



GridLab - A Grid Application Toolkit and Testbed

## **"GRMS v.1.9.6 - Administator Guide"**





## Contents

<b>1. Introduction.....</b>	<b>3</b>
1.1. GRMS Overview.....	3
1.2. Requirements for the GRMS version 1.9.....	4
1.3. Architecture of the GRMS.....	5
1.3.1GRMS Service and Core Services.....	7
1.3.2GRMS Service and other GridLab Middleware Services.....	7
<b>2. GRMS Web Service Interface.....</b>	<b>10</b>
2.1. Overview.....	10
2.2. Requirements.....	10
2.3. Installation of required components.....	11
2.3.1. Tomcat installation and configuration.....	11
2.3.2. Instalng and configuring MyProxy Server.....	13
2.4. Deploying GRMS Service.....	13
2.5. Starting the GRMS Service.....	16
<b>3. Broker Service.....</b>	<b>17</b>
3.1. Overview.....	17
3.2. Requirements.....	17
3.3. Instalation of required components.....	17
3.3.1. PostgreSQL installation and configuration.....	17
3.4. Configuring Broker Service.....	18
3.4.1. Instalation.....	18
3.4.2. Configuraton.....	18
3.5. Starting Broker Service.....	27
<b>4. Command-Line Client.....</b>	<b>28</b>
4.1. Overview.....	28
4.2. Requirements.....	28
4.3. Installation and configuration of GRMS client.....	28
4.4. Executing GRMS Client.....	29
<b>5. References.....</b>	<b>31</b>



## 1. Introduction.

### 1.1. GRMS Overview.

The GridLab [3] Resource Management System (GRMS) is an open source meta-scheduling system, developed under the GridLab IST-2001-32133 project, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. The GRMS, based on dynamic resource selection, mapping and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic Grid environment and resource management challenges, e.g. load-balancing among clusters, remote job control or file staging support. Therefore, the main goal of the GRMS is to manage the whole process of remote job submission to various batch queuing systems, clusters or resources. It has been designed as an independent core component for resource management processes which can take advantage of various low-level Core Services and existing technologies. Finally, the GRMS can be considered as a robust system, which provides abstraction of the complex grid infrastructure as well as a toolbox, which helps to form and adapts to distributing computing environments.

The current release of the GRMS is based on the Globus 2.X and uses Globus Core Services deployed on resources. The GRMS supports Grid Security Infrastructure by providing the GSI-enabled web service interface for all clients, e.g. portals or applications, and thus can be integrated with any other middleware grid environment. The GRMS has been developed entirely in Java and thus could be installed on many different kinds of operating systems and resources. One of the main assumptions for the GRMS is to perform remote jobs control and management in the way that satisfies Users (Job Owners) and their applications requirements. All users requirements are expressed within XML-based resource specification documents and sent to the GRMS as SOAP requests over GSI transport layer connections.

Simultaneously, Resource Administrators (Resource Owners) have full control over resources on which all jobs and operations will be performed by appropriate GRMS setup and installation. Note, that the GRMS together with Core Services reduces operational and integration costs for Administrators by enabling grid deployment across previously incompatible cluster and resources.

The current release of the GRMS (v1.9) allows Users to use the following functionalities:

- 1) ability to choose dynamically the best available resource or queuing system for the remote job execution according to provided Job Description, GRMS configuration and Administrator preferences,
- 2) ability to stage-in and stage-out files (input files, output files, stdin, stdout, stderr) required by jobs and users before and after executions according to provided Job Description,
- 3) ability to setup environments before and after job execution according to provided Job Description,
- 4) ability to submit and control a job remotely,
- 5) ability to cancel a job,



- 6) ability to get a complex information about the job,
- 7) ability to get a complex information about the history of the job,
- 8) ability to check a list of candidate resources for job and its Job Description,
- 9) ability to check a list of jobs submitted by an application user,
- 10) ability to deal with projects as a groups of jobs,
- 11) to receive the notification, when the statuses of the job or the request have changed.

Moreover, due to a specific interface for registration of applications in the GRMS, more complex and more dynamic application scenarios are supported now. This functionality is available for all applications, in particular GAT enabled applications [2], which are able to register callback information in the GRMS and then wait for a checkpoint GRMS's call. Once the GRMS's call is received by application a process of checkpointing begins. In this case, it is an application-level checkpoint in which the application is obliged to store all information required for restart in one or more checkpoint files (described also in Job Description). Therefore, we have listed below two additional functionalities of the GRMS:

- ability to register callback information in the GRMS,
- ability to migrate a job to a better resource according to provided Job Description.

## **1.2. Requirements for the GRMS version 1.9**

All requirements for the GRMS are specified in the Technical Specification Document [1]. However, the functionality of the current release of the GRMS is limited to the aforementioned list and fulfills the following requirements :

- to act in behalf of users on resources and meet application requirements concerning resources and their environment,
- to run and control precompiled batch jobs remotely,
- to run and control precompiled MPI batch jobs remotely,
- to run Java applications remotely,
- to register applications and receive unique JOB IDs,
- to checkpoint applications remotely,
- to migrate applications remotely,
- to store all historic information about job statuses and resources which have been used during a job submission process,
- to contact the Information Service to receive static and dynamic information about resources,
- to register GRMS in an Information Service,
- to contact an Adaptive Components Service to get additional information about distributed resources and networks,
- to stage-in and stage-out files required by jobs before and after executions using Core Services (GridFTP/GASS/FTP) or GridLab Middleware Services (Replica Catalog Service and Data Movement Service).



### **1.3. Architecture of the GRMS**

The current release of the GRMS is composed of the following modules (Fig. 1):

- Broker Module
  - Steering process of job submission
  - Choosing the best resources for job execution (scheduling algorithm)
  - Transferring input and output files for job's executable
- Resource Discovery Module
  - Finding resources that fulfills requirements described in Job Description
  - Providing information about resources, required for job scheduling
- Job Manager Module
  - Checking status of running job
  - Canceling running job
  - Monitoring for status changes of running job
- Job Registry Module
  - Storing information concerning job execution
  - Serves information about th job
- Job Queue
  - Providing internal job queueing
  - Providing ability to implement different queue management algorithms

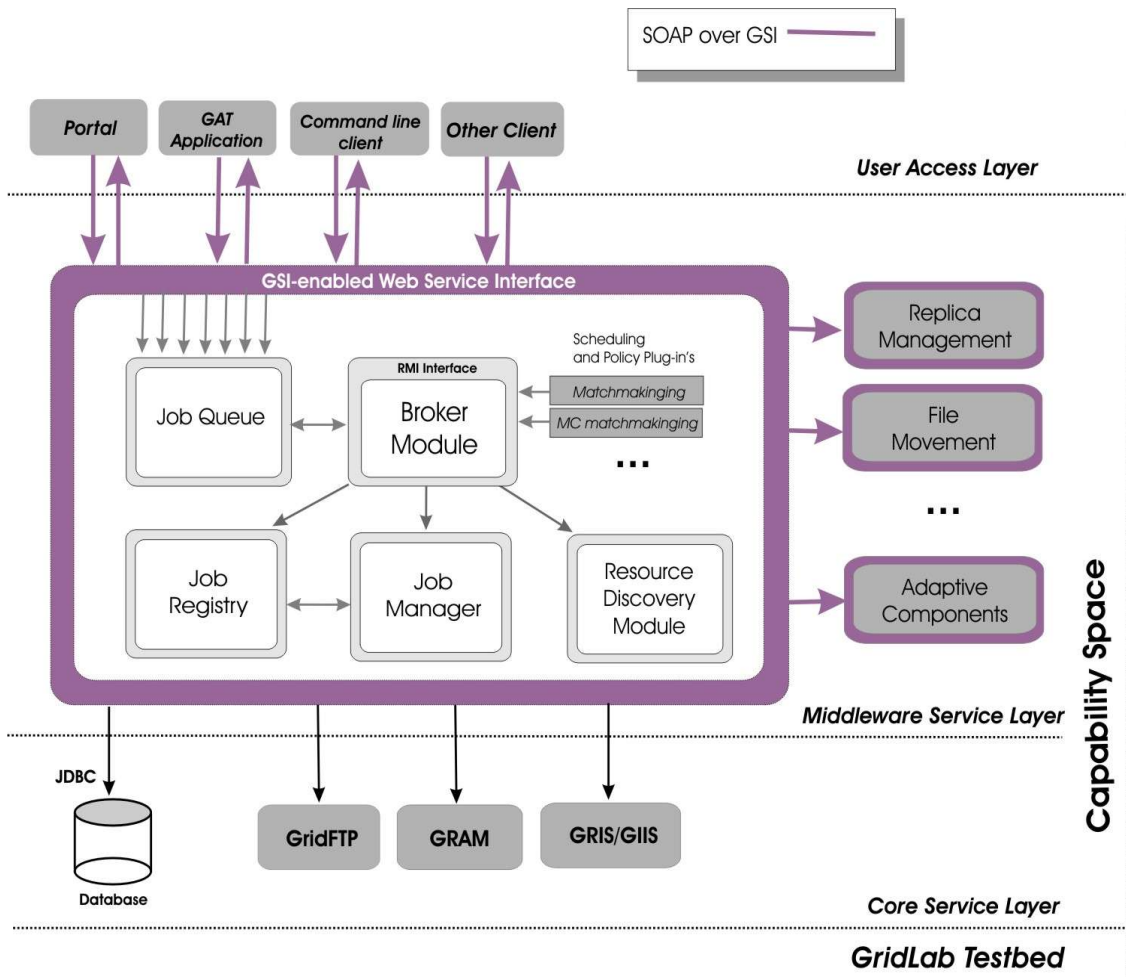


Fig. 1GRMS Architecture

The aim of the Broker Module is to control the whole process of resource and job management within the GRMS. This module steers a flow of requests to the GRMS and is also responsible for an appropriate cooperation with other modules. In the current release of the GRMS, the Broker Module contains basic scheduling and policy strategies: matchmaking and multi-criteria matchmaking. The first strategy is relatively simple but in fact very efficient approach for managing resources on which the advanced reservation is not possible. The second strategy allows more flexible and more accurate resource selections according to both users and administrator's requirements and preferences. These two strategies can be easily modified as well as new scheduling and policy modules can be integrated with the Broker Module.

The Job Manager Module is responsible for monitoring of job status changes within the GRMS and then storing information in a database together with many additional parameters (including resource requirements of jobs, user names, job Ids, submission times, pending times, execution times, jobmanagers to which jobs were submitted, history of migration if



jobs have been migrated, etc). Due to an importance of historic information, especially in multi site or large scale resource management systems, the GRMS provides the interface for users and administrator to receive information about past GRMS actives. The access to historic information can be considered as a feedback to the Broker Module and allows developers to implement new dynamic scheduling approaches and policies within the GRMS, e.g. prediction based schedulers or fair-sharing. The tracking of historical resource utilization for all users results in the ability to modify job priorities, ensuring a balance appropriate access, and optimizing administrator criteria (e.g. job throughput or turnaround time).

The Resource Discovery Module monitors a status of distributed resources and therefore uses a flexible hierarchical access to both central and local information services. This module uses various techniques to discover and get an efficient access to up-to-date and accurate (both static and dynamic) information about jobs and resources. The goal of the Resource Discovery Module it to deliver all information in a form and in time required by the Broker Module and its scheduling and policy strategies. Also to this module new extending techniques can be applied, (e.g. indexing or caching) to speed up a flow of information.

### **1.3.1 GRMS Service and Core Services**

The GRMS has been designed as a core resource management system independently on technologies and underlying Core Services. However, from the implementation point of view, we had to choose some uniform and scalable mechanisms for naming, locating and allocation computational and communication resources in distributed environment (see Fig. 1). Therefore, the GRMS version 1.9 is using a set of relatively stable Core Services taken from the Globus 3.2-preWS [4]. Using the Java CoG Kit version 1.2 [5] and APIs to these Core Services, the GRMS is able to perform remote job submission actions through the GRAM Service and staging/transfer files through GridFTP/GASS/FTP Services. As it was presented in the previous chapter, the GRMS uses a hierarchical access to GRIS/GIIS information services and thus receives static and dynamic information about jobs and resources. Note, that all Core Services taken from the Globus are essential for the GRMS and have to be deployed and setup appropriately before you start using the GRMS.

It is worth to mention here that within the GridLab project we have been developing also some useful Core Services, e.g. Mercury monitoring system, which basically extend the functionality of GRIS/GIIS services by adding more dynamic information about resources. The last but not least core service is a database to which the GRMS connects using the JDBC bridge.

### **1.3.2 GRMS Service and other GridLab Middleware Services**

Middleware developers want to have control over all aspects of the grid environment. This is especially true for resource and data management, security issues, interfaces to resources and networks, communications, and so on. Thus, within the GridLab project we have been developing useful middleware services for replica management, data and file transfer, etc. in



addition to the GRMS. Each of these middleware services is also using various Core Services and thus abstract the complexity of underlying technologies and solutions. The current release of the GRMS allow users to integrate it with at least three GridLab Middleware Services: Replica Management, File Movement and Adaptive Components (). Consequently, all these services working together provide a consistent, adaptive and robust grid middleware layer which covers and fits dynamically to many different distributing computing infrastructures.

Note, that all the GridLab Middleware Services are integrated together by using Grid Security Infrastructure and Web Service APIs. Therefore, each middleware service provides a specific functionalities which can be exploited by other services (see the chapter 1.5 and the GridLab Migration Scenario).

### **1.3.2.1 Integration with Replica Management and File Movement Services**

In order to perform job management remotely the GRMS has to stage-in and stage-out files (input files, output files, stdin, stdout, stderr) required by jobs and users before and after executions in a very efficient way. To deal with these issues the GRMS is able to use Core Services taken from Globus GridFTP/GASS/FTP and also use data management GridLab middleware services, namely Replica Management and File Movement [6]. The idea of collaboration with these two middleware services is to use more abstract operations on data and logical files rather than deals with physical file locations and low level data operations. The whole process of data management, mapping between logical and physical files, virtual collections and directories as well as fault tolerant data movement and file transfer are performed internally in these two services. The current release of the GRMS uses two basic functions invoking their Web Service interfaces, file and data movement for staging process and conversion between logical collection files and their physical locations. To start using these middleware services together with the GRMS we assume that they are appropriately deployed in a computing environment (see [6] for more technical details).

### **1.3.2.2 Integration with Adaptive Services**

In addition to basic information services taken from the Globus 3.2-preWS within the GridLab project we have been developing a set of adaptive components which collect and analyze dynamic information about distributed resources and network statuses. The GRMS as a client to the Adaptive Service is able to read results of some basic performance-prediction models concerning network bandwidth, capacity and latency among distributing resources. Currently, we are working on some scenarios which allow the GRMS to use analyses of the availability of CPU/memory of resources as well as various queuing system conditions. All these information taken by the GRMS from Adaptive Service improves basically the quality of knowledge about the whole infrastructure which is managed by the GRMS (see [7] for more technical details concerning GridLab Adaptive Services).

#### 1.4. Technical Details of Release.

The release of GRMS consists of two main components:

- GRMS Service: GSI enabled web service that provides interface for GRMS clients
- Broker Service: GRMS engine provides functionality of GRMS

The motivation for introducing that two elements is to separate access layer technology from logic layer in development process. It gives more flexibility and eases testing procedures. The only disadvantage of that solution is little overhead for RMI communication.

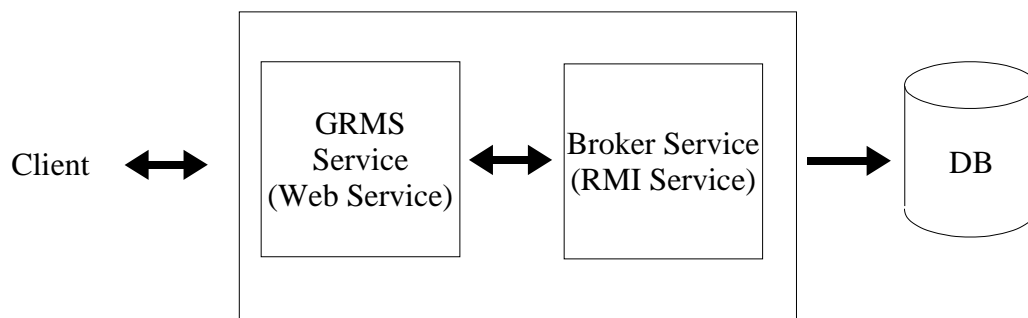


Fig. 2 Release technical details



## 2. GRMS Web Service Interface

### 2.1. Overview

GRMS Service is a web service written in java and running on a java based hosting environment. The hosting environment consists of Tomcat, which is the servlet container and the Apache Axis, which is a SOAP ("Simple Object Access Protocol") engine. The hosting environment allows access to the service's interface via the standard Web Services Definition Language (WSDL). The communication between the service and clients is done by GSI-enabled HTTP-based protocol called "httpg" implementing transport-level security introduced by Globus Community. The protocol is essentially equivalent to the "https" one, but additionally it supports delegation of credentials. To provide the transport-level security the part of GT3-core was used.

### 2.2. Requirements

- Globus Toolkit 3.0.2 Core or 3.2.1 from:  
<http://www-unix.globus.org/toolkit/download.html>
- Tomcat (the recommended version is 4.1.30).from:  
<http://www.apache.org/dist/jakarta/tomcat-4/>
- JDK 1.4.2 from: <http://java.sun.com/j2se/1.4.2>

There is a known bug connected with OGSA stuff (used in GRMS to provide Transport Level Security for SOAP communication) and new versions of Java starting from SUN JDK 1.4.2\_05. It is caused by the fact that these new versions of JRE have bundled a newer version of Xalan, which made an originally public variable private, and the Apache XML Security library used by OGSA is incorrectly accessing it. This cause that service code breaks with following exception

```
java.lang.IllegalAccessException: org.apache.xml.security.Init tried to
access field org/apache/xpath/compiler/FunctionTable.m_functions from
class at org.apache.xml.security.Init.init(Unknown Source)
```

A workarounds are either to use SUN JDK 1.4.2\_04, IBM JDK 1.4.1 or put xalan.jar from ogsa or taken from <http://xml.apache.org/xalan-j/> to \$JAVA\_HOME/jre/lib/endorsed/ and <tomcat-root>/common/endorsed

For more details please see:

- [http://issues.apache.org/bugzilla/show\\_bug.cgi?id=30764](http://issues.apache.org/bugzilla/show_bug.cgi?id=30764)
  - [http://www-unix.globus.org/mail\\_archive/discuss/2004/09/msg00017.html](http://www-unix.globus.org/mail_archive/discuss/2004/09/msg00017.html)
- MyProxy server from:<http://www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/>

AXIS, GSI-plugin, CoG and SSL are included in GT3, so there is no need to download them separately.



## 2.3. Installation of required components

### 2.3.1. Tomcat installation and configuration

- 1) Install the tomcat
- 2) Deploy axis and GSI-plugin performing following set of commands:

```
mkdir -p <tomcat-root>/webapps/axis/WEB-INF/lib  
mkdir <tomcat-root>/webapps/axis/WEB-INF/classes
```

a) for ogsa 3.0.2 please do

```
cp <ogsa-3.0.2>/tomcat/common/lib/* <tomcat-root>/common/lib/  
cp <ogsa-3.0.2>/tomcat/server/lib/cog-tomcat.jar <tomcat-  
root>/server/lib/  
cp <ogsa-3.0.2>/lib/ogsa.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.0.2>/lib/cog-axis.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.0.2>/lib/axis.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.0.2>/lib/wsd14j.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.0.2>/lib/xmlsec.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.0.2>/webapps/ogsa/WEB-INF/web.xml <tomcat-  
root>/webapps/axis/WEB-INF/
```

b) for ogsa 3.2.1 please do:

```
cp <ogsa-3.2.1>/lib/ogsa.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.2.1>/lib/cog-axis.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.2.1>/lib/axis.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.2.1>/lib/wsd14j.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.2.1>/lib/xmlsec.jar <tomcat-root>/webapps/axis/WEB-INF/lib/  
cp <ogsa-3.2.1>/lib/cog-jglobus.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/puretls.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/cryptix32.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/cryptix-asn1.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/cryptix.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/jce-jdk13-120.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/jgss.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/log4j-1.2.8.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/commons-logging.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/commons-discovery.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/jaxrpc.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/saaj.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/jaxp.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/xmlParserAPIs.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/xalan.jar <tomcat-root>/common/lib  
cp <ogsa-3.2.1>/lib/cog-tomcat.jar <tomcat-root>/server/lib  
cp <ogsa-3.2.1>/lib/xalan.jar <tomcat-root>/common/endorsed  
cp <ogsa-3.2.1>/webapps/ogsa/WEB-INF/web.xml <tomcat-  
root>/webapps/axis/WEB-INF/
```

- 3) Configure the Transport Level Security editing <tomcat\_root>/conf/server.xml file:  
Add GSI Connector in <Service name="Tomcat-Standalone"> section and update the parameters appropriately with your local configuration:

```
<!-- Define a GSI HTTP/1.1 Connector on port 8443  
Supported parameters include:  
proxy // proxy file for server to use
```



## "GRMS v.1.9.6 - ADMINISTRATOR GUIDE"

```
or
cert      // server certificate file in PEM format
key       // unencrypted server key file in PEM format
cacertdir // directory location containing trusted CA certs
gridMap   // grid map file used for authorization of users
debug     // "0" is off and "1" and greater for more info
-->
<Connector className="org.apache.catalina.connector.http.HttpConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="true" authenticate="true"
  acceptCount="10" debug="1" scheme="https" secure="true">
  <Factory
    className="org.globus.tomcat.catalina.net.GSIServerSocketFactory"
    proxy="d:\certs\x509up_u945"
    cert="d:\certs\hostcert.pem"
    key="d:\certs\hostkey.pem"
    cacertdir="d:\certs\certs"
    gridMap="d:\certs\gridmap"
    debug="1"/>
  </Connector>
```

For details see the comment above the connector definition.

Add GSI Valve in <Engine name="Standalone" ... > section:

```
<Valve className="org.globus.tomcat.catalina.valves.CertificatesValve" debug="1" />
```

Modify <tomcat\_root>/bin/catalina.bat (on Windows) or

<tomcat\_root>/bin/catalina.sh (on Unix/Linux) adding needed jars to the CLASSPATH

```
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME"/bin/bootstrap.jar
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/cog-jglobus.jar"
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/log4j-core.jar"
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/endorsed/xercesImpl.jar"
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/endorsed/xmlParserAPIs.jar"
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/puretls.jar"
```

```
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/jce-jdk13-117.jar"
```

or

```
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/jce-jdk13-120.jar"
```

depending on version of ogsa package

```
CLASSPATH=" %CLASSPATH" : "%CATALINA_HOME/common/lib/cryptix32.jar"
```



```
CLASSPATH= "$CLASSPATH" : "$CATALINA_HOME/common/lib/cryptix-asn1.jar"  
CLASSPATH= "$CLASSPATH" : "$CATALINA_HOME/common/lib/cryptix.jar"  
CLASSPATH= "$CLASSPATH" : "$CATALINA_HOME/webapps/axis/WEB-INF/lib/grms.jar"
```

Optionally, in order to use the Mercury monitoring system, the following lines must be added:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH":<mercury-root>/lib  
CLASSPATH= "$CLASSPATH" : "$CATALINA_HOME/webapps/axis/WEB-INF/lib/mercury-consumer-2.3.1.jar"  
CLASSPATH= "$CLASSPATH" : "$CATALINA_HOME/webapps/axis/WEB-INF/lib/mercury-producer-2.3.1.jar"
```

### 2.3.2. Instalng and configuring MyProxy Server

Install and run MyProxy server following instruction provided on web page:  
<http://www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/>

## 2.4. Deploying GRMS Service

- 1) Copy 'grms.jar', 'grms\_interface.jar' and 'gl\_services.jar' files from **\$GRMS\_LOCATION/lib** to <tomcat-root>/webapps/axis/WEB-INF/lib/.

To build all needed jars please invoke following commands in **\$GRMS\_LOCATION** directory:

```
ant  
ant -f build.interface.xml jar
```

Copy 'castor.jar' from **\$GRMS\_LOCATION/jars/castor** to <tomcat-root>/webapps/axis/WEB-INF/lib/.

If you also want to use MercuryLogging please copy to the same location files 'mercury-consumer-2.3.1.jar' and '[mercury-producer-2.3.1.jar](#)' from **\$GRMS\_LOCATION/jars/gridlab**

- 2) Copy the file server-config.wsdd describing configuration of hosting environment from **\$GRMS\_LOCATION/etc** to <tomcat-root>/webapps/axis/WEB-INF

```
</globalConfiguration>  
  <!-- GRMS GLOBAL PARAMETERS -->  
    <parameter name="GRMS_RMI_TIMEOUT" value="300000"/>  
    <parameter name="GRMS_FORCE_GC" value="false"/>  
  <!-- GRMS GLOBAL PARAMETERS (END OF SECTION)-->  
</globalConfiguration>
```



```
<service name="grms" provider="java:RPC" style="rpc" use="encoded">
  <parameter name="wsdlTargetNamespace" value="urn:grms"/>
  <parameter name="wsdlServiceElement" value="grmsService"/>
  <parameter name="wsdlServicePort" value="grms"/>
  <parameter name="className" value="grms.gl_services.grms.stub.GrmsSoapBindingSkeleton"/>
  <parameter name="wsdlPortType" value="grms"/>
  <parameter name="allowedMethods" value="*/>

  <parameter name="myproxy.host" value="MYPROXY_HOST"/>
  <parameter name="myproxy.port" value="MYPROXY_PORT"/>
  <parameter name="myproxy.lifetime" value="669600"/>
  <parameter name="myproxy.delegate" value="true"/>
  <parameter name="myproxy.passwd" value="PASSWORD"/>

  <parameter name="service.GAS.location.find" value="false"/>
  <parameter name="service.GAS.name" value="GAS"/>
  <parameter name="service.GAS.location" value="GAS_LOCATION"/>
  <parameter name="service.GAS.authorize" value="true"/>

  <parameter name="service.igrid.location" value="IGRID_LOCATION"/>

  <parameter name="rmiregistry.keep.interfaces" value="true"/>

  <parameter name="rmiregistry.host.0" value="BROKER_HOST"/>
  <parameter name="rmiregistry.port.0" value="BROKER_PORT"/>
  <parameter name="rmiregistry.name.0" value="BROKER_NAME"/>
  <parameter name="rmiregistry.timeout.0" value="TIMEOUT"/>

  <parameter name="service.get_interfaces" value="true"/>
  <parameter name="service.version" value="GRMS Service version 1.9.6"/>
  <parameter name="service.debug" value="true"/>

  <parameter name="service.anonymous_calls" value="false"/>
  <parameter name="anonymous.myproxy.user" value="anonymous"/>

  <parameter name="service.cert" value="../certificate/grmscert.pem"/>
  <parameter name="service.key" value="../certificate/grmskey.pem"/>
  <parameter name="service.proxy.update_time" value="3600"/>

  <parameter name="logger.conf.path" value="/grms/grms.log4j"/>
  <parameter name="logger.conf.name" value="grms.webservice.GRMSService"/>
  <parameter name="logger.conf.history.enable" value="true"/>
  <parameter name="logger.conf.history.dir" value="/grms/history"/>
  <parameter name="logger.conf.history.appender" value="HISTORY"/>

  <parameter name="mercury.logger.enable" value="true"/>
  .
  .
  .
</service>
```

Global configuration of all instances of GRMs service can be set in <globalConfiguration> section. Following parameters can be set:

- **GRMS\_RMI\_TIMEOUT** – value in miliseconds defining timeout for rmi connections between webservice interface and broker-engine,
- **GRMS\_FORCE\_GC** – boolean value defining if each request at the beginning will force invocation of “garbage collector”.

The service must be configured using following properties:

- **myproxy.host** – host of myproxy-server,
- **myproxy.port** – port of myproxy-server
- **myproxy.lifetime** – lifetime of delegated to myproxy-server proxy
- **myproxy.delegate** – determines if the service should delegate user proxy to myproxy-service. Useful for testing.
- **myproxy.passwd** – password to user proxy on myproxy server. Myproxy user is equal to user's DN. This properties must be equal to corresponding one in /etc/config.prop



- **service.GAS.location.find** – if this property is set to “true”. GRMS tries to find GAS location using iGrid. It assumes that GAS is registered under [service.GAS.name](#) name and the iGrid location is [service.igrid.location](#). Default value of [service.GAS.location.find](#) is false.
- **service.GAS.name** – name of the GAS in iGrid service.
- **service.GAS.location** – address of authorization service, if this property is empty the information service will be used to determine the location of the GAS,
- **service.GAS.authorize** – determine if the user request will be authorized by GAS,
  
- **service.igrid.location** – address of information service (iGrid),
  
- **rmiregistry.host.0** – host of the rmiregistry where the broker interface is registered– must be the same host where the grms engine was started
- **rmiregistry.port.0** – port of the rmiregistry, must correspond to the “common.rmi.port” property in \$GRMS\_LOCATION/etc/config.prop,
- **rmiregistry.name.0** – name of the broker interface in remiregistry, must correspond to the “common.factory.interface” property in \$GRMS\_LOCATION/etc/config.prop,
- **rmiregistry.timeout.0** – timeout for rmi connections.  
Grms (Web Service) is able to dispatch requests to many brokers using “round Robin” algorithm to define please add next rmi interface properties incrementing the number at the end of properties names (for example rmiregistry.host.1)
  
- **rmiregistry.keep.interfaces** – defines if RMI broker interfaces will be taken from rmi registry for every request or will be “remembered” on service side. Default value is “true”.
  
- **service.get\_interfaces** – determine if the service will try to get broker interfaces. Useful for testing.
- **service.version** – version string displayed by getServiceDescription method,
- **service.debug** – in case of unexpected problem determines if the debug information should be added to the standard response,
  
- **service.anonymous\_calls** – defines if requests using pure http protocol instead of httpg one are acceptable. Default value is “false”.
- **anonymous.myproxy.user** – defines the myproxy-server user name which will be used for anonymous calls
  
- **service.cert** – path to the file containing service certificate,
- **service.key** – path to the file containing service key,
- **service.proxy** – path to the file containing service proxy. If the **service.proxy** property is set **service.cert** and **service.key** properties are not taken into consideration.
- **service.proxy.update\_time** – the proxy used by service is refreshed when its remaining life time in seconds is smaller than value of this property.



- **logger.conf.path** – location of file describing configuration of logging system (log4)
- **logger.conf.name** – name of the logger dedicated for the instance of service
- **logger.conf.history.enable** – this property determines whether the service should store all requests concerning the same job in a separate file in “logger.conf.history.dir” directory (related to the axis/WEB-INF path)
- **logger.conf.history.dir** – directory where the history of job requests will be stored
- **logger.conf.history.appender** – name of the appender which will be used for logging historical information. The appender has to be of “FileAppender” type.
  
- **mercury.logger.enable** – defines if logging information should be also logged in MercuryLogging service. This functionality requires installed mercury service.

3) Add the contents of the **\$GRMS\_LOCATION/etc/rmi.policy** file at the end of `<tomcat_root>/conf/catalina.policy` file.

## ***2.5. Starting the GRMS Service***

- 1) Start the tomcat executing `<tomcat_root>/bin/startup.sh` or `startup.bat` with **-security** option.



## 3. Broker Service.

### 3.1. Overview.

Broker Service is an engine of GRMS. It is implemented in Java, and can be accessed through RMI interface.

### 3.2. Requirements.

- Java COG Kit
- Postgres (the recommended version 7.3.2) from:  
[www.postgres.org](http://www.postgres.org)
- JDK 1.4.2 from <http://java.sun.com/j2se/1.4.2>
- MyProxy server from <http://www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/>

### 3.3. Instalation of required components.

#### 3.3.1. PostgreSQL installation and configuration.

- 1) Install the postgresql,
- 2) Configure postgres to allow TCP connections,  
Please read a postgres documentation to find the most suitable for you authorization method. The simplest, but not recommended because of security aspects, method is presented below:

uncomment out or add following lines in /var/lib/pgsql/data/pg\_hba.conf file

```
# TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
local all all 127.0.0.1 255.255.255.255 trust
host all all 127.0.0.1 255.255.255.255 trust
```

- 3) Modify /etc/init.d/postgresql adding "-i" to the options of pg\_ctl

```
su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D $PGDATA -p /usr/bin/postmaster -o '-i -p ${PGPORT}' start > /dev/null 2>&1" < /dev/null
```

- 4) As a postgres administrator create user and database needed for GRMS

```
# dropuser grms-devel
# createuser grms-devel

# dropdb grms-devel_jobreg
# createdb -O grms-devel -W grms-devel_jobreg
```



```
# dropdb grms-devel_queue
# createdb -O grms-devel -W grms-devel_queue

5) Login to the postgres and create tables needed for GRMS
# psql -d grms-devel_jobreg -U grms-devel -W
# \i <grms_root>/etc/jobreg_create.sql

# psql -d grms-devel_queue -U grms-devel -W
# \i <grms_root>/etc/queue_create.sql

6) Please increase postmaster's limit on how many concurrent backend processes it can start.
http://www.postgresql.org/files/documentation/faqs/FAQ.html#3.5
http://jamesthornton.com/postgres/FAQ/faq-english.html#3.8
```

## 3.4. Configuring Broker Service

### 3.4.1. Instalation

- 1) Download software from GridLab WP9 web page: <http://www.gridlab.org>
- 2) After unpacking there is following directory structure in \$GRMS\_LOCATION:
  - bin/ - scripts
  - etc/ - directory containing configuration files
  - src/ - sources directory
  - classes/ - classes directory
  - lib/ - Broker Service jar files
  - jars/ - required external software jars directory
  - doc/ - javadoc directory
- 3) Run 'ant allgrmsengine' to build Broker Service.

### 3.4.2. Configuration

Configuration files located in \$GRMS\_LOCATION/etc directory:

- create.sql: script that can be used for creating database tables for job registering
- jobreg\_database.xml, jobreg\_mapping.xml: Job Registry database acces configuration files
- queue\_database.xml, queue\_mapping.xml: Job Queue database configuration files.
- grms\_schema.xsd: job describtion schema
- resdisc.conf: Resource Discovery module configuration – contains a list of records describing available grid information services which should be quieried in order to provide the most up to date information required for the job scheduling
- rmi.policy: RMI access configuration
- log4j.properties – properties for log4j logging
- errors.prop: error description strings



- sates.prop: States and Request States strings
- config.prop: Broker Service configuration properties

Configuration of Broker Service itself is mainly located in 'config.prop' file. It consists of grouped properties which are loaded at service startup:

### Common Settings:

**common.myproxy.host**

Name of host with MyProxy Server running

**common.myproxy.port**

Default '7512'. Port of MyProxy Server.

**common.myproxy.lifetime**

Lifetime of credentials put/get to MyProxy Server

**common.servicecert**

Path to certificate file for GRMS

**common.servicekey**

Path to key file for GRMS

**common.portRange**

Port range used by the COG kit

**common.gridftpPort**

Default '2811'. GridFTP port used to contact GridFTP server.

**common.gassPort**

Default port of GAAS service

**common.globus.debuglevel**

Debug level for COG kit

**common.gridlab.replicaLocation**

Location (url) for Replica Catalog System – developed in GridLab project by WP8

**common.gridlab.movementLocation**

Location (url) for File Transfer System - developed in GridLab project by WP8

**common.gridlab.gasLocation**

Location (url) of Grid Authorization Service – developed in GridLab project by WP6

**common.gridlab.adaptiveLocation**

Location (url) of Adaptive Components Service – developed in GridLab project by WP7



**common.gridlab.testbedLocation**

Location (url) of Testbed Service used in GridLab instead of GIIS server.

**common.rmi.codebase**

Location of Broker Service RMI stub

**common.rmi.policy**

Location of 'rmi.policy' file

**common.factory.interface**

Name of Broker Service interface bound to RMI registry

**common.rmi.port**

RMI registry port

**Broker Module:**

**broker.enable.gasFileAuthorization**

Default 'no'. Set 'yes' to use GAS Service to authorize some file transfer operations.

Remarks: prototype solution

**broker.enable.notification**

Default 'no'. Set 'yes' to send notifications after job status changes

**broker.enable.fileTransferService**

Default 'no'. Set 'yes' to use Data Transfer Service (Gridlab WP8) for transferring files.

**broker.envConfigFile**

Name of file with machine specific environment variables which is downloaded from each machine and used to set environment variables for applications, and for some job Description variables replacement.

**broker.envConfigValid**

Time to live of cached environment variables files (in ms)

**broker.setGridlabConfEnv**

Default 'no'. Set 'yes' to set environment variables from configuration file stored on machine.

**broker.replaceLocations**

Default 'no'. Set 'yes' to replace Job Description variables with information from environment files.

**broker.useConfFile**



Default 'no'. Set 'yes' to use information from environment file to replace variables in Job Description.

**broker.appCheckpoint**

Default 'yes'. Denotes if applications are checkpointable (implement known checkpoint interface). Setting 'no' means that after migration call application should be canceled, and then migrated (using periodically generated checkpoint files).

Remarks:Property will be removed soon – information will go to Job Description.

**broker.jobsdire**

Path to temporal Broker directory

**broker.removeWorkingDir**

Default 'yes'. Set 'no' if you do not want to remove working directory after job is finished

**broker.copyExecFile**

Default 'no'. Set 'yes' if to copy executable file described as local file to working directory of job execution.

**broker.castor.debug**

Sets (true) debug mode for Castor classes

**broker.preferences.weightLoad**

Weight assign to machine load by scheduling algorithm

**broker.preferences.weightMemory**

Weight assign to machine memory by scheduling algorithm

**broker.preferences.weightCPUSpeed**

Weight assign to machine cpu performance by scheduling algorithm

**broker.preferences.weightCPUCount**

Weight assign to machine number of cpus by scheduling algorithm

**broker.keepJobInQueue**

Default 'no'.

**broker.advancedScenario**

Default 'no'

Queue

**queue.mapping\_file**

Location fo database mapping file for Job Queue.



**queue.database\_file**

Location of file database configuration file for Job Queue

**queue.database\_name**

Name of Queue database.

**queue.initclear**

Default 'yes'. Clears Queue at service startup.

**Other properties:**

**broker.modulename**

Name of Broker Module used by log4j (default Broker)

**resdisc.modulename**

Name of Resource Discovery Module used by log4j (default ResDisc)

**jobmon.modulename**

Name of Job Manager Module used by log4j (default JobMon)

**server.modulename**

Name of Server Module used by log4j (default Server)

**jobRepository.timer.period**

The time period in milliseconds between checking for unused information, which can be removed from cache (default: 30000)

**jobRepository.timer.timeout**

Time in milliseconds after which the unused information is removed from cache repository (default: 100000)

**jobRepository.mapping\_file**

Location of database mapping file for Job Registry.

**jobRepository.database\_file**

Location of file database configuration file for Job Registry.

**jobRepository.database\_name**

Name of Job Registry database.

**Resource Discovery Module configuration file:**

The Resource Discovery module is configured by means of special configuration file. The module tries to find that file in location pointed by the value of 'resdisc.configfile' property



defined in file `./etc/config.prop`. If that property is not defined then the module tries to find its configuration in the file `./etc/resdisc.conf`. The configuration file consists of sections, which describe available information service. Each section describes one information service. Every information service should be defined by record of the following format:

```
[backup] INDEX serviceName
{
    TYPE=[iGrid|PureMDS|CachedMDS]
    [disable=[yes|no]]
    configurationProperty1 = value1
    configurationProperty2 = value2
    ...
    configurationPropertyN = valueN
}
```

Any line starting with '#' sign is treated as a comment and ignored. White spaces around values are truncated.

If the optional keyword 'backup' is used, the service will be considered a primary service. Otherwise the service will be considered backup service and it will be used only if querying other primary services failed.

The service Name may be arbitrary chosen and has no significant meaning for the way the module operates. It is mainly used by the logging facility.

Every INDEX record contains a list of its specific properties. Every INDEX record may contain 'disable' property. If its value is set to 'yes' the record will be ignored.

The property "TYPE" has a special meaning and every INDEX record must define it. The value of that property defines the type of information service and thus, the type of information provider, which should be to query that service. The type determines all the other properties, which must (or may) be defined in the record. Below is the list of available types with description of its specific properties:

1. PureMDS – defines standard Globus MDS information service. For that type a simple LDAP querying will be performed with only some basic caching facility. The following attributes are valid for that type:
  - a) HOST – is host name on which Globus GIIS is running
  - b) PORT – the Globus GIIS port
  - c) BASEDN – the DN of GIIS
  - d) LDAPTimeout – defines timeout for LDAP single search operation performed in GIIS server. Value of "0" means that there is no timeout and search operations will wait indefinitely. The value is a string representation of a decimal number of seconds.



- e) `LDAPBatchSize` – defines the batch size property for LDAP context for GIIS query operation. Changing value of this property affects the performance of the module. The value is a string representation of a decimal integer.
- f) `ohqLDAPTimeout`, `ohqLDAPBatchSize` – same as `LDAPTimeout` and `LDAPBatchSize` but these two are applied when query is performed for a single resource specified by user in host description and for queries performed in GRIS servers.
- g) `localCacheDir` – this property contains a path pointing to a directory where Resource Discovery module shall save its cache files.
- h) `cacheTTL` – cache time to live in seconds.

Example:

```
INDEX localMDS
{
    TYPE=PureMDS
    HOST=localhost
    PORT=2135
    BASEDN="mds-vo-name=local, o=grid"
    disable = no
    LDAPTimeout = 140
    LDAPBatchSize = 0
    ohqLDAPTimeout = 102
    ohqLDAPBatchSize = 53
    ignoreCacheIfLessThan = 1
    cacheTTL=3600
}
```

- 2. `CachedMDS` – defines an information provider which gets host specific information from standard Globus GRIS servers, but instead the Globus GIIS as a indexing service, the Gridlab TestBed webservice or local file is used. Moreover information provider of that type will use some more advanced caching methods. The following attributes are valid for that type:
  - a) `PrimaryIndex = [WS|file]` This option specifies whether the list of resources available in grid environment should be retrieved from Gridlab TestBed webservice (WS) or read from local file (file). The local file path is specified by `HostListFile` property.
  - b) `SecondaryIndex = [WS|file]` This option specifies what index should be used if getting resource list from primary index failed. In the



- current version this option is used only when it is set to file and PrimaryIndex is set to WS.
- c) HostListFile = [index file path] This option defines the localization of file containing the list of hostnames of available resources. Every line in that file should contain one host name.
  - d) UseAdditiveCache = [yes|no] If this option is enabled then in case user requests for host which has not been stored in cache so far then the GRIS on that host will be queried directly and if valid results are returned then the host will be added to cache.
  - e) UseTransitiveCache = [yes|no] If this option is set to yes then the hosts for which information has been cached in past but cannot be obtained during the latest cache rebuild process, will be kept in new cache, but no more than MaxCacheAge times.
  - f) MaxCacheAge = [integr value]
  - g) localCacheDir as described above
  - h) cacheTTL as described above

Example:

```
INDEX gridlab.org
{
    TYPE=CachedMDS
    PrimaryIndex=WS
    SecondaryIndex=file
    IndexWS=httpg://loni.ics.muni.cz:30443/axis/services/TestbedStatus
    disable          = no
    useHostsList     = yes
    HostsListFile    = ./etc/hosts.conf
    useTestbedWS     = yes
    useAdditiveCache = yes
    useTransitiveCache = yes
    maxCacheAge      = 3
    MPIonly=false
    cacheTTL         = 7200
}
```

3. iGrid – the GridLab iGrid information provider. The following attributes are valid for that type:
  - a) iStore defines the address of the iStore service
  - b) useIStoreFile=[yes|no] defines whether the iStoreFile shall be used instead of direct querying the iServe service.



- c) `iStoreFile` defines the location of file containing saved `iStore` results
- d) `DefaultJMC`, `defaultJMctype` defines the jobmanager contact address and type which shall be faked in case of a resource doesn't specify any jobmanager

Example:

```
INDEX GLabIStore
{
    TYPE=iGrid
    iStore=httpg://mds.gridlab.org:19000
    disable = no
    useIStoreFile = no
    iStoreFile = ./var/igrid/gridlab_11jan2005.xml
    defaultJMC = "jobmanager"
    defaultJMctype = "fork"
}
```

In the configuration file may be defined one special section. It is called "GLOBAL" section and has the following format:

```
GLOBAL
{
    configurationProperty1 = value1
    configurationProperty2 = value2
    ...
    configurationPropertyN = valueN
}
```

All properties defined within GLOBAL section will be applied to all INDEX sections, but GLOBAL properties will not overwrite local properties which have been already defined in INDEX sections. In other words, the GLOBAL section defines the defaults for given properties.



### **3.5. Starting Broker Service**

1) Run Broker Service using starting script  
# cd \$GRMS\_LOCATION  
# ./bin/grms.sh&

### 3.6. Stopping Broker Service

1) Stop Broker Service using script:  
# cd \$GRMS\_LOCATION  
# ./bin/stop\_grms.sh&



## 4. Command-Line Client.

### 4.1. Overview.

Simple command-line client is provided with GRMS release. It can be used for testing purposes. For more details concerning usage of Grms and functionality of command line client please see: Grms User Guide.

### 4.2. Requirements.

- JDK 1.4.2 from <http://java.sun.com/j2se/1.4.2>

There is a known bug connected with OGSA stuff (used in GRMS to provide Transport Level Security for SOAP communication) and new versions of Java starting from SUN JDK 1.4.2\_05. It is caused by the fact that these new versions of JRE have bundled a newer version of Xalan, which made an originally public variable private, and the Apache XML Security library used by OGSA is incorrectly accessing it. This cause that service code breaks with following exception

```
java.lang.IllegalAccessException: org.apache.xml.security.Init tried to
access field org/apache/xpath/compiler/FunctionTable.m_functions from
class at org.apache.xml.security.Init.init(Unknown Source)
```

A workarounds are either to use SUN JDK 1.4.2\_04, IBM JDK 1.4.1 or put xalan.jar from ogsa or taken from <http://xml.apache.org/xalan-j/> to \$JAVA\_HOME/jre/lib/endorsed/

For more details please see:

- [http://issues.apache.org/bugzilla/show\\_bug.cgi?id=30764](http://issues.apache.org/bugzilla/show_bug.cgi?id=30764)
  - [http://www-unix.globus.org/mail\\_archive/discuss/2004/09/msg00017.html](http://www-unix.globus.org/mail_archive/discuss/2004/09/msg00017.html)
- Other needed tools and libraries are delivered with release

### 4.3. Installation and configuration of GRMS client

1) Unpack the GRMS v.1.9 release

2) Edit \$GRMS\_LOCATION/bin/ws\_client script

Please modify following lines if it is necessary:

```
GRMS_URL="http://rage1.man.poznan.pl:8543/axis/services/grms"
```

```
GRMS_DN="grms-dn"
```

```
GRMS_TIMEOUT=5
```

```
GRMS_DELEG_TYPE=FULL
```

The GRMS\_URL describes location of GRMS service.

The GRMS\_DN describes the expected by the client distinguish name of the service. This value has to be set to perform delegation of users credential.

Currently Grms uses following certificate:



```
"/C=PL/O=GRID/O=PSNC/CN=grms_devel/ragel.man.poznan.pl"
```

The GRMS\_TIMEOUT property sets the timeout (in minutes) for service response.  
The GRMS\_DELEG\_TYPE – determines the type of proxy delegation.

3) Create or modify cog.properties file in ~/.globus directory, change values of properties according to your configuration

```
#Java CoG Kit Configuration File
#Wed Jul 17 09:25:01 CEST 2002
usercert=../../usercert.pem
userkey=../../userkey.pem
proxy=/tmp/x509up_u501
cacert=/etc/grid-security/certificates/
```

For details please see:  
<http://www-unix.globus.org/cog/distribution/1.1/FAQ.TXT>

#### 4.4. Executing GRMS Client

- 1) run <globus\_location>/bin/grid-proxy-init command to create user proxy
- 2) run the GRMS client in \$GRMS\_LOCATION/bin directory typing:

```
./ws_client.sh <operation> <parameters>
# GRMS USAGE:
# ./ws_client.sh submit <jobDescriptionFile>
# ./ws_client.sh migrate <jobId> [<jobDescriptionFile>]
# ./ws_client.sh suspend <jobId> [<jobDescriptionFile>]
# ./ws_client.sh resume <jobId> [<jobDescriptionFile>]
# ./ws_client.sh cancel <jobId>
# ./ws_client.sh list [QUEUED | PREPROCESSING | PENDING | RUNNING |
STOPPED | POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED]
# ./ws_client.sh project [QUEUED | PREPROCESSING | PENDING | RUNNING |
STOPPED | POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED]
# ./ws_client.sh list_all QUEUED | PREPROCESSING | PENDING | RUNNING |
STOPPED | POSTPROCESSING | FINISHED | SUSPENDED | FAILED | CANCELED
# ./ws_client.sh register <jobId> <service_location> <pid>
# ./ws_client.sh unregister <jobid>
# ./ws_client.sh access <jobId>
# ./ws_client.sh info <jobId>
# ./ws_client.sh history <jobId>
# ./ws_client.sh resources <resourceDescriptionFile>
# ./ws_client.sh test <resourceDescriptionFile>
# ./ws_client.sh add_notif <jobid> STATUS|REQUEST SOAP/GASS <destination>
[ true|false [<format>]]
# ./ws_client.sh del_notif <jobid> <notificationId>
# ./ws_client.sh list_notif <jobid>
# ./ws_client.sh info_notif <jobid> <notificationId>
# ./ws_client.sh add_output <jobId> <name> <path> PHYSICAL|LOGICAL FILE|
DIRECTORY
# ./ws_client.sh get_output <jobId>
# ./ws_client.sh del_output <jobId> <name> <path> PHYSICAL|LOGICAL FILE|
DIRECTORY
```



## "GRMS v.1.9.6 - ADMINISTRATOR GUIDE"

```
# ./ws_client.sh add_output <jobId> <name> <path> PHYSICAL|LOGICAL FILE|  
DIRECTORY  
# ./ws_client.sh get_output <jobId>  
# ./ws_client.sh del_output <jobId> <name> <path> PHYSICAL|LOGICAL FILE|  
DIRECTORY  
# ./ws_client.sh description SHORT|FULL
```



## 5. References

- [1] Technical specification document
- [2] GAT Applications
- [3] [www.gridlab.org](http://www.gridlab.org)
- [4] <http://www.globus.org>
- [5] <http://www-unix.globus.org/cog/java/index.php>
- [6] <http://www.gridlab.org/WorkPackages/wp-8/index.html>
- [7] <https://www.gridlab.org/Internal/WorkPackages/wp-7/index.html>
- [8] <https://www.gridlab.org/WorkPackages/wp-4/index.html>