UNICORE GATEWAY

UNICORE Team

Document Version:	1.0.5
Component Version:	6.4.3
Date:	04 05 2012

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

Contents

1 Introduction **1**

2 Installation **2**

2.1 Prerequisites 2

2.2 Installation from the core server bundle 2

2.3 Installation from RPM package (RedHat distributions) 3

2.4 Installation from the DEB package (Debian distributions) 3

3 Configuration **3**

3.1 startup.properties 3

3.2 connections.properties 3

3.3 gateway.properties 4

3.4 security.properties 4

3.5 Dynamic registration of Vsites 5

3.6 Disabling the detailed Vsite listing (monkey page) 6

3.7 Server parameters 6

3.8 Scalability settings 6

3.9 Proxy certificate support 7

3.10 AJP Connector for using Apache httpd as a frontend 7

3.11 Gateway plugins 9

3.12 logging.properties 9

4 Web interface ("monkey page") **11**

5 Using the Gateway for failover and/or loadbalancing of UNICORE sites **11**

5.1 Configuration 11

5.2 Available Strategies 12

6 Building the Gateway from source **13**

This is a gateway user manual providing information on running and using the UNICORE 6 gateway. Please note also the following places for getting more information:

UNICORE Website: http://www.unicore.eu

Support list: unicore-support@lists.sf.net

Developer's list: unicore-devel@lists.sf.net

1 Introduction

The Gateway is the entry point into a UNICORE site. It is installed in front of any networking firewall. It authenticates incoming messages and forwards them to their intended destination. The gateway receives the reply and sends it back to the client.

In effect, traffic to a "virtual" URL, e.g. "https://mygateway:8088/Alpha" is forwarded to the real URL, e.g. "https://host1:7777". In this way, only a single open port in a site's firewall has to be configured. (Technical note: this process only works for "most" HTTP requests (i.e. POST, GET and PUT), and is not a complete HTTP reverse proxy implementation).

The mappings of virtual URL to real URL for the available sites are listed in a configuration file "connections.properties". Additionally, the gateway supports dynamic registration of sites.

The second functionality of the gateway is authentication of incoming requests. Connections to the gateway are made using client-authenticated SSL, so the gateway will check whether the caller presents a certificate issued by a trusted authority.

IMPORTANT NOTE ON PATHS
UNICORE Gateway is distributed either as a platform independent and portable bundle (as a part of the UNICORE core server package) or as an installable, platform dependent package such as RPM.
Depending on the installation package used paths to various Gateway files are different. If installing using distribution-specific package the following paths are used:

```
CONF=/etc/unicore/gateway
BIN=/usr/sbin
LOG=/var/log/unicore/gateway
```

If installing using portable bundle all Gateway files are installed under a single directory. Path prefixes used then are as follows, where INST is a directory where Gateway was installed:

```
CONF=INST/conf
BIN=INST/bin
LOG=INST/log
```

The above variables (CONF, BIN and LOG) are used throughout the rest of this manual.

2 Installation

UNICORE Gateway is distributed in the following formats:

1. As a part of platform independent installation bundle called UNICORE core server bundle. The UNICORE core server bundle is provided in two forms: one with a graphical installer and one with a command line installer.
2. As a binary, platform-specific packages available currently for Scientific Linux 5, Scientific Linux 6 and Debian 6 platforms. Those packages are tested on the enumerated platforms, but should work without any problems with other versions of similar distributions (e.g. version for SL6 works well on Centos 6 or recent Fedora distributions. Differences between SL5 and SL6 version are only in the RPM tools used to create packages (so SL5 version should be more universal, while SL6 version can require a newer rpm software).

2.1 Prerequisites

UNICORE Gateway is a lightweight service and can be run on any commodity hardware, including virtual machines.

In case of platform specific package (RPM/deb) all software requirements are handled automatically. If installing from the UNICORE quickstart package the common UNICORE requirements must be satisfied:

1. Java JRE (or JDK) in version 1.6 (OpenJDK or Oracle are suggested).
2. Python - only for installation on UNIX platforms.
3. The BASH shell and common utilities (find, grep) - only for UNIX platforms.

UNICORE Gateway is used to as an access point to UNICORE servers therefore must be accessible from the outside world and internal site servers like Unicore/X or UNICORE Registry must have a network connectivity to the Gateway machine.

2.2 Installation from the core server bundle

Download the core server bundle from the UNICORE project website.

If you use the graphical installer follow the on-screen instructions and do not forget to check click the Gateway checkbox when prompted.

If you use the console installer then for generic installation instruction review the README file available after extracting the bundle. You don't have to change any defaults as the Gateway is installed by default.

2.3 Installation from RPM package (RedHat distributions)

The preferred way is to use Yum to install (and subsequently update) the Gateway.

To perform the Yum installation, EMI Yum repository must be installed first. Refer to the EMI release documentation (available at the EMI website http://www.eu-emi.eu/releases) for detailed instructions. Typically installation of the EMI repository requires to download a single RPM file and install it.

After the EMI repository is configured, the following command installs Gateway:

```
$> yum install unicore-gateway
```

2.4 Installation from the DEB package (Debian distributions)

The preferred installation way is to use apt to install and subsequently update the Gateway.

To perform the apt installation, EMI apt repository must be installed first. Refer to the EMI release documentation (available at the EMI website http://www.eu-emi.eu/releases) for detailed instructions. Typically installation of the EMI repository requires to download a single DEB file and install it.

After the EMI repository is configured, the following command installs the Gateway:

```
$> apt-get install unicore-gateway
```

3 Configuration

The gateway is configured using a set of configuration files, which usually reside in the `CONF` subdirectory.

3.1 startup.properties

This file contains settings related to the Java VM, such as the Java command to use, memory settings, library paths etc.

3.2 connections.properties

This is a simple list connecting the names of sites and their physical addresses. An example is:

```
DEMO-SITE = https://localhost:7777
REGISTRY = https://localhost:7778
```

If this file is modified, the gateway will re-read it at runtime, so there is no need to restart the gateway in order to add or remove sites.

Additionally sites may register dynamically at runtime.

3.3 gateway.properties

Use the “hostname” property to configure the network interface and port the gateway will listen on, and to select between https (recommended!) and http.

```
hostname = https://localhost:8080
```

Note
If you set the host to “0.0.0.0”, the gateway will listen on all network interfaces of the host machine, else it will listen only on the specified one.

If the scheme of the hostname URL is set to https, the Gateway uses the configuration data from security.properties to configure the HTTPS listener.

3.4 security.properties

```
keystore=/keystore/path
keystorepassword=xxx
truststore=/truststore/path
truststorepassword=xxx
```

The gateway supports the usual JKS (.jks) and PKCS12 (.p12) keystore / truststore types. These are determined automatically using the file extension.

As truststore types, you may also use “file” or “directory” by explicitely setting the “truststore-type” parameter. In case of “file”, the “truststore” parameter is assumed to denote a single file containing trusted certificates in PEM format. In case of “directory”, the truststore setting is assumed to denote a directory. This directory will be scanned for files in PEM format (having the file extension “.pem” or “.cert”), which will be loaded as trusted certs. For example, to load all the pem files in `/etc/security/trusted`, you can set

```
truststore=/etc/security/trusted
truststoretype=directory
```

CRL checking

The gateway supports CRL checking if the CRLs are available as local files or on the network. To enable CRL checking, you need to ensure that the gateway start script sets the following system property:

```
OPTS=$OPTS” -Dcrlmanager.properties.file=<path to properties file>”
```

The start script is found in `BIN` directory after installation and is named `start.sh" (in case of installation from Java bundle) or +unicore-gateway-start.sh` (in case of installation from distribution-specific package).

Windows note: you can set system properties in the wrapper.conf file.

The properties file contains the following settings:

```
#crl checking mode: STRICT, LAX or NONE
crlcheck.mode = LAX

#update interval in seconds
crlcheck.interval = 86400

# list of URLs to CRLs
1.crl.url = http://ca.grid-support.ac.uk/pub/crl/ca-crl.crl
2.crl.url = file:///path/to/local/CRLFile
```

In LAX mode, CRLs are checked but it is not an error if for a given CA no CRL has been loaded. In STRICT mode, a CRL must be checked for each certificate authority. Finally, NONE means no CRL checking. The update interval is given in seconds. The CRL location is given as a URL. Local files, HTTP servers, anonymous FTP etc can be used for CRL distribution.

Note
currently HTTPS connections will use the usual Java security, so it might be necessary to add the CRL server certificate to the Java truststore (i.e. the Gateway truststore is not used for this purpose).

3.5 Dynamic registration of Vsites

Dynamic registration is controlled by three properties in `CONF/gateway.properties` file:

```
registration.enable=true
```

If set to true, the gateway will accept dynamic registrations which are made by sending a HTTP POST request to the URL /VSITE_REGISTRATION_REQUEST

Filters can be set to forbid access of certain hosts, or to require certain strings in the Vsite addresses. For example

```
registration.deny=foo.org example.org
```

will deny registration if the remote hostname contains “foo.org” or “example.org”. Conversely,

```
registration.allow=mydomain.org
```

will only accept registrations if the remote address contains “mydomain.org”. These two (deny and allow) can be combined.

3.6 Disabling the detailed Vsite listing (monkey page)

To disable the Vsite details page, set

```
#disable details display on the web page
webpage.disable=true
```

3.7 Server parameters

To fine-tune the operational parameters of the embedded Jetty server, you can set some properties. These are the defaults:

```
#minimum number of threads
jetty.minThreads=1
#maximum number of threads
jetty,maxThreads=255
#time after which an idle connection will be terminated
jetty.maxIdleTime=3000
#same, but under low resource conditions
jetty.lowResourceMaxIdleTime=100
#low resource threshold
jetty.lowThreads=50
```

You can use the non-blocking IO connector offered by Jetty:

Note
due to a known issue (see http://jira.codehaus.org/browse/JETTY-387) the NIO connector has been reported to not work properly for large messages (e.g. when transfering files using BFT). Therefore we currently discourage the use of the NIO connector until this issue is resolved.

To enable the use of the NIO connector, set

```
#use NIO connector
jetty.useNIO=true
```

This will scale up to higher numbers of concurrent connections than the default connector.

3.8 Scalability settings

The gateway acts as a https client for the VSites behind it. The number of concurrent calls is limited, and controlled by two parameters

```
# maximum total number of concurrent calls to Vsites
http.connection.maxTotal=100
# total number of concurrent calls per site
http.connection.maxPerService=20
```

You can also control the limit on the maximum SOAP header size which is allowed by the gateway. **Typically you don't have to touch this parameter**. However if your clients do produce very big SOAP headers and gateway blocks them you can increase the limit. Note that such a giant SOAP header usually means that the client is not behaving in a sane way, e.g. is trying to perform a DoS attack.

```
# maximum size of an accepted SOAP header, in bytes
soapMaxHeader=102400
```

Note that gateway may consume this amount of memory (plus some extra amount for other data) for each opened connection. Therefore, this value multiplied by the number of maximum allowed connections, should be significantly lower, then the total memory available for the gateway.

3.9 Proxy certificate support

UNICORE optionally supports proxy certificates as used by other Grid middleware systems. In general, we think proxies are a bad idea, but for interoperability purposes, proxies are supported as an extension. To enable the proxy certificate support,

```
#enable proxy authentication
proxyAuthentication=true
#optional validator class name (default: eu.unicore.proxy. ↩
    RFC3820ProxyValidator)
proxyValidator=<class name>
```

You need to install the necessary extension libraries into the gateway lib directory. The proxy extension is available separately from Sourceforge, if in doubt, ask on the UNICORE support or development mailing lists.

3.10 AJP Connector for using Apache httpd as a frontend

If you wish to use the Apache webserver (httpd) as a frontent for the gateway (e.g. for security or fault-tolerance reasons), you can enable the AJP connector instead of the usual one.

Requirements

- Apache httpd
- mod_jk for Apache httpd

Enabling AJP13 Gateway with Jetty and mod_jk

Enable "mod_ssl" module in httpd configuration files, for instance "ssl.conf":

```
 LoadModule ssl_module modules/mod_ssl.so
 Listen 443
```

Enable "mod_jk" module in httpd configuration files, for instance "jk.conf":

```
LoadModule jk_module modules/mod_jk.so
JkWorkersFile "conf.d/worker.properties"
```

Define a "mod_jk" worker, to dialog with Gateway's AJP connector, in "worker.properties" configuration file as referred in the above "jk.conf":

```
worker.list=jetty
worker.jetty.port={{gateway_port}}
worker.jetty.host={{gateway_ip}}
worker.jetty.type=ajp13
worker.jetty.lbfactor=1
```

Configure a httpd virtual SSL host, dedicated to act as Gateway's frontend, in the httpd configuration files, for instance "unicore.conf". This virtual host lets the "mod_jk" worker defined above in "worker.properties", manage all the requests received and forwards the full client's certificate chain:

```
# Apache VirtualHost configuration to serve as frontend
# for an "AJP connector enabled" UNICORE6 Gateway
<VirtualHost {{frontend_ip}}:{{frontend_port}}>
    # Pass every request on this VirtualHost to the jetty worker
    # defined in mod_jk's worker properties file.
    JkMount /* jetty
    # Pass SSL_CLIENT_CERT environment variable to the AJP  ↩
        connector
    JkOptions +ForwardSSLCertChain

    # Log
    ErrorLog "logs/unicore_error_log"
    TransferLog "logs/unicore_access_log"
    JkLogFile "logs/unicore_mod_jk.log"

    # Enable SSL
    SSLEngine on

    # Export SSL-related environment variables, especially  ↩
        SSL_CLIENT_CERT
    # which contains client's PEM-encoded certificate
    SSLOptions +StdEnvVars +ExportCertData

    # Client have to present a valid certificate
    SSLVerifyClient require

    # Server certificate
    SSLCertificateFile "{{/path/to/httpd_cert.pem}}"
    SSLCertificateKeyFile "{{/path/to/httpd_key.pem}}"

    # Trusted CAs
    SSLCACertificateFile "{{/path/to/cacert.pem}}"
</VirtualHost>
```

On the Gateway, enable Jetty AJP connector instead of its HTTP connector in Gateway configuration file `CONF/gateway.properties`:

```
#enable AJP connector
jetty.ajp=true
```

External references

- Configuring AJP13 using mod_jk or mod_proxy_ajp - Jetty http://docs.codehaus.org/display/-JETTY/Configuring+AJP13+Using+mod_jk
- The Apache Tomcat Connector - Webserver HowTo http://tomcat.apache.org/connectors-doc/-webserver_howto/printer/apache.html

3.11 Gateway plugins

The gateway supports tunneling other protocols through its socket. This is still experimental, so use at your own risk. To establish the tunnel, a special HTTP message is sent to the gateway:

```
HEAD / HTTP/1.1
Upgrade: YOURPROTOCOLNAME
```

the method must be “HEAD”, and the message must contain the “Upgrade” header. The gateway replies:

```
HTTP/1.1 101 Switching protocols
Upgrade: YOURPROTOCOLNAME
```

After this the gateway’s socket connection is passed to a custom handler, which can read data from the client and write replies. The handler can be configured in `CONF/gateway.properties`:

```
protocolPlugin.YOURPROTOCOLNAME=your.handler.classname
```

The handler class must implement the eu.unicore.gateway.util.ProtocolPluginHandler interface. For more details please refer to the sourcecode of the plugin interface class.

3.12 logging.properties

The Gateway uses Log4j for logging, configured using a properties file (`CONF/logging.properties`). Edit this file if you need to change the granularity of the logging, or where the logs are written to. By default, the logs are written to files in the logs folder, and files are rolled over daily. Since the 6.2.0 release, you change the log levels at runtime by editing and saving the configuration file (`CONF/logging.properties by default`). The following logger names exist:

unicore.gateway	General gateway logging
unicore.gateway.connections	Log IPs of clients, and the DN after the SSL handshake
unicore.gateway.jetty	HTTP processing, Jetty server
unicore.gateway.security	Certificate details and other security

The recent versions of the Gateway uses the so called MDC (Mapped Diagnostic Context) to provide additional information on the client which is served. In the MDC the client's IP address and client's Distinguished Name is stored. You can control whether to attach MDC to each

Here is an example logging.properties file

```
# Set root logger level to INFO and its only appender to A1.
log4j.rootLogger=INFO, A1

# A1 is set to be a file appender with date rollover
log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=logs/gateway.log

# configure daily rollover: once per day the log will be copied
# to a file named e.g. gateway.log.2008-12-24
log4j.appender.A1.DatePattern='.'yyyy-MM-dd

# A1 uses the PatternLayout
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] [%X{clientIP} %X ←
    {clientName}] %-5p %c{1} - %m%n

# log levels
log4j.logger.unicore.gateway=INFO
log4j.logger.unicore.gateway.connections=INFO
log4j.logger.unicore.gateway.jetty=INFO
log4j.logger.unicore.gateway.security=INFO

# also configure JDK (java.util.logging) logging
handlers=java.util.logging.ConsoleHandler
# Default global logging level.
# Loggers and Handlers may override this level
.level=SEVERE
```

More details on Log4j configuration file format can be found here: http://logging.apache.org/-log4j/1.2/manual.html

Detailed list of variables of the PatternLayout's pattern are listed here: http://logging.apache.org/-log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

4 Web interface ("monkey page")

For testing and simple monitoring purposes, the gateway displays a website showing detailed site information (the details view can be disabled). Once the Gateway is running, open up a browser and navigate to https://<gateway_host>:8080 (or whichever URL the gateway is running on). As the gateway usually runs using SSL, you will need to import a suitable client certificate into your web browser.

A HTML form for testing the dynamic registration is available as well, by clicking the link in the footer of the main gateway page.

5 Using the Gateway for failover and/or loadbalancing of UNICORE sites

The Gateway can be used as a simple failover solution and/or loadbalancer to achieve high availability and/or higher scalability of UNICORE/X sites without additional tools.

A site definition (in `CONF/connections.properties`) can be extended, so that multiple physical servers are used for a single virtual site.

An example for such a so-called multi-site declaration in the connections.properties file looks as follows:

```
#declare a multisite with two physical servers

MYSITE=multisite:vsites=https://localhost:7788 https://localhost ↩
    :7789
```

This will tell the gateway that the virtual site "MYSITE" is indeed a multi-site with the two given physical sites.

5.1 Configuration

Configuration options for the multi-site can be passed in two ways. On the one hand they can go into the connections.properties file, by putting them in the multi-site definition, separated by ";" characters:

```
#declare a multisite with parameters

MYSITE=multisite:param1=value1;param2=value2;param3=value3;...
```

The following general parameters exist

vsites	List of physical sites
strategy	Class name of the site selection strategy to use (see below)
config	Name of a file containing additional parameters

Using the "config" option, all the parameters can be placed in a separate file for enhanced readability. For example you could define in connections.properties:

```
#declare a multisite with parameters read from a separate file

MYSITE=multisite:config=conf/mysite-cluster.properties
```

and give the details in the file "conf/mysite-cluster.properties":

```
#example multisite configuration
vsites=https://localhost:7788 https://localhost:7789

#check site health at most every 5 seconds
strategy.healthcheck.interval=5000
```

5.2 Available Strategies

A selection strategy is used to decide where a client request will be routed. By default, the strategy is "Primary with fallback", i.e. the request will go to the first site if it is available, otherwise it will go to the second site.

Primary with fallback

This strategy is suitable for a high-availability scenario, where a secondary site takes over the work in case the primary one goes down for maintenance or due to a problem. This is the default strategy, so nothing needs to be configured to enable it. If you want to explicitely enable it anyway, set

```
strategy=primaryWithFallback
```

The strategy will select from the first two defined physical sites. The first, primary one will be used if it is available, else the second one. Health check is done on each request, but not more frequently as specified by the "strategy.healthcheck.interval" parameter. By default, this parameter is set to 5000 milliseconds.

Changes to the site health will be logged at "INFO" level, so you can see when the sites go up or down.

Round robin

This strategy is suitable for a load-balancing scenario, where a random site will be chosen from the available ones. To enable it, set

```
strategy=roundRobin
```

Changes to the site health will be logged at "INFO" level, so you can see when the sites go up or down.

Custom strategy

You can implement and use your own failover strategy, in this case, use the name of the Java class as strategy name:

```
strategy=your_class_name
```

6 Building the Gateway from source

To checkout the latest version of the Gateway source code, do

```
svn co http://unicore.svn.sourceforge.net/svnroot/unicore/gateway/ ↩
    trunk gateway
```

You will need to install Maven from http://maven.apache.org Compile using

```
mvn install
```

Compiles the code and runs the tests.

```
mvn assembly:assembly
```

builds distribution archives in zip and tar.gz format in the target/ directory