wrong jLite can return rather cryptic errors from the underlying Globus/gLite libraries. Better error reporting is in the todo list. Until then here you will find typical errors, possible solutions and instructions on debugging.

Typical Errors

java.lang.IllegalArgumentException: Cannot find file usercert.pem

User credentials are not found. Check your configuration.

org.glite.voms.contact.VOMSException: No user credentials found!

User credentials are not found or wrong private key passphrase.

Could not find VOMS server info for VO: vo_name

VOMS server configuration is not found for vo_name. Check your configuration.

org.glite.voms.contact.VOMSException: Error instantiating PKIVerifier: Directory /etc/grid-security/vomsdir doesn't exist on this machine! Please specify a value for the vomsdir directory or set the VOMSDIR system property.

Directory with VOMS server certificates is not found. Check your configuration.

org.glite.voms.contact.VOMSException: AC validation failed!

Can be caused by missing VOMS server certificate. Check your configuration.

org.glite.voms.contact.VOMSException: Error communicating with server xxx:Authentication failed [Caused by: Failure unspecified at GSS-API level [Caused by: Unknown CA]]

Can be caused by missing CA certificate. Check your configuration.

java.io.IOException: No CA files found matching "/etc/grigrid workflow workshopd-security/certificates/*.0

CA certificates are not found. Check your configuration.

Could not find WMProxy server for VO: vo_name

WMProxy server configuration is not found for vo_name. Check your configuration.

Debugging

The errors you may see can be caused by many reasons, such as missing local configuration, improperly configured or failed remote grid service, network connectivity problems etc. This makes debugging such problems quite hard. The best way is to gather maximum information by enabling full debug of Globus/gLite/jLite libraries:

- Open file “JLITE_HOME/log4j.properties” and uncomment last four lines,
- Put this file in your application's classpath (for jLite CLI it is not needed),
- Now all debug messages should be saved in “debug.log” file in the current working directory.

If you need help submit your error report with debug log to jLite mailing list:
http://groups.google.com/group/jlite