WORLDWIDE LHC COMPUTING GRID

GLITE 3.0 USER GUIDE

MANUALS SERIES

Document identifier:	**CERN-LCG-GDEIS-722398**
EDMS id:	**722398**
Version:	**0.1**
Date:	**May 19, 2006**
Section:	**Experiment Integration and Distributed Analysis**
Document status:	**PRIVATE**
Author(s):	Stephen Burke, Simone Campana, Antonio Delgado Peris, Flavia Donno, Patricia Méndez Lorenzo, Roberto Santinelli, Andrea Sciabà
File:	**gLite-3-UserGuide**

Abstract: *This guide is an introduction to the WLCG/EGEE Grid and to the gLite 3.0 middleware from a user's point of view.*

Document Change Record

Issue	Item	Reason for Change
20/04/06	v1.0	First Draft

Files

Software Products	User files
PDF	https://edms.cern.ch/file/722398/1/gLite-3-UserGuide.pdf
PS	https://edms.cern.ch/file/722398/1/gLite-3-UserGuide.ps
HTML	https://edms.cern.ch/file/722398/1/gLite-3-UserGuide.html

CONTENTS

1. INTRODUCTION

1.1. ACKNOWLEDGMENTS

This work received support from the following institutions:

- Istituto Nazionale di Fisica Nucleare, Roma, Italy.
- Ministerio de Educación y Ciencia, Madrid, Spain.
- Particle Physics and Astronomy Research Council, UK.

1.2. OBJECTIVES OF THIS DOCUMENT

This document gives an overview of the gLite 3.0 middleware. It helps users to understand the building blocks of the Grid and the available interfaces to the Grid services in order to run jobs and manage data.

This document is neither an administration nor a developer guide.

1.3. APPLICATION AREA

This guide is addressed to WLCG/EGEE users and site administrators who would like to work with the gLite middleware.

1.4. DOCUMENT EVOLUTION PROCEDURE

The guide reflects the current status of the gLite middleware, and will be modified as new gLite releases are produced. In some parts of the document, references to the foreseeable future of the gLite software are made.

1.5. REFERENCE AND APPLICABLE DOCUMENTS

REFERENCES

[1] EGEE – Enabling Grids for E-sciencE
http://eu-egee.org/

[2] gLite – Lightweight Middleware for Grid Computing
http://cern.ch/glite/

[3] Worldwide LHC Computing Grid
http://cern.ch/LCG/

[4] The DataGrid Project
http://www.edg.org/

[5] DataTAG – Research & technological development for a Data TransAtlantic Grid
http://cern.ch/datatag/

[6] The Globus Alliance
http://www.globus.org/

[7] GriPhyN – Grid Physics Network
http://www.griphyn.org/

[8] iVDgL – International Virtual Data Grid Laboratory
http://www.ivdgl.org/

[9] Open Science Grid
http://www.opensciencegrid.org/

[10] The Virtual Data Toolkit
http://vdt.cs.wisc.edu/

[11] NorduGrid
http://www.nordugrid.org/

[12] Ian Foster, Carl Kesselman, Steven Tuecke,
The Anatomy of the Grid: Enabling Scalable Virtual Organizations
http://www.globus.org/alliance/publications/papers/anatomy.pdf

[13] M. Dimou,
LCG User Registration and VO Management
https://edms.cern.ch/document/428034/

[14] EGEE CIC Operations Portal
http://cic.in2p3.fr/

[15] Global Grid User Support
http://www.ggus.org/

[16] GOC Database 2.0
https://goc.grid-support.ac.uk/gridsite/gocdb2/

[17] Overview of the Grid Security Infrastructure
http://www-unix.globus.org/security/overview.html

[18] LCG-2 User Guide
https://edms.cern.ch/document/454439/

[19] The Storage Resource Manager
http://sdm.lbl.gov/srm-wg/

[20] MDS 2.2 Features in the Globus Toolkit 2.2 Release
http://www.globus.org/toolkit/mds/#mds_gt2

[21] R-GMA: Relational Grid Monitoring Architecture
http://www.r-gma.org/index.html

[22] B. Tierney *et al.*,
A Grid Monitoring Architecture,
GGF, 2001 (revised 2002)
http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf

[23] S. Campana, M. Litmaath, A. Sciabà,
LCG-2 Middleware overview
https://edms.cern.ch/document/498079/

[24] F. Pacini,
WMS User's Guide
https://edms.cern.ch/document/572489/

[25] WP1 Workload Management Software – Administrator and User Guide
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf

[26] CESNET,
LB Service User's Guide
https://edms.cern.ch/document/571273/

[27] The GLUE schema
http://infnforge.cnaf.infn.it/glueinfomodel/

[28] Using lxplus as an LCG-2 User Interface
http://grid-deployment.web.cern.ch/grid-deployment/documentation/UI-lxplus/

[29] GridICE: a monitoring service for the Grid
http://www.infn.it/gridice/

[30] Condor Classified Advertisements
http://www.cs.wisc.edu/condor/classad

[31] The Condor Project
http://www.cs.wisc.edu/condor/

[32] F. Pacini,
Job Description Language HowTo
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf

[33] F. Pacini,
JDL Attributes
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

[34] F. Pacini,
Job Attributes Specification
https://edms.cern.ch/document/555796/1/

[35] F. Pacini,
Job Description Language (JDL) Attributes Specification
https://edms.cern.ch/document/590869/1/

[36] The EDG-Brokerinfo User Guide
http://www.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2_2.pdf

[37] Workload Management Software – GUI User Guide
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0143-0_0.pdf

[38] GSIFTP Tools for the Data Grid
http://www.globus.org/toolkit/docs/2.4/datagrid/deliverables/gsiftp-tools.html

[39] RFIO: Remote File Input/Output
http://doc.in2p3.fr/doc/public/products/rfio/rfio.html

[40] CASTOR
http://cern.ch/castor/

[41] dCache
http://www.dCache.org

[42] POOL - Persistency Framework. Pool Of persistent Objects for LHC
http://lcgapp.cern.ch/project/persist
Learning POOL by examples, a mini tutorial
http://lcgapp.cern.ch/project/persist/tutorial/learningPoolByExamples.html

[43] The edg-replica-manager wrapper script
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/edg-rm-wrapper.pdf

[44] R. Santinelli, F. Donno,
Experiment Software Installation on LCG-2
https://edms.cern.ch/document/498080/

APPLICABLE DOCUMENTS

[A1] EDG User's Guide
http://marianne.in2p3.fr/datagrid/documentation/EDG-Users-Guide-2.0.pdf

[A2] LCG-1 User Guide
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/LCG-1-UserGuide.htm

[A3] LDAP Services User Guide
http://hepunx.rl.ac.uk/edg/wp3/documentation/wp3-ldap_user_guide.html

[A4] LCG-2 User Scenario
https://edms.cern.ch/file/498081//UserScenario2.pdf

[A5] LCG-2 Frequently Asked Questions
https://edms.cern.ch/document/495216/

[A6] Tank And Spark
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/internal/chep04/SW_Installation.pdf

[A7] How to Manually configure and test the Experiment Software Installation mechanism on LCG-2
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/configuration_of_tankspark

1.6. TERMINOLOGY

1.6.1. Glossary

AFS:	Andrew File System
API:	Application Programming Interface
BDII:	Berkeley Database Information Index
CASTOR	CERN Advanced STORage manager
CE:	Computing Element
CERN:	European Laboratory for Particle Physics
ClassAd:	Classified advertisement (Condor)
CLI:	Command Line Interface
CNAF:	INFN's National Center for Telematics and Informatics
dcap:	dCache Access Protocol
DIT:	Directory Information Tree
DLI:	Data Location Interface
DN:	Distinguished Name
EDG:	European DataGrid
EDT:	European DataTAG
EGEE:	Enabling Grids for E-sciencE
ESM:	Experiment Software Manager
FCR:	Freedom of Choice for Resources

FNAL:	Fermi National Accelerator Laboratory
GFAL:	Grid File Access Library
GGF:	Global Grid Forum
GGUS:	Global Grid User Support
GIIS:	Grid Index Information Server
GLUE:	Grid Laboratory for a Uniform Environment
GMA:	Grid Monitoring Archictecture
GOC:	Grid Operations Centre
GRAM:	Globus Resource Allocation Manager
GRIS:	Grid Resource Information Service
GSI:	Grid Security Infrastructure
gsidcap:	GSI-enabled version of the dCache Access Protocol
gsirfio:	GSI-enabled version of the Remote File Input/Output protocol
GUI:	Graphical User Interface
GUID:	Grid Unique ID
HDM:	Hierarchical Storage Manager
ID:	Identifier
INFN:	Istituto Nazionale di Fisica Nucleare
IS:	Information Service
JDL:	Job Description Language
kdcap:	Kerberos-enabled version of the dCache Access Protocol
LAN:	Local Area Network
LB:	Logging and Bookkeeping Service
LDAP:	Lightweight Directory Access Protocol
LFC:	LCG File Catalog
LFN:	Local File Name
LHC:	Large Hadron Collider
LGC:	LHC Computing Grid
LRC:	Local Replica Catalog
LRMS:	Local Resource Management System
LSF:	Load Sharing Facility
MDS:	Monitoring and Discovery Service
MPI:	Message Passing Interface
MSS:	Mass Storage System
OS:	Operating System
PBS:	Portable Batch System
PFN:	Physical File name
PID:	Process IDentifier
POOL:	Pool of Persistent Objects for LHC
RAL:	Rutherford Appleton Laboratory
RB:	Resource Broker
RFIO:	Remote File Input/Output
R-GMA:	Relational Grid Monitoring Archictecture
RLI:	Replica Location Index

RLS:	Replica Location Service
RM:	Replica Manager
RMC:	Replica Metadata Catalog
RMS:	Replica Management System
ROS:	Replica Optimization Service
SASL:	Simple Authorization & Security Layer (LDAP)
SE:	Storage Element
SFT:	Site Functional Tests
SMP:	Symmetric Multi Processor
SRM:	Storage Resource Manager
SURL:	Storage URL
TURL:	Transport URL
UI:	User Interface
URI:	Uniform Resource Identifier
URL:	Universal Resource Locator
UUID:	Universal Unique ID
VDT:	Virtual Data Toolkit
VO:	Virtual Organization
WLCG:	Worldwide LHC Computing Grid
WMS:	Workload Management System
WN:	Worker Node
WPn:	Work Package #n

2. Executive Summary

This user guide is intended for users of the gLite 3.0 middleware. In these pages, the user will find an introduction to the services provided by the WLCG/EGEE Grid and a description of how to use them. Examples are given of the management of jobs and data, the retrieval of information about resources, and other functionality.

An introduction to the gLite 3.0 middleware is presented in Chapter 3. This chapter describes all the middleware components and provides most of the necessary terminology. It also presents the WLCG and the EGEE projects, which developed the gLite 3.0 middleware.

In Chapter 4, the preliminary procedures to follow before starting to use the Grid are described: how to get a certificate, join a Virtual Organisation and manage proxy certificates.

Details on how to get information about the status of Grid resources are given in Chapter 5, where the different information services and monitoring systems are discussed.

An overview of the Workload Management service is given in Chapter 6. This chapter explains the basic commands for job submission and management, as well as those for retrieving information on running and finished jobs.

Data Management services are described in Chapter 7. Not only the high-level interfaces are described, but also commands that can be useful in case of problems or for debugging purposes.

Finally, the appendices give information about the gLite 3.0 middleware components (Appendix A), the configuration files and enviroment variables for users (Appendix B), the possible states of a job during submission and execution (Appendix C), user tools for the Grid (Appendix D), VO-wide utilities (Appendix E), APIs for data management and file access (Appendix F), and the GLUE Schema used to describe Grid resources (Appendix G).

3. OVERVIEW

The ***EGEE project*** [1] has a main goal of providing researchers with access to a geographically distributed computing Grid infrastructure, available 24 hours a day. It focuses on maintaining the ***gLite*** middleware [2] and on operating a large computing infrastructure for the benefit of a vast and diverse research community.

The ***World wide LHC Computing Grid Project (WLCG)*** [3] was created to prepare the computing infrastructure for the simulation, processing and analysis of the data of the ***Large Hadron Collider (LHC)*** experiments. The LHC, which is being constructed at the European Laboratory for Particle Physics (***CERN***), will be the world's largest and most powerful particle accelerator.

The WLCG and the EGEE projects share a large part of their infrastructure and operate it in conjunction. For this reason, we will refer to it as the ***WLCG/EGEE infrastructure***.

The gLite 3.0 middleware comes from a number of Grid projects, like DataGrid [4], DataTag [5], Globus [6], GriPhyN [7], iVDGL [8], EGEE and LCG. This middleware is currently installed in sites participating in WLCG/EGEE.

In WLCG other Grid infrastructures exist, namely the ***Open Science Grid (OSG)*** [9], which uses the middleware distributed by VDT [10], and NorduGrid [11], which uses the ARC middleware. These are not covered by this guide.

The case of the LHC experiments illustrates well the motivation behind Grid technology. The LHC accelerator will start operation in 2007, and the experiments that will use it (***ALICE, ATLAS, CMS*** and ***LHCb***) will generate enormous amounts of data. The processing of this data will require huge computational and storage resources, and the associated human resources for operation and support. It was not considered feasible to concentrate all the resources at one site, and therefore it was agreed that the LCG computing service would be implemented as a geographically distributed ***Computational Data Grid***. This means that the service will use computing and storage resources installed at a large number of computing sites in many different countries, interconnected by fast networks. The gLite middleware hides much of the complexity of this environment from the user, giving the impression that all of these resources are available in a coherent virtual computer centre.

The users of a Grid infrastructure are divided into ***Virtual Organisations (VOs)*** [12], abstract entities grouping users, institutions and resources in the same administrative domain [13].

The WLCG/EGEE VOs correspond to real organisations or projects, such as the four LHC experiments, the community of biomedical researchers, etc. An updated list of all the EGEE VOs can be found at the CIC portal [14].

3.1. Preliminary Matters

3.1.1. Code Development

Many of the services offered by WLCG/EGEE can be accessed both by the user interfaces provided (CLIs or GUIs), or from applications by making use of various APIs. References to APIs used for particular services will be given later in the sections describing such services.

A totally different matter is the development of software that forms part of the gLite middleware itself. This falls outside the scope of this guide.

3.1.2. Troubleshooting

This document will also explain the meaning of the most common error messages and give some advice on how to avoid some common errors. This guide cannot, however, include all the possible failures a user may encounter while using gLite 3.0. These errors may be produced due to user mistakes, to misconfiguration of the Grid components, to hardware or network failures, or even to bugs in the gLite middleware.

Subsequent sections of this guide provide references to documents which go into greater detail about the gLite 3.0 components.

The ***Global Grid User Support (GGUS)*** [15] service provides centralised user support for WLCG/EGEE, by answering questions, tracking known problems, maintaining lists of frequently asked questions, providing links to documentation, etc. The GGUS portal is the key entry point for Grid users looking for help.

Finally, a user who thinks that there is a security risk in the Grid may directly contact the relevant site administrator if the situation is urgent, as this may be faster than going through GGUS. Information on the local site contacts can be obtained from the Information Service or from the GOC database [16], which is described in Chapter 4.

3.1.3. User and VO utilities

This guide mainly covers information useful for the average user. Thus, only core WLCG/EGEE middleware is described. Nevertheless, there are several tools which are not part of the middleware, but may be very useful to users. Some of these tools are summarised in Appendix D.

Likewise, there are utilities that are only available to certain (authorised) users of the Grid. An example is the administration of the resources viewed by a VO or the installation of VO software on WLCG/EGEE nodes. Authorised users can install software on the computing resources of WLCG/EGEE:

the installed software is also published in the Information Service, so that users can select sites where the software they need is installed. Information on such topics is given in Appendix E.

3.2. The WLCG/EGEE Architecture

This section provides a quick overview of the WLCG/EGEE architecture and services.

3.2.1. Security

As explained earlier, the WLCG/EGEE user community is grouped into Virtual Organisations. Before WLCG/EGEE resources can be used, a user must read and agree to the WLCG/EGEE usage rules and any further rules for the VO she wishes to join, and register some personal data with a Registration Service.

Once the user registration is complete, he can access WLCG/EGEE. The ***Grid Security Infrastructure (GSI)*** in WLCG/EGEE enables secure authentication and communication over an open network [17]. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol, with extensions for single sign-on and delegation.

In order to authenticate herself to Grid resources, a user needs to have a digital X.509 certificate issued by a ***Certification Authority (CA)*** trusted by WLCG/EGEE; Grid resources are generally also issued with certificates to allow them to authenticate themselves to users and other services.

The user certificate, whose private key is protected by a password, is used to generate and sign a temporary certificate, called a ***proxy certificate*** (or simply a proxy), which is used for the actual authentication to Grid services and does not need a password. As possession of a proxy certificate is a proof of identity, the file containing it must be readable only by the user, and a proxy has, by default, a short lifetime (typically 12 hours) to reduce security risks if it should be stolen.

The authorisation of a user on a specific Grid resource can be done in two different ways. The first is simpler, and relies on the ***grid-mapfile*** mechanism. The Grid resource has a local grid-mapfile which maps user certificates to local accounts. When a user's request for a service reaches a host, the certificate subject of the user (contained in the proxy) is checked against what is in the local grid-mapfile to find out to which local account (if any) the user certificate is mapped, and this account is then used to perform the requested operation [17]. The second way relies on the ***Virtual Organisation Membership Service (VOMS)*** and the ***LCAS/LCMAPS*** mechanism, which allow for a more detailed definition of user privileges, and will be explained in more detail later.

A user needs a valid proxy to submit jobs; those jobs carry their own copies of the proxy to be able to authenticate with Grid services as they run. For long-running jobs, the job proxy may expire before the job has finished, causing the job to fail. To avoid this, there is a proxy renewal mechanism to keep the job proxy valid for as long as needed. The ***MyProxy server*** is the component that provides this functionality.

3.2.2. User Interface

The access point to the WLCG/EGEE Grid is the ***User Interface (UI)***. This can be any machine where users have a personal account and where their user certificate is installed. From a UI, a user can be authenticated and authorized to use the WLCG/EGEE resources, and can access the functionalities offered by the Information, Workload and Data management systems. It provides CLI tools to perform some basic Grid operations:

- list all the resources suitable to execute a given job;
- submit jobs for execution;
- cancel jobs;
- retrieve the output of finished jobs;
- show the status of submitted jobs;
- retrieve the logging and bookkeeping information of jobs;
- copy, replicate and delete files from the Grid;
- retrieve the status of different resources from the Information System.

In addition, the WLCG/EGEE APIs are also available on the UI to allow development of Grid-enabled applications.

3.2.3. Computing Element

A ***Computing Element (CE)***, in Grid terminology, is some set of computing resources localized at a site (i.e. a cluster, a computing farm). A CE includes a ***Grid Gate (GG)***[1], which acts as a generic interface to the cluster; a ***Local Resource Management System (LRMS)*** (sometimes called ***batch system***), and the cluster itself, a collection of ***Worker Nodes (WNs)***, the nodes where the jobs are run.

There are two GG implementations in gLite 3.0: the ***LCG CE***, developed by EDG and used in LCG-2[2], and the ***gLite CE***, developed by EGEE. Sites can choose what to install, and some of them provide both types. The GG is responsible for accepting jobs and dispatching them for execution on the WNs via the LRMS.

In gLite 3.0 the supported LRMS types are OpenPBS, LSF, Maui/Torque, BQS and Condor, with work underway to support Sun GridEngine.

[1]For Globus-based CEs, it is called ***Gatekeeper***.
[2]LCG-2 is the former middleware stack used by WLCG/EGEE.

The WNs generally have the same commands and libraries installed as the UI, apart from the job management commands. VO-specific application software may be preinstalled at the sites in a dedicated area, typically on a shared file system accessible from all WNs.

It is worth stressing that, strictly speaking, a CE corresponds to a single ***queue*** in the LRMS, following this naming syntax:

```
CEName = <gg\_hostname>:<port>/jobmanager-<LRMS_type>-<batch_queue_name>
```

According to this definition, different queues defined in the same cluster are considered different CEs. This is currently used to define different queues for jobs of different lengths or other properties (e.g. RAM size), or from different VOs. Examples of CE names are:

```
ce101.cern.ch:2119/jobmanager-lcglsf-grid_alice
t2-ce-01.mi.infn.it:2119/jobmanager-lcgpbs-short
```

3.2.4. Storage Element

A ***Storage Element (SE)*** provides uniform access to storage resources. The Storage Element may control simple disk servers, large disk arrays or tape-based Mass Storage Systems (MSS). Most WLCG/EGEE sites provide at least one SE.

Storage Elements can support different data access protocols and interfaces, described in detail in Section 7.2. Simply speaking, ***GSIFTP*** (a GSI-secure FTP) is the protocol for whole-file transfers, while local and remote file access is performed using ***RFIO*** or ***gsidcap***.

Some storage resources are managed by a ***Storage Resource Manager (SRM)*** [19], a middleware module providing capabilities like transparent file migration from disk to tape, file pinning, space reservation, etc. However, different SEs may support different versions of the SRM protocol and the capabilities can vary.

Disk-only SEs are normally implemented as ***classic SEs***, which do not have an SRM interface, or using the ***Disk Pool Manager (DPM)***, which is SRM-enabled; classic SEs are going to be phased out soon.

Mass Storage Systems (with front-end disks and back-end tape storage), like ***CASTOR***, and large disk arrays (e.g. managed with ***dCache***) always provide an SRM interface.

The most common types of SEs currently present in WLCG/EGEE are summarized in the following table:

Type	Resources	File transfer	File I/O	SRM
Classic SE	Disk server	GSIFTP	insecure RFIO	No
DPM	Disk pool	GSIFTP	secure RFIO	Yes
dCache	Disk pool/MSS	GSIFTP	gsidcap	Yes
CASTOR	MSS	GSIFTP	insecure RFIO	Yes

3.2.5. Information Service

The ***Information Service (IS)*** provides information about the WLCG/EGEE Grid resources and their status. This information is essential for the operation of the whole Grid, as it is via the IS that resources are discovered. The published information is also used for monitoring and accounting purposes.

Much of the data published to the IS conforms to the ***GLUE Schema***, which defines a common conceptual data model to be used for Grid resource monitoring and discovery. More details about the GLUE schema can be found in Appendix G.

Two IS systems are used in gLite 3.0: the ***Globus Monitoring and Discovery Service (MDS)*** [20], used for resource discovery and to publish the resource status, and the ***Relational Grid Monitoring Architecture (R-GMA)*** [21], used for accounting, monitoring and publication of user-level information.

MDS

The Globus MDS implements the GLUE Schema using OpenLDAP, an open source implementation of the ***Lightweight Directory Access Protocol (LDAP)***, a specialised database optimised for reading, browsing and searching information. Access to MDS data is insecure, both for reading (clients and users) and for writing (services pubilishing information).

The LDAP information model is based on *entries* (objects like a person, a computer, a server, etc.), each with one or more *attributes*. Each entry has a ***Distinguished Name (DN)*** that uniquely identifies it, and each attribute has a type and one or more values.

A DN is formed by a sequence of attributes and values, and based on their DNs entries can be arranged into a hierarchical tree-like structure, called a ***Directory Information Tree (DIT)***.

Figure 1 schematically depicts the Directory Information Tree (DIT) of a site: the root entry identifies the site, and entries for site information, CEs, SEs and the network are defined in the second level. Actual entries published in gLite 3.0 are shown in Appendix G.

The ***LDAP schema*** describes the information that can be stored in each entry of the DIT and defines ***object classes***, which are collections of mandatory and optional attribute names and value types. While a directory entry describes some object, an object class can be seen as a general description of an object, as opposed to the description of a particular instance.

Figure 2 shows the MDS architecture in WLCG/EGEE. Computing and storage resources at a site run a piece of software called an ***Information Provider***, which generates the relevant information about the resource (both static, like the type of SE, and dynamic, like the used space in an SE). This information is published via an LDAP server called a ***Grid Resource Information Server (GRIS)***, which normally runs on the resource itself.

At each site another LDAP server, called a ***Site Grid Index Information Server (GIIS)***, collects the information from the local GRISes and republishes it. In WLCG/EGEE, the GIIS uses a ***Berkeley Database Information Index (BDII)*** to store data, which is more stable than the original Globus GIIS.

Finally, a BDII is also used as the top level of the hierarchy: this BDII queries the GIISes at every site and acts as a cache by storing information about the Grid status in its database. The BDII therefore contains all the available information about the Grid. Nevertheless, it is always possible to get information about specific resources by directly contacting the GIISes or even the GRISes.

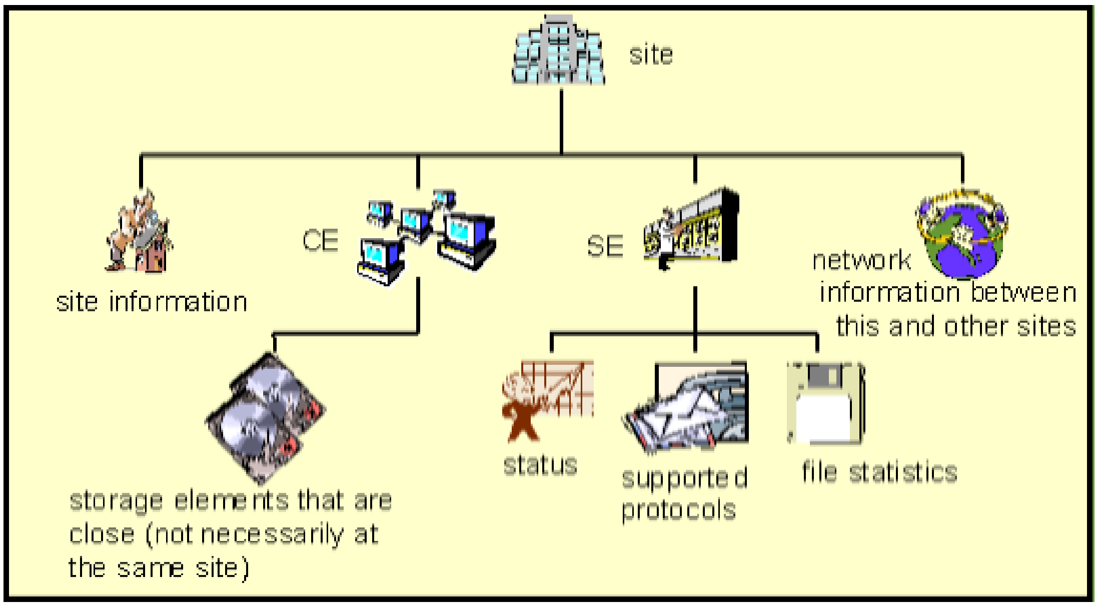

site
CE
SE
site information
network information between this and other sites
storage elements that are close (not necessarily at the same site)
status
supported protocols
file statistics

Figure 1: The Directory Information Tree (DIT)

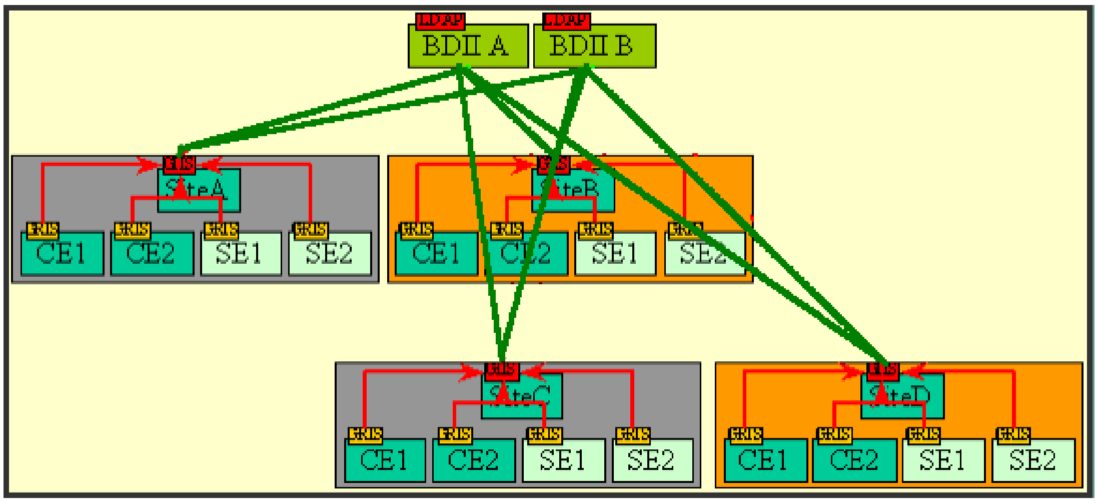


Figure 2: The MDS Information Service in WLCG/EGEE.

The top-level BDII obtains information about the sites in the Grid from the Grid Operations Centre (GOC) database, where site managers can insert the contact address of their GIIS as well as other useful information about the site.

R-GMA

R-GMA is an implementation of the ***Grid Monitoring Architecture (GMA)*** proposed by the ***Global Grid Forum (GGF)*** [22]. In R-GMA, information is in many ways presented as though it were in a global distributed relational database, although there are some differences (for example, a table may have multiple rows with the same primary key). This model is more powerful than the LDAP-based one, since relational databases support more advanced query operations. It is also much easier to modify the schema in R-GMA, making it more suitable for user information.

The architecture consists of three major components (Figure 3):

- The ***Producers***, which provide the information, register themselves with the Registry and describe the type and structure of the information they provide.
- The ***Consumers***, which request the information, can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known, the Consumer can contact the Producer directly to obtain the relevant data.
- The ***Registry***, which mediates the communication between the Producers and the Consumers.

Note that the producers and consumers are processes (servlets) running in a server machine at each site (sometimes known as a MON box). Users interact with these servlets using CLI tools or APIs on the WNs and UIs, and they in turn interact with the Registry, and with consumers and producers at other sites, on the user's behalf.

From the user's point of view, the information and monitoring system appears like a large relational database and it can be queried as such. Hence, R-GMA uses a subset of SQL as query language. The Producers publish ***tuples*** (database rows) with an SQL `insert` statement and Consumers query them using SQL `select` statements.

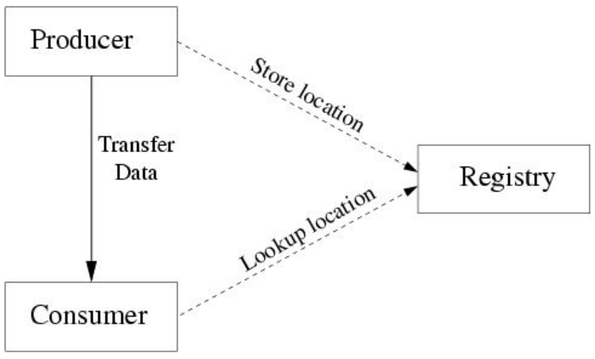


Figure 3: The R-GMA architecture.

R-GMA presents the information as a single virtual database containing a set of virtual tables. A ***schema*** contains the name and structure (column names, types and settings) of each virtual table in the system (Figure 4). The registry contains a list of producers which publish information for each table. A consumer runs an SQL query for a table and the registry selects the best producers to answer the query through a process called ***mediation***. The consumer then contacts each producer directly, combines the information and returns a set of tuples. The details of this process are hidden from the user, who just receives the tuples in response to a query.

An R-GMA system is defined by the registry and the schema: what information will be seen by a consumer depends on what producers are registered with the registry. There is only one registry and one schema in the WLCG/EGEE Grid.

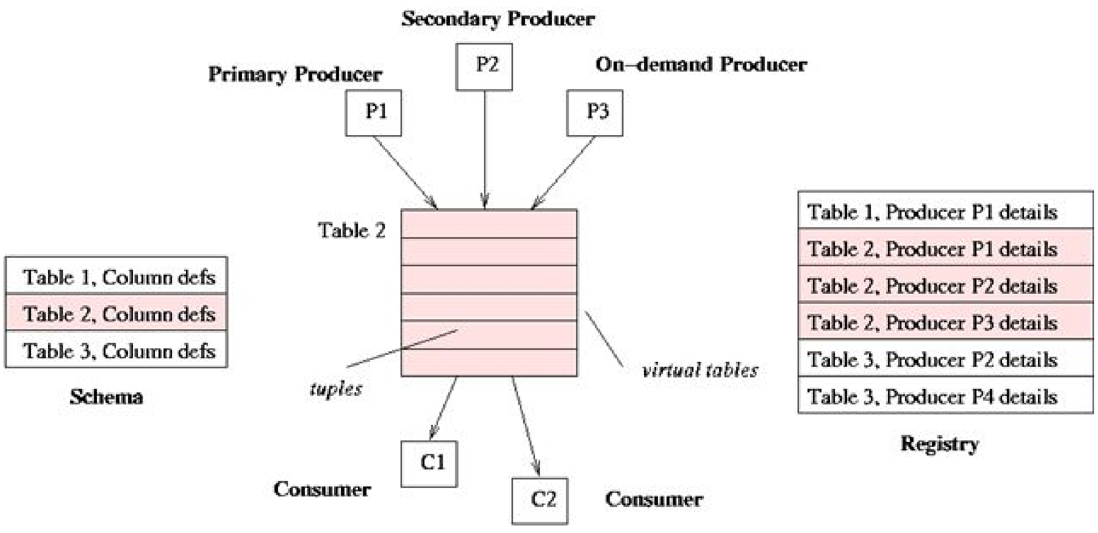


Figure 4: The virtual database of R-GMA

There are two types of producers: ***primary producers***, which publish information coming from a user or an Information Provider, and ***secondary producers***, which consume and republish information from primary producers and normally store it in a real database.

Producers can also be classified depending on the type of queries accepted:

- ***Continuous (or stream)***: information is sent directly to consumers as it is produced;
- ***Latest***: only the latest information (the tuple with the most recent timestamp for a given value of the primary key) is sent to the consumer;
- ***History***: all tuples within a configurable retention period are stored to allow subsequent retrieval by consumers.

Latest queries correspond most directly to a standard query on a real database. Primary producers are usually of type *continuous*. Secondary producers (which often use a real database to store the data) must be set up in advance to archive information and be able to reply to *latest* and/or *history* queries. Secondary producers are also required for joins to be supported in the consumer queries.

R-GMA is currently used for accounting and both system and user-level monitoring. It also holds the same GLUE schema information as the MDS, although this is not currently used to locate resources for job submission.

3.2.6. Data Management

In a Grid environment, ***files*** can have ***replicas*** at many different sites. Ideally, the users do not need to know where a file is located, as they use logical names for the files that the Data Management services will use to locate and access them.

Files in the Grid can be referred to by different names: ***Grid Unique IDentifier (GUID)***, ***Logical File Name (LFN)***, ***Storage URL (SURL)*** and ***Transport URL (TURL)***. While the GUIDs and LFNs identify a file irrespective of its location, the SURLs and TURLs contain information about where a physical replica is located, and how it can be accessed.

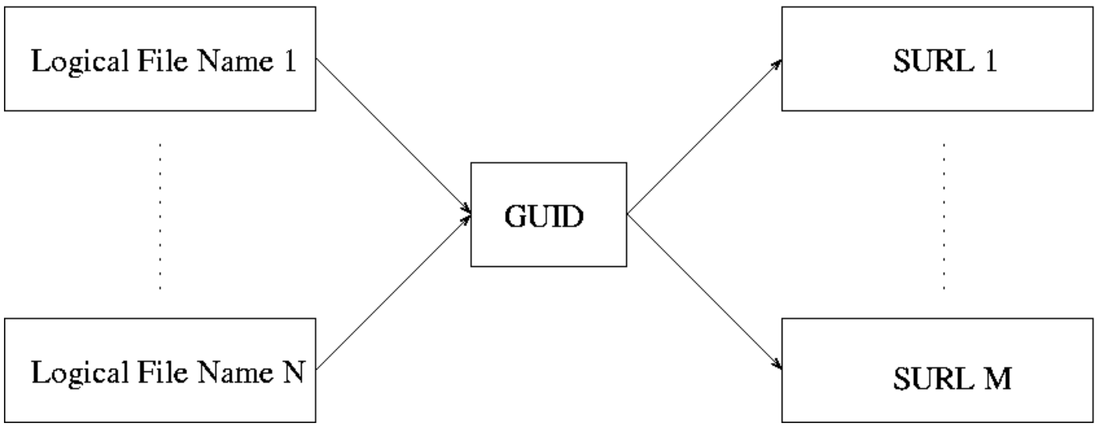


Figure 5: Different filenames in gLite 3.0.

A file can be unambigously identified by its GUID; this is assigned the first time the file is registered in the Grid, and is based on the UUID standard to guarantee its uniqueness. A GUID is of the form: `guid:<unique_string>` (e.g. `guid:93bd772a-b282-4332-a0c5-c79e99fc2e9c`). In order to locate a file in the Grid, a user will normally use an LFN. LFNs are usually more intuitive, human-readable strings, since they are allocated by the user. Their form is: `lfn:<any_alias>`. A Grid file can have many LFNs, in the same way as a file in a Unix file system can have many links.

The SURL provides informations about the physical location of the file. Currently, SURLs have the following formats:
`sfn:<SE_hostname>/<path>`
or
`srm:<SE_hostname>/<path>`
for files residing on a classic SE and on an SRM-enabled SE, respectively.

Finally, the TURL gives the necessary information to retrieve a physical replica, including hostname, path, protocol and port (as for any conventional URL), so that the application can open or copy it. The format is `<protocol>://<SE_hostname>:<port>/<path>`. There is no guarantee that the path, or even the hostname, in the SURL is the same as in the TURL for the same file. For a given file there can be as many TURLs as there are data access protocols supported by the SE. Figure 5 shows the relationship between the different names of a file.

The mappings between LFNs, GUIDs and SURLs are kept in a service called a ***File Catalog***, while the files themselves are stored in Storage Elements. The only file catalog officially supported in WLCG/EGEE is the ***LCG File Catalog (LFC)***.

The Data Management client tools are described in detail in Chapter 7. They allow a user to move data in and out of the Grid, replicate files between Storage Elements, interact with the File Catalog and more. LCG high level data management tools shield the user from the complexities of Storage Element and catalog implementations as well as transport and access protocols. Low level tools are also available, but should be needed only by expert users.

3.2.7. Workload Management

The purpose of the ***Workload Management System (WMS)*** is to accept user jobs, to assign them to the most appropriate Computing Element, to record their status and retrieve their output. The ***Resource Broker (RB)*** is the machine where the WMS services run [23] [24].

Jobs to be submitted are described using the ***Job Description Language (JDL)***, which specifies, for example, which executable to run and its parameters, files to be moved to and from the worker node, input Grid files needed, and any requirements on the CE and the worker node.

The choice of CE to which the job is sent is done in a process called ***match-making***, which first selects, among all available CEs, those which fulfill the requirements expressed by the user and which are close to specified input Grid files. It then chooses the CE with the highest ***rank***, a quantity derived from the CE status information which expresses the “goodness” of a CE (typically a function of the numbers of running and queued jobs).

The RB locates the Grid input files specified in the job description using a service called the ***Data Location Interface (DLI)***, which provides a generic interface to a file catalog. In this way, the Resource Broker can talk to file catalogs other than LFC (provided that they have a DLI interface).

The most recent implementation of the WMS from EGEE allows not only the submission of single jobs, but also collections of jobs (possibly with dependencies between them) in a much more efficient way then the old LCG-2 WMS [25].

Finally, the ***Logging and Bookkeeping service (LB)*** [26] tracks jobs managed by the WMS. It collects events from many WMS components and records the status and history of the job.

3.3. JOB FLOW

This section briefly describes what happens when a user submits a job to the WLCG/EGEE Grid to process some data, and explains how the different components interact.

3.3.1. Job Submission

Figure 6 illustrates the process that takes place when a job is submitted to the Grid. It refers to the LCG-2 WMS, but the gLite WMS is similar. The individual steps are as follows:

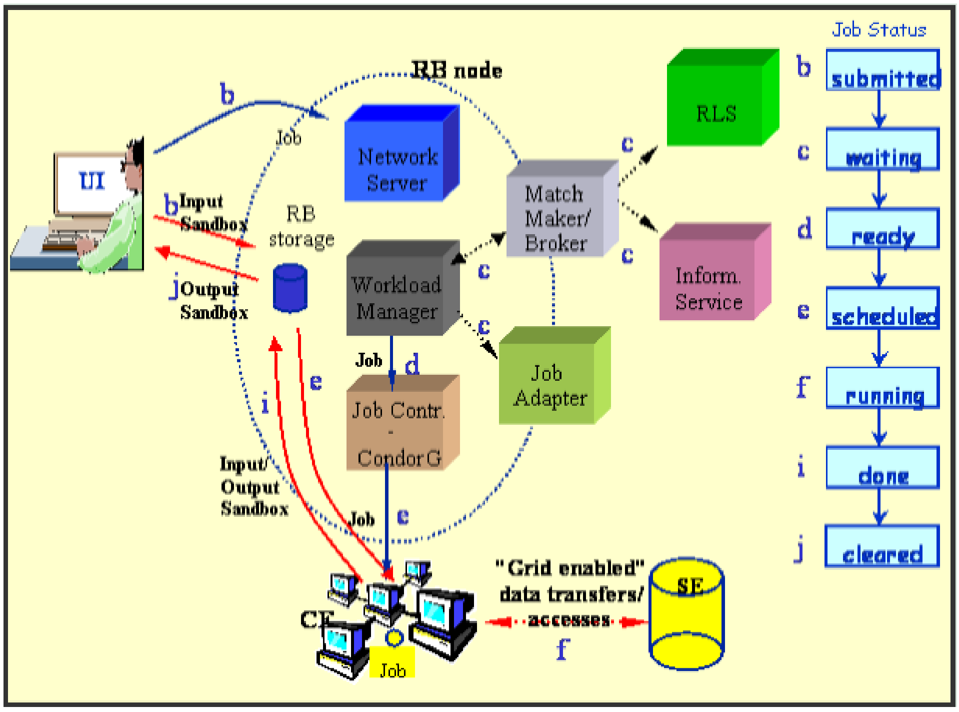


Figure 6: Job flow in the WLCG/EGEE Grid.

a. After obtaining a digital certificate from a trusted Certification Authority, registering in a VO and obtaining an account on a User Interface, the user is ready to use the WLCG/EGEE Grid. He logs in to the UI and creates a proxy certificate to authenticate himself in subsequent secure interactions.

b. The user submits a job from the UI to a Resource Broker. In the job description file one or more files to be copied from the UI to the WN can be specified. This set of files is called the ***Input Sandbox***. An event is logged in the LB and the status of the job is SUBMITTED.

c. The WMS looks for the best available CE to execute the job. To do so, it interrogates the BDII to query the status of computational and storage resources, and the File Catalog to find the location of any required input files. Another event is logged in the LB and the status of the job is WAITING.

d. The RB prepares the job for submission, creating a wrapper script that will be passed, together with other parameters, to the selected CE. An event is logged in the LB and the status of the job is READY.

e. The CE receives the request and sends the job for execution to the local LRMS. An event is logged in the LB and the status of the job is SCHEDULED.

f. The LRMS handles the job execution on the available local farm worker nodes. User files are copied from the RB to the WN where the job is executed. An event is logged in the LB and the status of the job is RUNNING.

g. While the job runs, Grid files can be directly accessed from an SE using either the secure RFIO or gsidcap protocols, or after copying them to the local filesystem on the WN with the Data Management tools.

h. The job can produce new output files which can be uploaded to the Grid and made available for other Grid users to use. This can be achieved using the Data Management tools described later. Uploading a file to the Grid means copying it to a Storage Element and registering it in a file catalog.

i. If the job ends without errors, the output (not large data files, but just small output files specified by the user in the so called ***Output Sandbox***) is transferred back to the RB node. An event is logged in the LB and the status of the job is DONE.

j. At this point, the user can retrieve the output of his job to the UI. An event is logged in the LB and the status of the job is CLEARED.

k. Queries for the job status can be addressed to the LB from the UI. Also, from the UI it is possible to query the BDII for the status of the resources.

l. If the site to which the job is sent is unable to accept or run it, the job may be automatically resubmitted to another CE that satisfies the user requirements. After a maximum allowed number of resubmissions is reached, the job will be marked as aborted. Users can get information about the history of a job by querying the LB service.

3.3.2. Other operations

While the Input and Output Sandboxes are a mechanism for transferring small data files needed to start a job or to check its results, large data files should be read and written from/to SEs and registered in a File Catalog, and possibly replicated to other SEs. The LCG Data Management client tools are available for performing these tasks.

In general, the user should not directly interact with the File Catalog; instead, she should use the LCG tools, or the POOL interface (see Section 7.9).

Users can interrogate the information system to retrieve static or dynamic information about the status of WLCG/EGEE resources and services. Although site GIISes/BDIIs, or even GRISes, can be directly queried, it is recommended to query only a central BDII (or R-GMA). Details and examples on how to interrogate GRIS, GIIS and BDII are given in Chapter 5.

4. Getting Started

This section describes the preliminary steps to gain access to the WLCG/EGEE Grid. Before using the WLCG/EGEE Grid, the user must do the following:

a. Obtain a Cryptographic X.509 certificate from a ***Certification Authority (CA)*** recognized by WLCG/EGEE;
b. Get registered with WLCG/EGEE by joining one of the WLCG/EGEE Virtual Organisations;
c. Obtain an account on a machine which has the WLCG/EGEE User Interface software installed;
d. Create a proxy certificate.

Steps a. to c. need to be executed only once to have access to the Grid. Step d. needs to be executed the first time a request to the Grid is submitted. It generates a proxy valid for a certain period of time (usually 12 hours). At the proxy expiration, a new proxy must be created before the Grid services can be used again.

The following sections provide details on the prerequisites.

4.1. Obtaining a Certificate

4.1.1. X.509 Certificates

The first requirement the user must fulfill is to be in possession of a valid X.509 certificate issued by a recognized Certification Authority (CA). The role of a CA is to guarantee that a user is who he claims to be and is entitled to own his certificate. It is up to the user to discover which CA he should contact. In general CAs are organized geographically and by research institute. Each CA has its own procedure to release certificates.

The following URL maintains an updated list of recognized CAs, as well as detailed information on how to request certificates from a particular CA:

http://lcg.web.cern.ch/LCG/users/registration/certificate.html

An important property of a certificate is the ***subject*** (or ***Distinguished Name, DN***), a string containing information about the user. A typical example is:

```
/O=Grid/O=CERN/OU=cern.ch/CN=John Doe
```

4.1.2. Requesting the Certificate

Generally speaking, obtaining a certificate involves creating a request to a CA. The request is normally generated using either a web-based interface or console commands. Details of which type of request a particular CA accepts are described on each CA's website.

For a web-based certificate request, a form must be usually filled in with information such as name of the user, institute, etc. After submission, a pair of private and public keys are generated together with a request for the certificate containing the public key and the user data. The request is then sent to the CA.

Note: The user must usually install the CA certificate on him browser first. This is because the CA has to sign the certificate using its own one, and the user's browser must recognize it as a valid one.

For some CAs the certificate requests are generated using a command line interface. The following discussion describes a common scenario for command-line certificate application using a hypothetical `grid-cert-request` command. Again, details of the exact command and requirements of each CA will vary and can be found on the CA's website.

The `grid-cert-request` command would create, for example, the following 3 files:

`userkey.pem`	contains the private key associated with the certificate. (**This should be set with permissions so that only the owner can read it**) (i.e. `chmod 400 userkey.pem`);
`userreq.pem`	contains the request for the user certificate (essentially, the publick key);
`usercert.pem`	a placeholder, to be replaced by the actual certificate when received from the CA (This should be readable by everyone) (i.e. `chmod 444 usercert.pem`).

Then the `userreq.pem` file has to be sent (usually by e-mail) to the desired CA.

4.1.3. Getting the Certificate

After a request is generated and sent to a CA, the CA will have to confirm that the user asking for a certificate is who he claims he is. This usually involves to physically meet, or to have a phone call, with a ***Registration Authority***, somebody delegated by the CA to verify the legitimacy of a request, and in case approve it. After approval, the certificate is generated and delivered to the user. This can be done via e-mail, or by giving instructions to the user to download it from a web page. If the certificate was directly installed in the user's browser, then it must be exported (saved) to disk for Grid use. Details of how to do this will depend on supported browser versions and are described on the CA's website.

The received certificate will usually be in one of two formats: ***PEM*** (extension `.pem`) or ***PKCS12*** format (extension `.p12`). This last one is the most common one for certificates installed in a browser, but it is the other one, the PEM format, which must be used in WLCG/EGEE. The certificates can be converted from one format to the other.

If the certificate is in PKCS12 format, then it can be converted to PEM using the `openssl` command:

```
$ openssl pkcs12 -nocerts -in my_cert.p12 -out userkey.pem
$ openssl pkcs12 -clcerts -nokeys -in my_cert.p12 -out usercert.pem
```

where:

`my_cert.p12`	is the input `PKCS12` format file.
`userkey.pem`	is the output private key file.
`usercert.pem`	is the output `PEM` certificate file.

The first command creates only the private key (due to the `-nocerts` option), and the second one creates the user certificate (`-nokeys -clcerts` option).

The `grid-change-pass-phrase -file <private_key_file>` command changes the pass phrase that protects the private key. This command will work even if the original key is not password protected. It is important to know that if the user loses the pass phrase, the certificate will become unusable and a new certificate will have to be requested.

Once in PEM format, the two files, `userkey.pem` and `usercert.pem`, should be copied to a User Interface. This will be described later.

4.1.4. Renewing the Certificate

Most CAs issue certificates with a limited duration (usually one year); this implies the need to renew it periodically. The renewal procedure usually requires that the certificate holder sends a request for renewal signed with the old certificate and/or that the request is confirmed by a phone call; the details depend on the policy of each CA.

Renewed certificates have the same DN as the old ones; failing to renew the certificate usually implies the loss of the DN and the necessity to request a completely new certificate with a different DN.

4.2. Registering with WLCG/EGEE

4.2.1. The Registration Service

Before a user can use WLCG/EGEE, registration of some personal data and acceptance of some usage rules are necessary. In the process, the user must also choose a ***Virtual Organisation (VO)***. The VO must ensure that all its members have provided the necessary information to the VO's database and have accepted the usage rules. The procedure through which this is accomplished may vary from VO to VO: pointers to all the VOs in WLCG/EGEE can be found at

http://cic.in2p3.fr/.

As an example of registration service, we describe here the example of the ***LCG Registrar***, which serves the VOs of the LHC experiments. For detailed information please visit the following URL:

http://lcg-registrar.cern.ch/

The registration procedure normally requires to use a web browser with the user certificate loaded, for the request to be properly authenticated.

Browsers (including Internet Explorer and Mozilla) use the PKCS12 certificate format: if the certificate was issued to a user in PEM format, it has to be converted to PKCS12. The following command can be used to perform that conversion:

```
openssl pkcs12 -export -inkey userkey.pem -in usercert.pem \
              -out my_cert.p12 -name "My certificate"
```

where:

`userkey.pem`	is the path to the private key file.
`usercert.pem`	is the path to the `PEM` certificate file.
`my_cert.p12`	is the path for the output `PKCS12` format file to be created.
`"My certificate"`	is an optional name which can be used to select this certificate in the browser after the user has uploaded it if the user has more than one.

Once in PKCS12 format, the certificate can be loaded into the WWW browser. Instructions about how to do this for some popular browsers are available at:

http://lcg.web.cern.ch/LCG/users/registration/load-cert.html

4.2.2. Virtual Organisations

A VO is an entity which typically corresponds to a particular organisation or group of people in the real world. The membership of a VO grants specific privileges to the user. For example, a user belonging to the *ATLAS* VO will be able to read the ATLAS files or to exploit resources reserved to the ATLAS collaboration.

Becoming member of a VO usually requires being a member of the corresponding collaboration; the user must comply with the rules of the VO to gain membership. Of course, it is also possible to be expelled from a VO when the user fails to comply with these rules.

Currently, it is only possible to register a certificate to one VO at a time: the only way to belong to more than one VO is to use different certificates for each one.

4.3. Setting Up the User Account

4.3.1. The User Interface

Apart from registering with WLCG/EGEE, a user must also have an account on a WLCG/EGEE User Interface in order to access the Grid. To obtain such an account, a local system administrator must be contacted. The official list of LCG sites is available in the GOC database [16].

As an example, the CERN LXPLUS service can be used as UI as described in [28]. This use could be extended to other (non LXPLUS) machines mounting AFS.

Once the account has been created, the user certificate must be installed. For that, it is necessary to create a directory named `.globus` under the user home directory and put the user certificate and key files there, naming them `usercert.pem` and `userkey.pem` respectively, with permissions `0444` for the former, and `0400` for the latter. A directory listing should give a result similar to this:

```
ls -l $HOME/.globus
```

```
total 13
-rw-r--r--    1 doe      xy          4541 Aug 23  2005 usercert.pem
-r--------    1 doe      xy           963 Aug 23  2005 userkey.pem
```

4.3.2. Checking a Certificate

To verify that a certificate is not corrupted and print information about it, the Globus command `grid-cert-info` can be used from the UI. The `openssl` command can be used instead to verify the validity of a certificate with respect to the certificate of the certification authority that issued it. The command `grid-proxy-init` can be used to check if there is a mismatch between the private key and the certificate.

Example 4.3.2.1   (Retrieving information on a user certificate)

With the certificate properly installed in the `$HOME/.globus` directory of the user's UI account, issue the command:

```
$ grid-cert-info
```

If the certificate is properly formed, the output will be something like:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 5 (0x5)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=CH, O=CERN, OU=cern.ch, CN=CERN CA
        Validity
            Not Before: Sep 11 11:37:57 2002 GMT
            Not After : Nov 30 12:00:00 2003 GMT
        Subject: O=Grid, O=CERN, OU=cern.ch, CN=John Doe
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA  Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ab:8d:77:0f:56:d1:00:09:b1:c7:95:3e:ee:5d:
                    c0:af:8d:db:68:ed:5a:c0:17:ea:ef:b8:2f:e7:60:
                    2d:a3:55:e4:87:38:95:b3:4b:36:99:77:06:5d:b5:
                    4e:8a:ff:cd:da:e7:34:cd:7a:dd:2a:f2:39:5f:4a:
                    0a:7f:f4:44:b6:a3:ef:2c:09:ed:bd:65:56:70:e2:
                    a7:0b:c2:88:a3:6d:ba:b3:ce:42:3e:a2:2d:25:08:
                    92:b9:5b:b2:df:55:f4:c3:f5:10:af:62:7d:82:f4:
                    0c:63:0b:d6:bb:16:42:9b:46:9d:e2:fa:56:c4:f9:
                    56:c8:0b:2d:98:f6:c8:0c:db
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Netscape Base Url:
                http://home.cern.ch/globus/ca
            Netscape Cert Type:
                SSL Client, S/MIME, Object Signing
            Netscape Comment:
               For DataGrid use only
            Netscape Revocation Url:
                http://home.cern.ch/globus/ca/bc870044.r0
            Netscape CA Policy Url:
                http://home.cern.ch/globus/ca/CPS.pdf
    Signature Algorithm: md5WithRSAEncryption
        30:a9:d7:82:ad:65:15:bc:36:52:12:66:33:95:b8:77:6f:a6:
        52:87:51:03:15:6a:2b:78:7e:f2:13:a8:66:b4:7f:ea:f6:31:
        aa:2e:6f:90:31:9a:e0:02:ab:a8:93:0e:0a:9d:db:3a:89:ff:
        d3:e6:be:41:2e:c8:bf:73:a3:ee:48:35:90:1f:be:9a:3a:b5:
        45:9d:58:f2:45:52:ed:69:59:84:66:0a:8f:22:26:79:c4:ad:
        ad:72:69:7f:57:dd:dd:de:84:ff:8b:75:25:ba:82:f1:6c:62:
        d9:d8:49:33:7b:a9:fb:9c:1e:67:d9:3c:51:53:fb:83:9b:21:
        c6:c5
```

The `grid-cert-info` command takes many options. Use the `-help` for a full list. For example, the `-subject` option returns the certificate subject:

```
$ grid-cert-info -subject
/O=Grid/O=CERN/OU=cern.ch/CN=John Doe
```

Or, one can check the certificate expiration date:

```
$ grid-cert-info -enddate
Oct 15 05:37:09 2005 GMT
```

or, to know which CA issued the certificate:

```
$ grid-cert-info -issuer
/C=CH/O=CERN/OU=GRID/CN=CERN CA
```

Example 4.3.2.2 (Verifying a user certificate)

To verify a user certificate, issue the following command from the UI:

```
$ openssl verify -CApath /etc/grid-security/certificates ~/.globus/usercert.pem
```

and if the certificate is valid and properly signed, the output will be:

```
/home/doe/.globus/usercert.pem: OK
```

If the certificate of the CA that issued the user certificate is not found in `-CApath`, an error message like this will appear:

```
usercert.pem: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
error 20 at 0 depth lookup:unable to get local issuer certificate
```

If the variable X509_CERT_DIR is defined, use its value as argument of `/etc/grid-security/certificates`.

Example 4.3.2.3 (Verifying the consistency between private key and certificate)

If for some reason the user is using a certificate (`usercert.pem`) which does not correspond to the private key (`userkey.pem`), strange errors may occur. To test if this is the case, run the command:

```
grid-proxy-init -verify
```

In case of mismatch, the output will be:

```
Your identity: /C=CH/O=CERN/OU=GRID/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ................................ Done


ERROR: Couldn't verify the authenticity of the user's credential to
generate a proxy from.
Use -debug for further information.
```

4.4. PROXY CERTIFICATES

4.4.1. Proxy Certificates

At this point, the user is able to generate a ***proxy certificate***. A proxy certificate is a delegated user credential that authenticates the user in every secure interaction, and has a limited lifetime: in fact, it prevents having to use one's own certificate, which could compromise its safety, and avoids having to give the certificate pass phrase each time.

The command to create a proxy certificate is `grid-proxy-init`, which prompts for the user pass phrase, as in the next example.

Example 4.4.1.1 ***(Creating a proxy certificate)***

To create a proxy certificate, issue the command:

```
$ grid-proxy-init
```

If the command is successful, the output will be like

```
Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ............................................. Done
Your proxy is valid until: Tue Jun 24 23:48:44 2003
```

and the proxy certificate will be written in `/tmp/x509up_u<uid>`, where `<uid>` is the Unix UID of the user, unless the environment variable X509_USER_PROXY is defined, in which case its value is taken as the proxy file name.

If the user gives a wrong pass phrase, the output will be

```
ERROR: Couldn't read user key. This is likely caused by
either giving the wrong pass phrase or bad file permissions
key file location: /home/doe/.globus/userkey.pem
Use -debug for further information.
```

If the proxy certificate file cannot be created, the output will be

```
ERROR: The proxy credential could not be written to the output file.
Use -debug for further information.
```

If the user certificate files are missing, or the permissions of `userkey.pem` are not correct, the output is:

```
ERROR: Couldn't find valid credentials to generate a proxy.
Use -debug for further information.
```

By default, the proxy has a lifetime of 12 hours. To specify a different lifetime, the `-valid H:M` option can be used (the proxy is valid for `H` hours and `M` minutes –default is 12:00). The old option `-hours` is deprecated. When a proxy certificate has expired, it becomes useless and a new one has to be created with `grid-proxy-init`. Longer lifetimes imply bigger security risks, though. Use the option `-help` for a full listing of options.

It is also possible to print information about an existing proxy certificate, or to destroy it before its expiration, as in the following examples.

Example 4.4.1.2 ***(Printing information on a proxy certificate)***

To print information about a proxy certificate, for example, the subject or the time left before expiration, give the command:

```
$ grid-proxy-info
```

The output, if a valid proxy exists, will be similar to

```
subject  : /O=Grid/O=CERN/OU=cern.ch/CN=John Doe/CN=proxy
issuer   : /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
type     : full
strength : 512 bits
path     : /tmp/x509up_u7026
timeleft : 11:59:56
```

If a proxy certificate does not exist, the output is:

```
ERROR: Couldn't find a valid proxy.
Use -debug for further information.
```

Example 4.4.1.3 (Destroying a proxy certificate)

To destroy an existing proxy certificate before its expiration, it is enough to do:

```
$ grid-proxy-destroy
```

If no proxy certificate exists, the result will be:

```
ERROR: Proxy file doesn't exist or has bad permissions
Use -debug for further information.
```

4.4.2. Virtual Organisation Membership Service

The ***Virtual Organisation Membership Service (VOMS)*** is a system which allows to complement a proxy certificate with ***extensions*** containing information about the VO, the VO groups the user belongs to, and the role the user has.

In VOMS terminology, a ***group*** is a subset of the VO containing members who share some responsibilites or privileges in the project. Groups are organized hierarchically like a directory tree, starting from a VO-wide root group. A user can be a member of any number of groups, and a VOMS proxy contains the list of all groups the user belongs to, but when the VOMS proxy is created, the user can choose one of these groups as the “primary” group.

A ***role*** is an attribute which typically allows a user to acquire special privileges to perform specific tasks. In principle, groups are associated to privileges that the user always has, while roles are associated to privileges that a user needs to have only from time to time.

To map groups and roles to specific privileges, what counts is the group/role combination, which is sometimes referred to as a FQAN (short form for Fully Qualified Attribute Name). The format is:

```
FQAN = <group name>[/Role=<role name>]
```

for example, `/cms/HeavyIons/Role=production`. It should be noted that a role is by no means global, but is always associated to a group.

Example 4.4.2.1 (Creating a VOMS proxy)

The `voms-proxy-init` command generates a Grid proxy, contacts a VOMS server, retrieves the user attributes and includes them in the proxy. If used without arguments, it works exactly as `grid-proxy-init`.

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

```
$ voms-proxy-init -voms <vo>
```

where `<vo>` is the VO of the user. The output is similar to:

```
Your identity: /C=CH/O=CERN/OU=GRID/CN=John Doe
Enter GRID pass phrase:
Creating temporary proxy ............................................. Done
Contacting  lcg-voms.cern.ch:15002 [/C=CH/O=CERN/OU=GRID/CN=host/lcg-voms.cern.ch]
"cms" Done
Creating proxy ................................................................... Done
Your proxy is valid until Thu Mar 30 06:17:27 2006
```

One clear advantage of VOMS proxies over normal proxies is that the middleware can find out to which VO the user belongs to from the proxy, while using a normal proxy the VO has to be explicitly specified by other means.

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

```
$ voms-proxy-init -voms <alias>:<group name>[Role=<role name>]
```

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

```
$ voms-proxy-init -voms cms:/cms/HeavyIons/Role=production}
```

voms-proxy-init uses a configuration file, whose path can be specified in several ways; if the file is a directory, the files inside it are concatenated and taken as the actual configuration file. A user-level configuration file, which must be owned by the user, is looked for in these locations:

- the argument of the -userconf option;
- the file $HOME/.glite/vomses.

If it is not found, a system-wide configuration file, which must be owned by root, is looked for in these locations:

- the argument of the -confile option;
- the file $GLITE_LOCATION/etc/vomses;
- the file /opt/glite/etc/vomses.

The configuration file must contain lines with the following syntax:

```
alias host port subject vo
```

where the items are respectively an alias (usually the name of the VO), the host name of the VOMS server, the port number to contact for a given VO, the DN of the server host certificate, and the name of the VO. For example:

```
"dteam" "lcg-voms.cern.ch" "15004"
"/C=CH/O=CERN/OU=GRID/CN=host/lcg-voms.cern.ch" "dteam"
```

Example 4.4.2.2 (Printing information on a VOMS proxy)

The voms-proxy-info command is used to print information about an existing VOMS proxy. Two useful options are -all, which prints everything, and fqan, which prints the groups and roles in FQAN format. For example:

```
$ voms-proxy-info -all
subject   : /C=CH/O=CERN/OU=GRID/CN=John Doe/CN=proxy
issuer    : /C=CH/O=CERN/OU=GRID/CN=John Doe
identity  : /C=CH/O=CERN/OU=GRID/CN=John Doe
type      : proxy
strength  : 512 bits
path      : /tmp/x509up_u10585
timeleft  : 11:59:58
=== VO cms extension information ===
VO        : cms
subject   : /C=CH/O=CERN/OU=GRID/CN=John Doe
issuer    : /C=CH/O=CERN/OU=GRID/CN=host/lcg-voms.cern.ch
attribute : /cms/Role=NULL/Capability=NULL
timeleft  : 11:59:58
```

4.4.3. Proxy Renewal

Proxy certificates created as described in the previous section pose a problem: if the job does not finish before the expiration time of the proxy used to submit it, is aborted. This can easily happen, for example, if the job takes very long to execute, or if it stays in the queue for very long. The easiest solution to the problem is to use very long lived proxies, but at the expense of an increased security risk. Moreover, the duration of a VOMS proxy is limited by the VOMS server and cannot be made arbitrarily long.

To overcome this limitation, a proxy credential repository system is used, which allows the user to create and store a long-term proxy certificate on a dedicated server (***MyProxy server***). The WMS will then be able to use this long-term proxy to periodically renew the proxy for a submitted job before it expires and until the job ends (or the long-term proxy expires).

To see if a WLCG/EGEE site has a MyProxy Server, the GOC database should be consulted; MyProxy servers have as node type `PROX`.

As the renewal process starts 30 minutes before the old proxy expires, it is necessary to generate an initial proxy long enough, or the renewal may be triggered too late, after the job has failed with the following error:

```
Status Reason: Got a job held event, reason: Globus error 131:
the user proxy expired (job is still running)
```

The minimum recommended time for the initial proxy is 30 minutes, and job submission is forbidden for proxies with a remaining lifetime less than 20 minutes: an error message like the following will be produced:

```
**** Error: UI_PROXY_DURATION ****
Proxy certificate will expire within less then 00:20 hours.
```

The advanced proxy management offered by the UI of gLite 3.0 through the renewal feature is available via the `myproxy` commands. The user must know the host name of a MyProxy server, or it may be the value of the MYPROXY_SERVER environment variable.

For the WMS to know what MyProxy server must be used in the proxy certificate renewal process, the name of the server must be included in an attribute of the job's JDL file (see Chapter 6). If the user does not add it manually, then the name of the default MyProxy server is added automatically when the job is submitted. This default MyProxy server node is VO-dependent and is usually defined in the UI VO's configuration file, stored at `$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf` (for the old LCG job management) and at `$GLITE_WMS_LOCATION/etc/<vo>/glite_wmsui.conf` (for the new gLite job management).

Example 4.4.3.1 (Creating a long-term proxy and storing it in a MyProxy Server)

To create and store a long-term proxy certificate, the user must do, for example:

```
$ myproxy-init -s <myproxy_server> -d -n
```

where `-s <myproxy_server>` specifies the hostname of the machine where a MyProxy Server runs, the `-d` option instructs the server to associate the user DN to the proxy, and the `-n` option avoids the use of a pass phrase to access to the long-term proxy, so that the WMS can perform the renewal automatically.

The output will be similar to:

```
Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ............................................. Done
Your proxy is valid until: Thu Jul 17 18:57:04 2003
A proxy valid for 168 hours (7.0 days) for user /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
now exists on myproxy.cern.ch.
```

By default, the long-term proxy lasts for one week and the proxy certificates created from it last 12 hours. These lifetimes can be changed using the `-c` and the `-t` option, respectively.

If the `-s <myproxy_server>` option is missing, the command will try to use the MYPROXY_SERVER environment variable to determine the MyProxy Server.

ATTENTION! If the hostname of the MyProxy Server is wrong, or the service is unavailable, the output will be similar to:

```
Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ...................................... Done
Your proxy is valid until: Wed Sep 17 12:10:22 2003
Unable to connect to adc0014.cern.ch:7512
```

where only the last line reveals that an error occurred.

Example 4.4.3.2 (Retrieving information about a long-term proxy)

To get information about a long-term proxy stored in a Proxy Server, the following command may be used:

```
$ myproxy-info -s <myproxy_server> -d
```

where the `-s` and `-d` options have the same meaning as in the previous example.

The output is similar to:

```
username: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
owner: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
timeleft: 167:59:48  (7.0 days)
```

Note that the user must have a valid proxy certificate on the UI, created with grid-proxy-init, to successfully interact with his long-term certificate on the Proxy server.

Example 4.4.3.3 (Deleting a long-term proxy)

Deleting a stored long-term proxy is achieved by doing:

```
$ myproxy-destroy -s <myproxy_server> -d
```

And the output is:

```
Default MyProxy credential for user /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
was successfully removed.
```

Also in this case, a valid proxy certificate must exist for the user on the UI.

5. INFORMATION SERVICE

The architecture of the gLite 3.0 Information Services, both MDS and R-GMA, was described in Chapter 3. In this chapter, we have a closer look at the structure of the information published by different elements of those architectures, and we examine the tools that can be used to get information from them.

Remember that although most middleware components (from Data and Workload Management) rely on MDS, R-GMA is already in use and many applications, specially for accounting and monitoring purposes, depend on it.

Information of current tools used for monitoring in gLite 3.0 is also provided.

5.1. THE MDS

In the following sections examples are given on how to interrogate the MDS Information Service in gLite 3.0. In particular, the different servers from which the information can be obtained are discussed. These are the local GRISes, the site GIISes/BDIIs and the global (or top) BDIIs. Of them, the BDII is usually the one queried, since it contains all the interesting information for a VO in a single place.

But before the procedure to query directly the IS elements is described, two higher level tools, `lcg-infosites` and `lcg-info`, are presented. These tools should be enough for most common user needs and will usually avoid the necessity of raw LDAP queries (though these are very useful for more complex or subtle requirements).

As explained in Chapter 3, the data in the IS of WLCG/EGEE conforms to the LDAP implementation of the GLUE Schema, although some extra attributes (not initially in the schema) are also being published and actually queried and used by clients of the IS. For a list of the defined object classes and their attributes, as well as for a reference on the Directory Information Tree used to publish those attributes, please check Appendix G.

As usual, the tools to query the IS shown in this section are command line based. There exist, however, graphical tools that can be used to browse the LDAP catalogs. As an example, the program **gq** is open source and can be found in some Linux distributions by default. Some comments on this tool are given in Section 5.1.5.

5.1.1. lcg-infosites

The `lcg-infosites` command can be used as an easy way to retrieve information on Grid resources for the most common use cases.

USAGE: `lcg-infosites --vo <vo name> options -v <verbose level> -f <Name of the site> --is <BDII to query>`

Description of the Attributes:

vo	: The name of the user vo (mandatory)
options	: The tool admits the following options:

ce : The information related to number of CPUs, running jobs, waiting jobs and names of the CEs are provided. All these data group all VOs together.
-v 1 only the names of the queues will be printed.
-v 2 The RAM Memory together with the operating system and its version and the processor included in each CE are printed.

se : The names of the SEs supported by the user's VO together with the kind of Storage System, the used and available space will be printed.
-v 1 only the names of the SEs will be printed.

closeSE : The names of the CEs where the user's VO is allowed to run together with their corresponding closest SEs are provided

lrc (rmc) : The name of the lrc (rmc) corresponding to the user's VO

lfc (lfcLocal) : The name of the (local) machine hosting the LFC catalog is printed.

rb : Names of the RBs availables for each VO.

dli : Names of the dlis for each VO.

vobox : Names of the VOBOX available for each VO.

fts : Names of the fts Endpoints per VO.

sitenames : Names of the LCG sites

NOTE : rb, dli, vobox, fts support the f option.

tag : The names of the tags relative to the software installed in site is printed together with the corresponding CE.

all : It groups together the information provided by ce and se.

is : If not specified the BDII defined in default by the variable LCG_GFAL_INFOSYS will be queries. However the user may want to query any other BDII without redefining this environment variable. This is possible specifying this argument followed by the name of the BDII which the user wants to query. All options admits this argument.

f : Giving as input the name of the site, the chosen local service is provided.

Example 5.1.1.1 ***(Obtaining information about computing resources)***

The way to get the information relative to the computing resources for a certain VO is:

```
$ lcg-infosites --vo alice ce
```

A typical output is as follows:

```
****************************************************************
These are the related data for alice: (in terms of queues and CPUs)
****************************************************************

#CPU    Free    Total Jobs      Running Waiting ComputingElement
```

```
----------------------------------------------------------
   2        2       0             0        0    globus.it.uom.gr:2119/blah-pbs-alice
  12       12       0             0        0    pprod03.lip.pt:2119/jobmanager-pbs-alice
   4        4       0             0        0    zeus76.cyf-kr.edu.pl:2119/blah-pbs-alice
   4        3       0             0        0    lxb2087.cern.ch:2119/jobmanager-lcgpbs-alice
   2        2       0             0        0    pps-ce-fzk.gridka.de:2119/jobmanager-lcgpbs-pps
3903      132       0             0        0    lxb2055.cern.ch:2119/jobmanager-lcglsf-grid_glite
   5        5       0             0        0    cclcgceli07.in2p3.fr:2119/jobmanager-bqs-short
   5        5       0             0        0    cclcgceli07.in2p3.fr:2119/jobmanager-bqs-medium
  34        0       0             0        0    prep-ce-02.pd.infn.it:2119/jobmanager-lcglsf-alice
   4        4       0             0        0    zeus75.cyf-kr.edu.pl:2119/jobmanager-lcgpbs-alice
   1        1       0             0        0    cclcgceli07.in2p3.fr:2119/jobmanager-bqs-alice_long
   2        2       0             0        0    tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-alice
[...]
```

Example 5.1.1.2   ***(Obtaining information about storage resources)***

To know the status of the storage resources:

```
$ lcg-infosites --vo atlas se


**************************************************************
These are the related data for atlas: (in terms of SE)
**************************************************************

Avail Space(Kb) Used Space(Kb)  Type    SEs
----------------------------------------------------------
0               0               n.a     se02.lip.pt
160250000       8860000         n.a     se01d.lip.pt
1000000000000   500000000000    n.a     castorsrm.pic.es
745343488       6277966848      n.a     lcg-gridka-se.fzk.de
1000000000000   500000000000    n.a     castorsrm.ific.uv.es
1000000000000   500000000000    n.a     castorgrid.ific.uv.es
66810000        6500000         n.a     cert-se-01.cnaf.infn.it
138020000       10260000        n.a     prep-se-01.pd.infn.it
4145003580      249174980       n.a     prod-se-01.pd.infn.it
5650364         2187160         n.a     zeus73.cyf-kr.edu.pl
1               1               n.a     dpm01.grid.sinica.edu.tw
275585848       35358272        n.a     gw38.hep.ph.ic.ac.uk
3840000000      1360000000      n.a     grid08.ph.gla.ac.uk
186770000       10090000        n.a     grid13.csl.ee.upatras.gr
3700000000      12440000000     n.a     castor.grid.sinica.edu.tw
58460000        3160000         n.a     epbf004.ph.bham.ac.uk

[...]
```

Example 5.1.1.3 (Listing the close Storage Elements)

The option `closeSE` will give an output as follows:

```
$ lcg-infosites --vo dteam closeSE

Name of the CE: lxb2039.cern.ch:2119/blah-pbs-dteam
Name of the close SE:   lxb2058.cern.ch

Name of the CE: grid06.ph.gla.ac.uk:2119/jobmanager-lcgpbs-dteam
Name of the close SE:   grid08.ph.gla.ac.uk

Name of the CE: lxb2090.cern.ch:2119/blah-lsf-grid_tests
Name of the close SE:   lxb2058.cern.ch

Name of the CE: cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps
Name of the close SE:   cert-se-01.cnaf.infn.it

Name of the CE: prep-ce-01.pd.infn.it:2119/blah-lsf-cert
Name of the close SE:   prod-se-01.pd.infn.it

Name of the CE: epbf005.ph.bham.ac.uk:2119/jobmanager-lcgpbs-dteam
Name of the close SE:   epbf004.ph.bham.ac.uk

Name of the CE: imalaydee.hep.ph.ic.ac.uk:2119/jobmanager-lcgpbs-dteam
Name of the close SE:   gw38.hep.ph.ic.ac.uk

Name of the CE: tb023.grid.sinica.edu.tw:2119/blah-pbs-dteam
Name of the close SE:   dpm01.grid.sinica.edu.tw

Name of the CE: tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-dteam
Name of the close SE:   dpm01.grid.sinica.edu.tw
Name of the close SE:   castor.grid.sinica.edu.tw


[...]
```

Example 5.1.1.4 (Listing local LFC servers)

In order to retrieve the names corresponding to the local LFC servers of a certain VO, use the command as follows:

```
lcg-infosites --vo atlas lfcLocal
lxb2038.cern.ch
pps-lfc.cnaf.infn.it
```

```
cclcglfcli03.in2p3.fr
[...]
```

5.1.2. lcg-info

The `lcg-info` command can be used to list either CEs or SEs satisfying a given set of conditions on their attributes, and to print, for each of them, the values of a given set of attributes. The information is taken from the BDII specified by the `LCG_GFAL_INFOSYS` environment variable or in the command line.

The general format of the command for listing CEs or SEs information is:

```
  $ lcg-info [--list-ce | --list-se] [--query <some_query>] [--attrs <some_attrs>]
```

where either `--list-ce` or `--list-se` must be used to indicate if CEs or SEs should be listed; the `--query` option introduces a filter (conditions to be accomplished) to the elements of the list, and the `--attrs` option may be used to specify which attributes to print. If `--list-ce` is specified, then only CE attributes are considered (others are just ignored), and the reverse is true for `--list-se`.

The attributes supported (which may be included with `--attrs` or within the `--query` expression) are only a subset of the attributes present in the GLUE schema, those that are most relevant for a user.

The `--vo` option can be used to restrict the query to CEs and SEs which support the given VO; it is mandatory when querying for attributes which are inherently referred to a VO, like `AvailableSpace` and `UsedSpace`.

Apart from the listing options, the `--help` option can be specified (alone) to obtain a detailed description of the command, and the `--list-attrs` option can be used to get a list of the supported attributes.

Example 5.1.2.1 (Get the list of supported attributes)

To have a list of the supported attributes, give:

```
  $ lcg-info --list-attrs
```

the output is similar to:

```
Attribute name  Glue object class     Glue attribute name

EstRespTime     GlueCE                GlueCEStateEstimatedResponseTime
WorstRespTime   GlueCE                GlueCEStateWorstResponseTime
TotalJobs       GlueCE                GlueCEStateTotalJobs
TotalCPUs       GlueCE                GlueCEInfoTotalCPUs
[...]
```

For each attribute, the simplified attribute name used by `lcg-info`, the corresponding object class and the attribute name in the GLUE schema are given.

***Example 5.1.2.2** (List all the Computing Elements in the BDII satisfying given conditions and print the desired attributes)*

You want to know how many jobs are running and how many free CPUs there are on CEs that have more an Athlon CPU and have Scientific Linux:

```
$ lcg-info --list-ce --query 'Processor=Athlon,OS=*SL*' --attrs 'RunningJobs,FreeCPUs'
```

The output could be:

```
- CE: lcgce.psn.ru:2119/jobmanager-lcgpbs-biomed
  - RunningJobs          30
  - FreeCPUs             0
```

It must be stressed that `lcg-info` only supports a logical AND of logical expressions, separated by commas, and the only allowed operator is $=$. In equality comparisons of strings, the $*$ matches any number of characters. Another useful query is the one to know which CEs have installed a particular version of an experiment's software. That would be something like:

```
$ lcg-info --vo cms --list-ce --attrs Tag --query 'Tag=*ORCA_8_7_1*'
```

***Example 5.1.2.3** (List all the Storage Elements in the BDII)*

Similarly, suppose that you want to know which CEs are close to each SE:

```
$ lcg-info --list-se --vo cms --attrs CloseCEs
```

the output will be like:

```
- SE: castorgrid.cern.ch
  - CloseCE               ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid
                          ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid_cms
                          hephygr.oeaw.ac.at:2119/jobmanager-torque-cms
                          ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid_lhcb
                          ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid_alice
                          ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid_atlas
                          ce01-slc3.cern.ch:2119/jobmanager-lcglsf-grid_dteam
                          hephygr.oeaw.ac.at:2119/jobmanager-torque-dteam
[...]
```

The `--bdii` option can be used to specify a particurar bdii (e.g. `--bdii exp-bdii.cern.ch:2170`), and the `--sed` option can be used to output the results of the query in a format easy to parse in a script, in which values for different attributes are separated by `%` and values of list attributes are separated by `&`.

5.1.3. The Local GRIS

The local GRISes running on Computing Elements and Storage Elements at the different sites report information on the characteristics and status of the services. They give both static and dynamic information.

In order to interrogate the GRIS on a specific Grid Element, the hostname of the Grid Element and the TCP port where the GRIS run must be specified. Such port is always `2135`. The following command can be used:

```
$ ldapsearch -x -h <hostname> -p 2135 -b "mds-vo-name=local, o=grid"
```

where the `-x` option indicates that simple authentication (instead of LDAP's SASL) should be used; the `-h` and `-p` options precede the hostname and port respectively; and the `-b` option is used to specify the initial entry for the search in the LDAP tree.

For a GRIS, the initial entry of the DIT is always `o=grid`, and the second one (next level) is `'mds-vo-name=local'`. It is in the entries in the deeper levels, that the actual resource information is shown. That is why `'mds-vo-name=local, o=grid'` is used as DN of the initial node for the search. For details, please refer to Appendix G.

The same effect can be obtained with:

```
$ ldapsearch -x -H <LDAP_URI> -b "mds-vo-name=local, o=grid"
```

where the hostname and port are included in the `-H <LDAP_URI>` option, avoiding the use of `-h` and `-p`.

Example 5.1.3.1 (Interrogating the GRIS on a Computing Element)

The command used to interrogate the GRIS located on host `lxb2006` is:

```
$ ldapsearch -x -h lxb2006.cern.ch -p 2135 -b "mds-vo-name=local, o=grid"
```

or:

```
$ ldapsearch -x -H ldap://lxb2006.cern.ch:2135 -b "mds-vo-name=local, o=grid"
```

And the obtained reply will be:

```
version: 2

#
# filter: (objectclass=*)
# requesting: ALL
#

# lxb2006.cern.ch:2119/jobmanager-lcgpbs-atlas, local, grid
dn: GlueCEUniqueID=lxb2006.cern.ch:2119/jobmanager-lcgpbs-atlas,mds-vo-name=lo
 cal,o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
objectClass: GlueCEInfo
objectClass: GlueCEPolicy
objectClass: GlueCEState
objectClass: GlueInformationService
objectClass: GlueKey
GlueCEHostingCluster: lxb2006.cern.ch
GlueCEName: atlas
GlueCEUniqueID: lxb2006.cern.ch:2119/jobmanager-lcgpbs-atlas
GlueCEInfoGatekeeperPort: 2119
GlueCEInfoHostName: lxb2006.cern.ch
GlueCEInfoLRMSType: torque
GlueCEInfoLRMSVersion: torque_1.0.1p5
GlueCEInfoTotalCPUs: 2
GlueCEInfoJobManager: lcgpbs
GlueCEInfoContactString: lxb2006.cern.ch:2119/jobmanager-lcgpbs-atlas
GlueCEInfoApplicationDir: /opt/exp_soft
GlueCEInfoDataDir: unset
GlueCEInfoDefaultSE: lxb2058.cern.ch
GlueCEStateEstimatedResponseTime: 2146660842
GlueCEStateFreeCPUs: 2
GlueCEStateRunningJobs: 0
GlueCEStateStatus: Production
GlueCEStateTotalJobs: 0
GlueCEStateWaitingJobs: 4444
GlueCEStateWorstResponseTime: 2146660842
GlueCEStateFreeJobSlots: 0
GlueCEPolicyMaxCPUTime: 2880
GlueCEPolicyMaxRunningJobs: 0
GlueCEPolicyMaxTotalJobs: 0
GlueCEPolicyMaxWallClockTime: 4320
GlueCEPolicyPriority: 1
GlueCEPolicyAssignedJobSlots: 0
GlueCEAccessControlBaseRule: VO:atlas
GlueForeignKey: GlueClusterUniqueID=lxb2006.cern.ch
GlueInformationServiceURL: ldap://lxb2006.cern.ch:2135/mds-vo-name=local,o=gri
```

```
 d
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 2

# lxb2006.cern.ch:2119/jobmanager-lcgpbs-alice, local, grid
dn: GlueCEUniqueID=lxb2006.cern.ch:2119/jobmanager-lcgpbs-alice,mds-vo-name=lo
 cal,o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
objectClass: GlueCEInfo
objectClass: GlueCEPolicy
objectClass: GlueCEState
objectClass: GlueInformationService
objectClass: GlueKey
GlueCEHostingCluster: lxb2006.cern.ch
GlueCEName: alice
GlueCEUniqueID: lxb2006.cern.ch:2119/jobmanager-lcgpbs-alice
GlueCEInfoGatekeeperPort: 2119
GlueCEInfoHostName: lxb2006.cern.ch
GlueCEInfoLRMSType: torque
GlueCEInfoLRMSVersion: torque_1.0.1p5
GlueCEInfoTotalCPUs: 2
GlueCEInfoJobManager: lcgpbs
GlueCEInfoContactString: lxb2006.cern.ch:2119/jobmanager-lcgpbs-alice
GlueCEInfoApplicationDir: /opt/exp_soft
GlueCEInfoDataDir: unset
GlueCEInfoDefaultSE: lxb2058.cern.ch
GlueCEStateEstimatedResponseTime: 2146660842
GlueCEStateFreeCPUs: 2
GlueCEStateRunningJobs: 0
GlueCEStateStatus: Production

[...]
```

In order to restrict the search, a filter of the form `attribute operator value` can be used. The operator is one of the defined in the following table:

Operator	Description
=	Entries whose attribute is equal to the value
>=	Entries whose attribute is greater than or equal to the value
<=	Entries whose attribute is less than or equal to the value
=*	Entries that have a value set for that attribute
˜=	Entries whose attribute value approximately matches the specified value

Furthermore, complex search filters can be formed by using boolean operators to combine constraints. The boolean operators that can be used are ”AND” (`&`), ”OR” (`|`) and ”NOT” (`!`). The syntax for that is the following:

```
( "&" or "|" or "!" (filter1) [(filter2) ...] )
```

Example of search filters are:

```
(& (Name=Smith) (Age>=32))
(! (GlueHostMainMemoryRAMSize<=1000))
```

In LDAP, a special attribute `objectClass` is defined for each directory entry. It indicates which object classes are defined for that entry in the LDAP schema. This makes it possible to filter entries that contain a certain object class. The filter for this case would be:

```
'objectclass=<name>'
```

Apart from filtering the search, a list of attribute names can be specified, in order to limit the values returned. As shown in the next example, only the value of the specified attributes will be returned.

A description of all objectclasses and their attributes to optimize the LDAP search command can be found in Appendix G.

Example 5.1.3.2 (Getting information about the site name from the GRIS on a CE)

```
$ ldapsearch -x -h lxb2006.cern.ch -p 2135 -b "mds-vo-name=local, o=grid" \
'objectclass=GlueTop' GlueSiteDescription'
version: 2

#
# filter: objectclass=GlueTop
# requesting: GlueSiteDescription
#

# CERN_PPS, local, grid
dn: GlueSiteUniqueID=CERN_PPS,mds-vo-name=local,o=grid
GlueSiteDescription: LCG Site

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

By adding the `-LLL` option, we can avoid the comments and the version information in the reply.

```
$ ldapsearch -LLL -x -h lxn1181.cern.ch -p 2135 -b "mds-vo-name=local,o=grid" \
objectclass=GlueTop' GlueSiteDescription
dn: GlueSiteUniqueID=CERN_PPS,mds-vo-name=local,o=grid
GlueSiteDescription: LCG Site
```

5.1.4. The Site GIIS/BDII

At each site, a site GIIS or BDII collects information about all resources present at a site (i.e. data from all GRISes of the site). Site BDIIs are preferred to site GIISes and are the default in gLite 3.0 releases. In this section we explain how to query a site GIIS/BDII.

For a list of all sites and all resources present, please refer to the GOC database.

Usually a site GIIS/BDII runs on a Computing Element. The port used to interrogate a site GIIS is usually the same as that of GRISes: `2135`. In order to interrogate the GIIS (and not the local GRIS) a different base name must be used (instead of `mds-vo-name=local, o=grid`), and one formed basing on the site name is generally used. For the site BDII, the port is different: `2170`, but the base name is also of the same format of site GIISes.

The complete contact string for a site GIIS is published in the GOC page. So if for example you have a look to the following URL:

https://goc.grid-support.ac.uk/gridsite/db/index.php?siteSelect=INFN-CNAF

You will retrieve the information shown in Figure 7, the GIIS URL is `ldap://gridit-ce-001.cnaf.infn.it:2135/mds-vo-name=infn-cnaf,o=grid`. In this case, the site still has a site GIIS. In order to interrogate it, we can use the command shown in the following example:

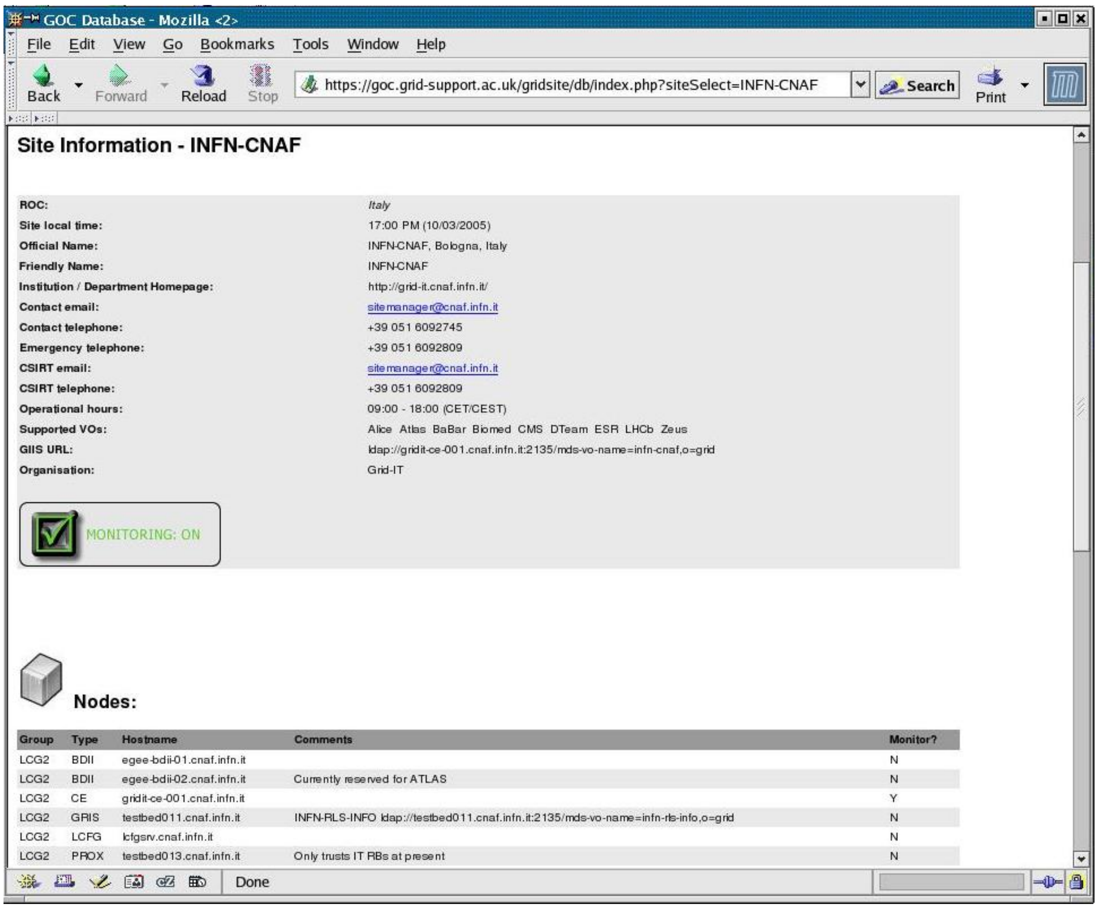
Example 5.1.4.1 (Interrogating the site BDII)

```
$ ldapsearch -x -H ldap://cert-ce-03.cnaf.infn.it:2170 -b "o=grid"
ersion: 2

#
# filter: (objectclass=*)
# requesting: ALL
#

# grid
dn: o=grid
objectClass: GlueTop

# resource, grid
dn: mds-vo-name=resource,o=grid
objectClass: GlueTop
```


GOC Database - Mozilla <2>
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
https://goc.grid-support.ac.uk/gridsite/db/index.php?siteSelect=INFN-CNAF
Search
Print
Done

Site Information - INFN-CNAF

ROC:	*Italy*
Site local time:	17:00 PM (10/03/2005)
Official Name:	INFN-CNAF, Bologna, Italy
Friendly Name:	INFN-CNAF
Institution / Department Homepage:	http://grid-it.cnaf.infn.it/
Contact email:	sitemanager@cnaf.infn.it
Contact telephone:	+39 051 6092745
Emergency telephone:	+39 051 6092809
CSIRT email:	sitemanager@cnaf.infn.it
CSIRT telephone:	+39 051 6092809
Operational hours:	09:00 - 18:00 (CET/CEST)
Supported VOs:	Alice Atlas BaBar Biomed CMS DTeam ESR LHCb Zeus
GIIS URL:	ldap://gridit-ce-001.cnaf.infn.it:2135/mds-vo-name=infn-cnaf,o=grid
Organisation:	Grid-IT

MONITORING: ON

Nodes:

Group	Type	Hostname	Comments	Monitor?
LCG2	BDII	egee-bdii-01.cnaf.infn.it		N
LCG2	BDII	egee-bdii-02.cnaf.infn.it	Currently reserved for ATLAS	N
LCG2	CE	gridit-ce-001.cnaf.infn.it		Y
LCG2	GRIS	testbed011.cnaf.infn.it	INFN-RLS-INFO ldap://testbed011.cnaf.infn.it:2135/mds-vo-name=infn-rls-info,o=grid	N
LCG2	LCFG	lcfgsrv.cnaf.infn.it		N
LCG2	PROX	testbed013.cnaf.infn.it	Only trusts IT RBs at present	N

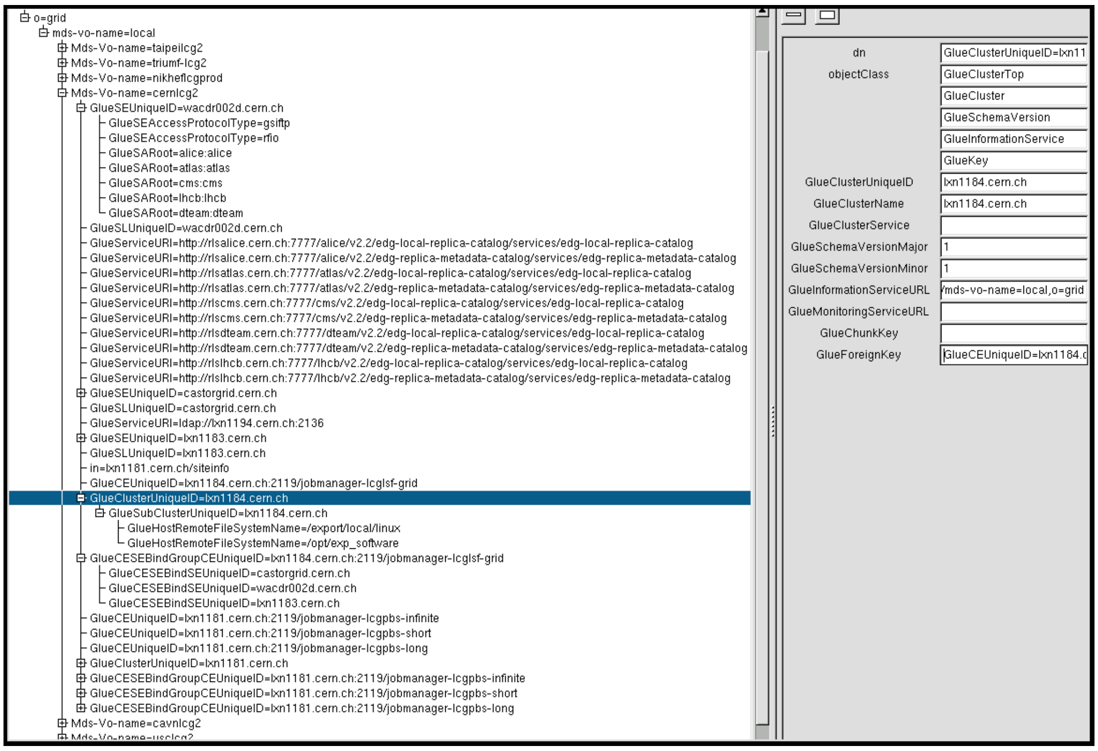
Figure 7: The status page of the INFN-CNAF site

```
# cert-ce-03.cnaf.infn.it, resource, grid
dn: GlueClusterUniqueID=cert-ce-03.cnaf.infn.it,mds-vo-name=resource,o=grid
objectClass: GlueClusterTop
objectClass: GlueCluster
objectClass: GlueSchemaVersion
objectClass: GlueInformationService
objectClass: GlueKey
GlueClusterName: cert-ce-03.cnaf.infn.it
GlueClusterService: cert-ce-03.cnaf.infn.it
GlueClusterUniqueID: cert-ce-03.cnaf.infn.it
GlueForeignKey: GlueCEUniqueID=cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps
GlueForeignKey: GlueCEUniqueID=cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps
GlueInformationServiceURL: ldap://cert-ce-03.cnaf.infn.it:2170/mds-vo-name=res
 ource,o=grid
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 2
```

```
# cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps, resource, grid
dn: GlueCEUniqueID=cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps,mds-vo-name=resou
 rce,o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
objectClass: GlueCEInfo
objectClass: GlueCEPolicy
```

5.1.5. The top BDII

A top BDII collects all information coming from site GIISes/BDIIes and stores them in a permanent database. The top BDII can be configured to get published information from resources in all sites, or only from some of them. In order to find out the location of a top BDII in a site, you can consult the GOC page of this site. The BDII will be listed with the rest of the nodes of the site (refer to Figure 7).



```
o=grid
  mds-vo-name=local
    Mds-Vo-name=taipeilcg2
    Mds-Vo-name=triumf-lcg2
    Mds-Vo-name=nikheflcgprod
    Mds-Vo-name=cernlcg2
      GlueSEUniqueID=wacdr002d.cern.ch
        GlueSEAccessProtocolType=gsiftp
        GlueSEAccessProtocolType=rfio
        GlueSARoot=alice:alice
        GlueSARoot=atlas:atlas
        GlueSARoot=cms:cms
        GlueSARoot=lhcb:lhcb
        GlueSARoot=dteam:dteam
      GlueSLUniqueID=wacdr002d.cern.ch
      GlueServiceURI=http://rlsalice.cern.ch:7777/alice/v2.2/edg-local-replica-catalog/services/edg-local-replica-catalog
      GlueServiceURI=http://rlsalice.cern.ch:7777/alice/v2.2/edg-replica-metadata-catalog/services/edg-replica-metadata-catalog
      GlueServiceURI=http://rlsatlas.cern.ch:7777/atlas/v2.2/edg-local-replica-catalog/services/edg-local-replica-catalog
      GlueServiceURI=http://rlsatlas.cern.ch:7777/atlas/v2.2/edg-replica-metadata-catalog/services/edg-replica-metadata-catalog
      GlueServiceURI=http://rlscms.cern.ch:7777/cms/v2.2/edg-local-replica-catalog/services/edg-local-replica-catalog
      GlueServiceURI=http://rlscms.cern.ch:7777/cms/v2.2/edg-replica-metadata-catalog/services/edg-replica-metadata-catalog
      GlueServiceURI=http://rlsdteam.cern.ch:7777/dteam/v2.2/edg-local-replica-catalog/services/edg-local-replica-catalog
      GlueServiceURI=http://rlsdteam.cern.ch:7777/dteam/v2.2/edg-replica-metadata-catalog/services/edg-replica-metadata-catalog
      GlueServiceURI=http://rlslhcb.cern.ch:7777/lhcb/v2.2/edg-local-replica-catalog/services/edg-local-replica-catalog
      GlueServiceURI=http://rlslhcb.cern.ch:7777/lhcb/v2.2/edg-replica-metadata-catalog/services/edg-replica-metadata-catalog
      GlueSEUniqueID=castorgrid.cern.ch
      GlueSLUniqueID=castorgrid.cern.ch
      GlueServiceURI=ldap://lxn1194.cern.ch:2136
      GlueSEUniqueID=lxn1183.cern.ch
      GlueSLUniqueID=lxn1183.cern.ch
      in=lxn1181.cern.ch/siteinfo
      GlueCEUniqueID=lxn1184.cern.ch:2119/jobmanager-lcglsf-grid
      GlueClusterUniqueID=lxn1184.cern.ch
        GlueSubClusterUniqueID=lxn1184.cern.ch
          GlueHostRemoteFileSystemName=/export/local/linux
          GlueHostRemoteFileSystemName=/opt/exp_software
      GlueCESEBindGroupCEUniqueID=lxn1184.cern.ch:2119/jobmanager-lcglsf-grid
        GlueCESEBindSEUniqueID=castorgrid.cern.ch
        GlueCESEBindSEUniqueID=wacdr002d.cern.ch
        GlueCESEBindSEUniqueID=lxn1183.cern.ch
      GlueCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-infinite
      GlueCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-short
      GlueCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-long
      GlueClusterUniqueID=lxn1181.cern.ch
      GlueCESEBindGroupCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-infinite
      GlueCESEBindGroupCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-short
      GlueCESEBindGroupCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-long
    Mds-Vo-name=cavnlcg2
    Mds-Vo-name=usclcg2
```

dn	GlueClusterUniqueID=lxn11
objectClass	GlueClusterTop
	GlueCluster
	GlueSchemaVersion
	GlueInformationService
	GlueKey
GlueClusterUniqueID	lxn1184.cern.ch
GlueClusterName	lxn1184.cern.ch
GlueClusterService	
GlueSchemaVersionMajor	1
GlueSchemaVersionMinor	1
GlueInformationServiceURL	/mds-vo-name=local,o=grid
GlueMonitoringServiceURL	
GlueChunkKey	
GlueForeignKey	GlueCEUniqueID=lxn1184.c

Figure 8: The LDAP directory of an gLite 3.0 BDII

The BDII can be interrogated using the same base name as in the case of the GRIS (`mds-vo-name=local,o=grid`), but using the BDII port: `2170`. The sub-tree corresponding to a particular site appears under an entry with a DN like the following:

```
Mds-Vo-name=<sitename>,mds-vo-name=local,o=grid
```

In Figure 8, a view of the DIT of the BDII of the gLite 3.0 is shown. In the figure, only the sub-tree that corresponds to the CERN site is expanded. The DN for every entry in the DIT is shown. Entries for storage and computing resources, as well as for the bindings between CEs and SEs can be seen in the figure.

Each entry can contain attributes from different object classes. This can be seen in the entry with DN `GlueClusteringUniqueID=lxn1184.cern.ch,Mds-Vo-name=cernlcg2,mds-vo-name=local,o=grid`, which is highlighted in the figure. This entry contains several attributes from the object classes `GlueClusterTop`, `GlueCluster`, `GlueSchemaVersion`, `GlueInformationService` and `GlueKey`.

In the right-hand side of the window, the DN of the selected entry and the name and value (in the cases, where it exists) of the attributes for this entry are shown. Notice how the special `objectclass` attribute gives information about all the object classes that are applied to this entry.

As seen, a graphical tool can be quite useful to examine the structure (and, certainly, the details also) of the Information Service LDAP directory. In addition, the schema (object classes, attributes...) can be also examined.

Example 5.1.5.1 (Interrogating a BDII)

In this example, a query is sent to the BDII in order to retrieve two attributes from the `GlueCESEBind` object class for all sites.

```
$ ldapsearch -x -LLL -H ldap://lxn1187.cern.ch:2170 -b "o=grid" \
'objectclass=GlueCESEBind' GlueCESEBindCEUniqueID GlueCESEBindSEUniqueID

dn: GlueCESEBindSEUniqueID=castor.grid.sinica.edu.tw,GlueCESEBindGroupCEUnique
 ID=tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-atlas,mds-vo-name=resource
 ,mds-vo-name=Taiwan-PPS,mds-vo-name=local,o=grid
GlueCESEBindSEUniqueID: castor.grid.sinica.edu.tw
GlueCESEBindCEUniqueID: tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-atlas

dn: GlueCESEBindSEUniqueID=castor.grid.sinica.edu.tw,GlueCESEBindGroupCEUnique
 ID=tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-dteam,mds-vo-name=resource
 ,mds-vo-name=Taiwan-PPS,mds-vo-name=local,o=grid
GlueCESEBindSEUniqueID: castor.grid.sinica.edu.tw
GlueCESEBindCEUniqueID: tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-dteam

dn: GlueCESEBindSEUniqueID=dpm01.grid.sinica.edu.tw,GlueCESEBindGroupCEUniqueI
 D=tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-biomed,mds-vo-name=resource
 ,mds-vo-name=Taiwan-PPS,mds-vo-name=local,o=grid
GlueCESEBindSEUniqueID: dpm01.grid.sinica.edu.tw
```

```
GlueCESEBindCEUniqueID: tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-biomed

dn: GlueCESEBindSEUniqueID=castor.grid.sinica.edu.tw,GlueCESEBindGroupCEUnique
 ID=tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-biomed,mds-vo-name=resourc
 e,mds-vo-name=Taiwan-PPS,mds-vo-name=local,o=grid
GlueCESEBindSEUniqueID: castor.grid.sinica.edu.tw
GlueCESEBindCEUniqueID: tb009.grid.sinica.edu.tw:2119/jobmanager-lcgpbs-biomed

[...]
```

Example 5.1.5.2 ***(Listing all the CEs which publish a given tag querying the BDII)***

The attribute `GlueHostApplicationSoftwareRunTimeEnvironment` can be used to publish experiment-specific information (***tag***) on a CE, for example that a given experiment software is installed. To list all the CEs which publish a given tag, a query to the BDII can be performed. In this example, that information is retrieved for all the subclusters:

```
$ ldapsearch -h lxn1187.cern.ch -p 2170 -b "o=grid" \
-x 'objectclass=GlueSubCluster' GlueChunkKey GlueHostApplicationSoftwareRunTimeEnvironment
```

Example 5.1.5.3 ***(Listing all the SEs which support a given VO)***

A Storage Element *supports* a VO if users of that VO are allowed to store files on that SE. It is possible to find out which SEs support a VO with a query to the BDII. For example, to have the list of all SEs supporting ATLAS, together with the storage space available in each of them, the `GlueSAAccessControlBaseRule`, which specifies a supported VO, is used:

```
$ ldapsearch -LLL -h  lxn1187.cern.ch -p 2170 -b \
"mds-vo-name=local,o=grid" -x "GlueSAAccessControlBaseRule=alice" \
GlueChunkKey GlueSAStateAvailableSpace GlueSAStateUsedSpace
```

And the obtained result will be something like the following:

```
dn: GlueSALocalID=alice,GlueSEUniqueID=gw38.hep.ph.ic.ac.uk,mds-vo-name=UKI-LT
 2-IC-HEP-PPS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 275474688
GlueSAStateUsedSpace: 35469432
GlueChunkKey: GlueSEUniqueID=gw38.hep.ph.ic.ac.uk

dn: GlueSALocalID=alice,GlueSEUniqueID=grid08.ph.gla.ac.uk,mds-vo-name=UKI-Sco
 tGrid-Gla-PPS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 3840000000
```

```
GlueSAStateUsedSpace: 1360000000
GlueChunkKey: GlueSEUniqueID=grid08.ph.gla.ac.uk

dn: GlueSALocalID=alice,GlueSEUniqueID=grid13.csl.ee.upatras.gr,mds-vo-name=Pr
 eGR-02-UPATRAS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 186770000
GlueSAStateUsedSpace: 10090000
GlueChunkKey: GlueSEUniqueID=grid13.csl.ee.upatras.gr

dn: GlueSALocalID=alice:alice,GlueSEUniqueID=castor.grid.sinica.edu.tw,mds-vo-
 name=Taiwan-PPS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 3700000000
GlueSAStateUsedSpace: 12440000000
GlueChunkKey: GlueSEUniqueID=castor.grid.sinica.edu.tw

dn: GlueSALocalID=alice,GlueSEUniqueID=epbf004.ph.bham.ac.uk,mds-vo-name=UKI-S
 OUTHGRID-BHAM-PPS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 58460000
GlueSAStateUsedSpace: 3160000
GlueChunkKey: GlueSEUniqueID=epbf004.ph.bham.ac.uk
[...]
```

5.2. R-GMA

As explained in section 3.2.5, R-GMA is an alternative information system to MDS. The standard Glue information is published in R-GMA, together with various monitoring data, and the system is also available for users to publish their own data. The system can be used via a command-line interface or APIs for C, C++, python and java, and for queries there is also a web browser interface. Several applications already use R-GMA, especially for accounting and monitoring purposes.

This section gives a brief overview of R-GMA, but for more information see [21].

5.2.1. R-GMA concepts

From a user point of view, R-GMA is very similar to a standard relational database. Data are organised in relational tables, and inserted and queried with SQL-style INSERT and SELECT statements (the allowed syntax is a subset of SQL, but reasonably complete for most purposes). However, there are some differences to bear in mind. The most basic is that a standard relational database can only have one row (tuple) with a given primary key value, but R-GMA usually has more than one. Related to this is the fact that R-GMA supports three different query types. Each tuple has a timestamp, and for a given primary key value you can query the most recent tuple (***Latest query***), a history of all tuples within some defined retention period (***History query***), or ask for tuples to be streamed to you as they are published (***Continuous query***). Continuous queries can also return a limited amount of historical ("old") data.

There are also some differences depending on how and where the data are stored. Each site has an R-GMA server which deals with all R-GMA interaction from clients on that site. The servers store data published from local clients (known as ***primary producers***), and may also collect data from other sites and re-publish it (***secondary producers***). Generally speaking, primary producers answer Continuous queries and secondary producers answer Latest and History queries; the latter query types are only supported if someone has created a secondary producer for the table(s) concerned (this is normally the case for standard tables, e.g. Glue). The data may be stored either in memory or in a real database, and some queries, notably joins, are only possible if all the required data can be found in a single real database. Such producers are known as ***archivers***.

The local R-GMA servers store all the data and deal with all the client interactions, so in this sense R-GMA is a distributed system. However, there is also a central server known as the ***Registry***, which holds the schema (the definitions of all the tables), and has lists of all consumers and producers to allow them to find each other. At present the Registry is a unique service in the Grid.

Users are free to create and use their own tables. However, at present there is only a single namespace for tables, so users should try to choose distinctive table names, e.g. prefixed with your VO or application name. There is a standard table called userTable which can be used for simple tests.

R-GMA is a secure service to the extent that you need a valid proxy to use it (or a valid certificate in your web browser). However, there is currently no authorisation control, so anyone can read and publish to any table. This is expected to be added in future releases.

5.2.2. The R-GMA Browser

The ***R-GMA browser*** is usually installed on each R-GMA server. It allows the user to easily navigate the schema (to see what tables are available and how they are defined), see all available producers for a table and query the (selected) producers. All this can be achieved using a web interface.

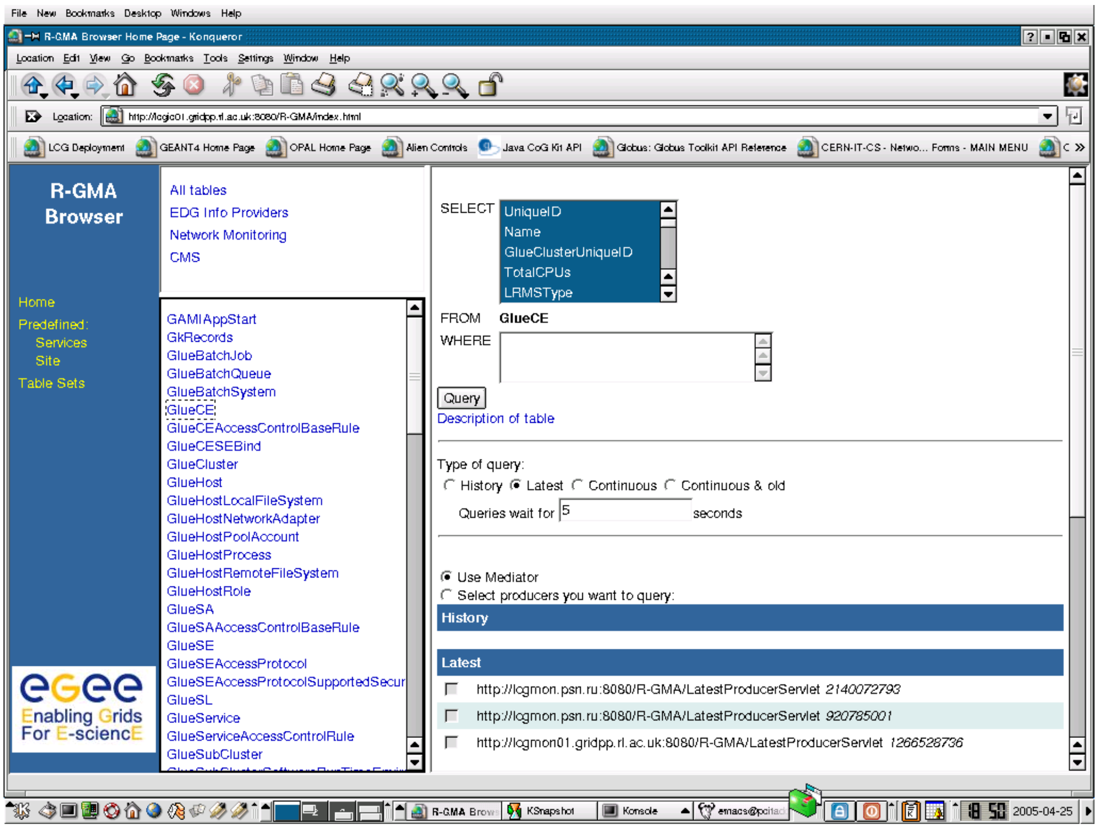
Figure 9 shows this R-GMA browser web interface. It is accessible via the following URL:

```
https://lcgmon01.gridpp.rl.ac.uk:8443/R-GMA/index.html
```

You can replace the hostname with the name of your local server to get a better response. In the left-hand bar you have a list of predefined tables to query; selecting one of them will give a drop-down list of matching items you can select, or you can just hit the Query button to see everything.

Alternatively, selecting the “Table Sets” link gives a complete list of all tables in the schema. Clicking on a table name gives a page where you can query the table definition, enter an SQL query, select the query type, and see and select from a list of all producers for that table. If you simply hit the Query button you get a Latest query for all data in the table, which corresponds to the intuitive idea of the current content of the table.

The best way to get some understanding of R-GMA is to play with the query interface for one of the standard tables, e.g. GlueSite, as the interface is reasonably intuitive. The browser is read-only so you can’t do any damage.



SELECT UniqueID, Name, GlueClusterUniqueID, TotalCPUs, LRMSType

FROM GlueCE

WHERE

Type of query: History [✓ Latest] Continuous Continuous & old

Queries wait for 5 seconds

[✓ Use Mediator]
Select producers you want to query:

http://lcgmon.psn.ru:8080/R-GMA/LatestProducerServlet *2140072793*
http://lcgmon.psn.ru:8080/R-GMA/LatestProducerServlet *920785001*
http://lcgmon01.gridpp.rl.ac.uk:8080/R-GMA/LatestProducerServlet *1266528736*

Figure 9: The R-GMA Web Interface

5.2.3. The R-GMA CLI

An R-GMA CLI is available on every UI and WN. This interface allows the user to perform queries and also to publish new information. It includes a consumer and can initiate both primary and secondary producers, although it does not provide all the detailed options available in the APIs.

The user can interact with the CLI directly from the command line by using the `-c` option:

```
rgma -c ``select Web from GlueSite where UniqueId='lcgmon01.gridpp.rl.ac.uk'''

+--------------------------------+
| Web                            |
+--------------------------------+
| http://www.gridpp.ac.uk/tier1a/ |
+--------------------------------+
1 rows
```

If you simply type `rgma` an interactive shell is started:

```
Welcome to the R-GMA virtual database for Virtual Organisations.
================================================================

Your local R-GMA server is:

  https://lcgmon01.gridpp.rl.ac.uk:8443/R-GMA

You are connected to the following R-GMA Registry services:

  https://lcgic01.gridpp.rl.ac.uk:8443/R-GMA/RegistryServlet

You are connected to the following R-GMA Schema service:

  https://lcgic01.gridpp.rl.ac.uk:8443/R-GMA/SchemaServlet

Type ``help'' for a list of commands.

rgma> select Web from GlueSite where UniqueId='lcgmon01.gridpp.rl.ac.uk'

+----------------------------------+
| Web                              |
+----------------------------------+
| http://www.gridpp.ac.uk/tier1a/ |
+----------------------------------+
1 rows
rgma>
```

As shown, the CLI reports the location of the registry, which holds pointers to all the R-GMA producers for all sites and VOs. Queries will collect information from the appropiate producers wherever they are located.

The syntax of all the commands available in the R-GMA interface can be obtained using the `help` command to get a list of the supported commands, and typing `help <command>` to get information on a particular command. A list of the most important commands is as follows:

Command	Description
`help [<command>]`	Display information (general or about a specific command)
`exit / quit / CTRL-D`	Exit the R-GMA command line shell
`show [tables | producers of <table>]`	Show the tables in the schema, or the current producers for a given table
`describe <table>`	Show the column names and types for the specified table
`select`	Query R-GMA (SQL syntax)
`set query latest | continuous | history`	Set the type of subsequent queries
`insert`	Insert a tuple into a primary producer (SQL syntax)
`secondaryproducer <table>`	Declare a table to be consumed and republished by a secondary producer
`set [secondary]producer latest | continuous | history`	Set the supported query type for the primary or secondary producer
`set [timeout | maxage] <timeout> [<units>]`	Set the timeout for queries or the maximum age of tuples to return

A simple example of how to query the R-GMA virtual database follows.

Example 5.2.3.1 (Querying the R-GMA Information System)

Inside the interface you can easily perform any query using SQL syntax:

```
rgma> set query continuous
Set query type to continuous
rgma> set timeout 120 seconds
Set timeout to 120 seconds
rgma> select UniqueID, TotalCPUs from GlueCE

+-----------------------------------------------------+-----------+
| UniqueID                                            | TotalCPUs |
+-----------------------------------------------------+-----------+
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-atlas  | 498       |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-dteam  | 498       |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-lhcb   | 498       |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-babar  | 498       |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-lhcb      | 68        |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-cms       | 68        |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-atlas    | 68         |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-lhcb     | 68         |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-cms      | 68         |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-atlas    | 68         |
| grid012.ct.infn.it:2119/jobmanager-lcglsf-alice    | 174        |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-lhcb     | 68         |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-cms      | 68         |
| grid001.fi.infn.it:2119/jobmanager-lcgpbs-atlas    | 68         |
| grid012.ct.infn.it:2119/jobmanager-lcglsf-infinite | 174        |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-atlas | 498        |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-dteam | 498        |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-lhcb  | 498        |
| hepgrid2.ph.liv.ac.uk:2119/jobmanager-lcgpbs-babar | 498        |
+----------------------------------------------------+------------+
19 rows
```

In this example, we first set the type of query to continuous. That is, new tuples are received as they are published, and the query will not terminate unless the user aborts or a maximum time for the query is reached. This timeout is then defined as 120 seconds. Finally, we query for the ID and the number of CPUs of all CEs publishing information into R-GMA in the two minutes following the query.

5.2.4. R-GMA APIs

There exist R-GMA APIs in Java, C, C++ and Python. They include methods for creating consumers, as well as primary and secondary producers; setting the types of queries and of producers, retention periods and time outs; retrieving tuples, and inserting data. The APIs are beyond the scope of this introduction, but detailed documentation exists for all APIs, including example code [21].

5.3. MONITORING

The ability to monitor resource related parameters is currently considered a necessary functionality in any network. In such an heterogeneous and complex system as the Grid, this necessity becomes fundamental. A proper monitoring system permits the existence of a central point of operational information (i.e.: in gLite 3.0, the GOC). The monitoring system should be able to collect data from the resources in the system, in order to analyze usage, behavior and performance of the Grid, detect and notify fault situations, contract violations and user-defined events.

The GOC web page contains a whole section containing monitoring information for gLite 3.0. Apart from R-GMA that was explained previously, several different monitoring tools can be used, including general purpose monitoring tools and Grid specific systems like GridICE [29].

Also important are the web pages publishing the results of functional tests applied periodically to the all the sites registered within gLite 3.0. The results of this tests show if a site is responding correctly to standard Grid operations; otherwise, an investigation on the cause of the unexpected results is undertaken. Some VOs may even decide to automatically exclude from their BDII the sites that are not passing the functional tests successfully, so that they do not appear in the IS and are not considered for possible use by their applications.

Note: Please do not report problems occurring with a site if this site is marked as ill in the test reports. If that is the case, the site is already aware of the problem and working to solve.

The results of some sets of functional sites can be checked in the following URLs:

https://lcg-sft.cern.ch/sft-l-pps/lastreport.cgi

https://lcg-sft.cern.ch/sft-pps/lastreport.cgi

http://goc.grid.sinica.edu.tw/gstat/

In the following section, as an example of monitoring system, the GridICE service is reviewed.

5.3.1. GridICE

The GridICE monitoring service is structured in a five layer architecture. The resource information is obtained from the gLite 3.0 Information Service, namely MDS. The information model for the retrieved data is an extended GLUE Schema, where some new objects and attributes have been added to the original model. Please refer to the documentation presented in [29] for details on this.

GridICE not only periodically retrieves the last information published in MDS, but also collects historical monitoring data in a persistent storage. This allows the observation of the evolution in time of the published data. In addition, GridICE will provide performance analysis, usage level and general reports and statistics, as well as the possibility to configure event detection and notification actions; though these two functionalities are still at an early development stage.

NOTE: All the information retrievable using GridICE (including the extensions of the GLUE schema) is also obtainable through R-GMA, by defining the proper archives. This represents an alternative way to get that information.

The GridICE web page that shows the monitoring information for gLite 3.0 is accessible at the following URL (also linked in the GOC web site):

http://gridice2.cnaf.infn.it:50080/gridice/site/site.php

In the initial page (site view) a summary of the current status of the computing and storing resources in a per site basis is presented. This includes the load of the site network, the number of jobs being run or waiting to be run, and the amount of total and available storage space in the site. If a particular site is selected, then several informations regarding each one of the services present in each of the nodes of the site are shown. The nodes are classified as Resource Brokers, CE access nodes or SE access nodes.

There are also other types of views: Geo, Gris and VO views. The Geo view presents a geographical representation of the Grid. The Gris view shows current and historical information about the status (on or off) of every node. Finally, the VO view holds the same information that the site view, but here nodes are classified in a per VO basis. The user can specify a VO name, and get the data about all the nodes that support it.

Finally, the job monitoring section of GridICE provides figures about the number of jobs of each VO that are running or are queued in each Grid site.

6. WORKLOAD MANAGEMENT

6.1. INTRODUCTION

The ***Workload Management (WMS)*** is the gLite 3.0 component that allows users to submit ***jobs***, and performs all tasks required to execute them, without exposing the user to the complexity of the Grid. It is the responsibility of the user to describe his jobs and their requirements, and to retrieve the output when the jobs are finished.

In the WLCG/EGEE Grid, two different workload management systems are deployed: the legacy LCG-2 system, derived from the EDG project, and the new system from the EGEE project, which is an evolution of the former and therefore has more functionalities.

In the following sections, we will describe the basic concepts of the language used to describe a job, the basic command line interface to submit and manage user jobs, the more advanced command line interface that allows to submit groups of jobs, some useful tools and how to use advanced job types.

6.2. JOB DESCRIPTION LANGUAGE

The ***Job Description Language (JDL)*** is a high-level language based on the ***Classified Advertisement (ClassAd) language*** [30], used to describe jobs and aggregates of jobs with arbitrary dependency relations. The JDL is used in WLCG/EGEE to specify the desired job characteristics and constraints, which are used by the match-making process to select the best resource to execute the job.

The fundamentals of the JDL are given in this section. A complete description of the JDL syntax is out of the scope of this guide, and can be found in [32]. The description of the JDL attributes for the EDG WMS is in [33], and for the gLite WMS is in [33][35].

A job description is a file (called ***JDL file***) consisting of lines having the format:

attribute = expression;

and terminated by a semicolon. Expressions can span several lines, but only the last one must be terminated by

a semicolon. Literal strings are enclosed in double quotes. If a string itself contains double quotes, they must be escaped with a backslash (e.g.: `Arguments = " \"hello\" 10"`). The character “ ‘ ” cannot be specified in the JDL.

Comments must be preceded by a sharp character (#) or a double slash (//) at the beginning if each line. Multi-line comments must be enclosed between “/*” and “*/” .

ATTENTION! The JDL is sensitive to blank characters and tabs. No blank characters or tabs should follow the semicolon at the end of a line.

Example 6.2.1 ***(Define a simple "Hello world!" job)***

To define a job which runs the `hostname` command on the WN, write a JDL like this:

```
Executable = "/bin/hostname";
StdOutput = "std.out";
StdError = "std.err";
```

The `Executable` attribute specifies the command to be run by the job. If the command is already present on the WN, it will be expressed as a absolute path; if it has been copied from the UI, only the file name must be specified, and the path of the command on the UI should be given in the `InputSandbox` attribute (see below). For example:

```
Executable = "test.sh";
InputSandbox = {"/home/doe/test.sh"};
StdOutput = "std.out";
StdError = "std.err";
```

The `Arguments` can contain a string value, which is taken as argument list for the executable:

```
Arguments = "fileA 10";
```

In the `Executable` and in the `Arguments` attributes it may be necessary to use special characters, such as &, \, |, >, <. If these characters should be escaped in the shell (for example, if they are part of a file name), they should be preceded by triple \ in the JDL, or specified inside quoted strings.

The attributes `StdOutput` and `StdError` define the name of the files containing the standard output and standard error of the executable, once the job output is retrieved.

For the standard input, an input file can be similarly specified (though this is not required):

```
StdInput = "std.in";
```

If some files have to be copied from the UI to the execution node, they can be listed in the `InputSandbox` attribute:

```
InputSandbox = {"test.sh", "fileA", "fileB", ...};
```

Only the file specified as `Executable` will have automatically the execution flag: if other files in the input sandbox have such flag on the UI, they will usually lose it when copied to the WN.

Finally, the files to be transferred back to the UI after the job is finished can be specified using the `OutputSandbox` attribute:

```
OutputSandbox = {"std.out","std.err"};
```

Wildcards are allowed only in the `InputSandbox` attribute. The list of files in the Input Sandbox is specified relatively to the current working directory. Absolute paths cannot be specified in the `OutputSandbox` attribute. The `InputSandbox` cannot contain two files with the same name, even if they have a different absolute path, as when transferred they would overwrite each other.

The environment of the job can be modified using the `Environment` attribute. For example:

```
Environment = {"CMS_PATH=$HOME/cms",
               "CMS_DB=$CMS_PATH/cmdb"};
```

The `VirtualOrganisation` attribute can be used to explicitly specify the VO of the user:

```
VirtualOrganisation = "cms";
```

but is superseded by the VO contained in the user proxy, if a VOMS proxy is used. For normal proxies, the VO can either be specified in the JDL, in the UI configuration files or as argument to the job submission command (see section 6.3.1).

Note: A common error is to write `VirtualOrganization`. It will not work.

To summarise, a typical JDL for a simple Grid job will look like:

```
Executable = "test.sh";
Arguments = "fileA fileB";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"test.sh", "fileA", "fileB"};
OutputSandbox = {"stdout", "std.err"};
```

where `test.sh` could be:

```
#!/bin/sh
echo "First file:"
cat $1
echo "Second file:"
cat $2
```

In section 6.3.1 it is explained how to submit such job.

Example 6.2.2 (Specifying requirements on the CE)

The `Requirements` attribute can be used to express constraints on the resources where the job should run. Its value is a Boolean expression that must evaluate to `true` for a job to run on that specific CE. For that purpose all the GLUE attributes of the IS can be used, by prepending the `other.` string to the attribute name. For a list of GLUE attributes, see Appendix G.

Note: Only one `Requirements` attribute can be specified (if there are more than one, only the last one is considered). If several conditions must be applied to the job, then they all must be included in a single `Requirements` attribute, using a boolean expression.

For example, let us suppose that the user wants to run on a CE using PBS as batch system, and whose WNs have at least two CPUs. He will write then in the job description file:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" && other.GlueCEInfoTotalCPUs > 1;
```

The WMS can be also asked to send a job to a particular CE with the following expression:

```
Requirements = other.GlueCEUniqueID == "lxshare0286.cern.ch:2119/jobmanager-pbs-short";
```

Note: As explained in 6.3.7, normally the condition that a CE is in production state is automatically added to the requirements clause. Thus, CEs that do not correctly publish this will not match. This condition is, nevertheless, configurable.

If the job must run on a CE where a particular experiment software is installed and this information is published by the CE, something like the following must be written:

```
Requirements = Member("CMSIM-133",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Note: The `Member` operator is used to test if its first argument (a scalar value) is a member of its second argument (a list). In this example, the `GlueHostApplicationSoftwareRunTimeEnvironment` attribute is a list.

Example 6.2.3 (Specifying requirements using wildcards)

It is also possible to use regular expressions when expressing a requirement. Let us suppose for example that the user wants all his jobs to run on any CE in the domain `cern.ch`. This can be achieved putting in the JDL file the following expression:

```
Requirements = RegExp("cern.ch", other.GlueCEUniqueId);
```

The opposite can be required by using:

```
Requirements = (!RegExp("cern.ch", other.GlueCEUniqueId));
```

Example 6.2.4 (Specifying requirements on a close SE)

In order to specify requirements on the SE "close" to the CE where the job will run, the RB uses a special match-making mechanism, called ***gang-matching***[34]. For example, to ensure that the job runs on a CE with at least 200 MB of free disk space on a close SE, the following JDL expression can be used:

```
Requirements = anyMatch(other.storage.CloseSEs,target.GlueSAStateAvailableSpace > 204800);
```

Example 6.2.5 (A complex requirement used in gLite 3.0)

The following example has been actually used by the Alice experiment in order to find a CE that has some software packages installed (`VO-alice-AliEn` and `VO-alice-ALICE-v4-01-Rev-01`), and that allows the job to run for more than 86,000 seconds (i.e., so that the job is not aborted before it has time to finish).

```
Requirements = other.GlueHostNetworkAdapterOutboundIP==true &&
Member("VO-alice-AliEn",other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
Member("VO-alice-ALICE-v4-01",other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
(other.GlueCEPolicyMaxWallClockTime > 86000 ) ;
```

Example 6.2.6 (Using the automatic resubmission)

It is possible to have the WMS automatically resubmitting jobs which, for some reason, are aborted by the Grid. Two kinds of resubmission are available for the gLite 3.0 WMS: the **deep resubmission** and the **shallow resubmission** (only the former is available in the LCG-2 WMS). The resubmission is deep when the job fails after it has started running on the WN, and shallow if it fails before the user executable starts.

The user can limit the number of times the WMS should resubmit a job by using the JDL attributes `RetryCount` and `ShallowRetryCount` for the deep and shallow resubmission respectively. For example, to disable the deep resubmission and limit the attempts of shallow resubmission to 3:

```
RetryCount = 0;
ShallowRetryCount = 3;
```

It is advisable to disable the deep resubmission, as in some conditions the WMS can lose track of a job even if it is actually running on the WN, which may cause problems if an identical job is started elsewhere by the system.

The values of the `MaxRetryCount` and `MaxShallowRetryCount` parameters in the WMS configuration file represent both the default and the maximum limits for the number of resubmissions.

Example 6.2.7 (Using the automatic proxy renewal)

The proxy renewal feature of the WMS is automatically enabled, as long as the user has stored a long proxy in the default MyProxy server (usually defined in the `MYPROXY_SERVER` environment variable). However it is possible to indicate to the WMS a different MyProxy server in the JDL file:

```
MyProxyServer = "myproxy.cern.ch";
```

The proxy renewal can be disabled altogether by adding to the JDL:

```
MyProxyServer = "";
```

Example 6.2.8 (Customizing the "goodness" of a CE)

The choice of the CE where to execute the job, among all the ones satisfying the requirements, is based on the ***rank*** of the CE; namely, a quantity expressed as a floating-point number. The CE with the highest rank is the one selected.

By default, the Rank is equal to `-other.GlueCEStateEstimatedResponseTime`, where the estimated response time is an estimation of the time interval between the job submission and the beginning of the job execution. However, the user can redefine the rank with the `Rank` attribute as a function of the CE attributes. For example:

```
Rank = other.GlueCEStateFreeCPUs;
```

which will rank best the CE with the most free CPUs. The next one is a more complex expression:

```
Rank = ( other.GlueCEStateWaitingJobs == 0 ? other.GlueCEStateFreeCPUs :
-other.GlueCEStateWaitingJobs);
```

In this case, the selected CE will be the one with the least waiting jobs, or the most free CPus, if there are no waiting jobs.

6.3. The Command Line Interface

In this section, all commands available for the user to manage jobs are described. For completeness, both the LCG and gLite CLI are described and more informations are given whenever the two differ considerably. For a more detailed information on all these topics, and on the different commands, please refer to [25] and [24].

6.3.1. Job Submission

To submit a job to the WLCG/EGEE Grid, the user must have a valid proxy certificate in the User Interface machine (as described in Chapter 4) and use the following command:

```
$ edg-job-submit <jdl_file>        (for the LCG WMS)
$ glite-job-submit <jdl_file>      (for the gLite WMS)
```

where `<jdl_file>` is a file containing the job description, usually with extension `.jdl`.

Example 6.3.1.1   (Submitting a simple job)

Create a file `test.jdl` with these contents:

```
Executable = "/bin/hostname";
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err"};
```

It describes a simple job that will execute `/bin/hostname`. Standard output and error are redirected to the files `std.out` and `std.err` respectively, which are then transferred back to the User Interface after the job is finished, as they are in the Output Sandbox. The job is submitted by issuing:

```
$ edg-job-submit test.jdl    (for the LCG WMS)
$ glite-job-submit test.jdl  (for the gLite WMS)
```

If the submission is successful, the output is similar to:

```
Selected Virtual Organisation name (from proxy certificate extension): atlas
Connecting to host egee-rb-01.mi.infn.it, port 7772
Logging to host egee-rb-01.mi.infn.it, port 9002


*********************************************************************************************
                               JOB SUBMIT OUTCOME
 The job has been successfully submitted to the Network Server.
 Use glite-job-status command to check job current status. Your job identifier is:

 - https://egee-rb-01.mi.infn.it:9000/LPHg_qAnRlPjXtWvQlc7QQ


*********************************************************************************************
```

In case of failure, an error message will be displayed instead, and an exit status different from zero will be returned.

The command returns to the user the job identifier (***jobId***), which defines uniquely the job and can be used to perform further operations on the job, like interrogating the system about its status, or canceling it. The format of the jobId is:

```
https://Lbserver_address[:port]/unique_string
```

where `unique_string` is guaranteed to be unique and `Lbserver_address` is the address of the Logging and Bookkeeping server for the job, and usually (but not necessarily) is also the gLite WMS (or LCG Resource Broker).

Note: the jobId does NOT identify a web page.

If the command returns the following error:

```
**** Error: API_NATIVE_ERROR ****
Error while calling the "NSClient::multi" native api
AuthenticationException: Failed to establish security context...

**** Error: UI_NO_NS_CONTACT ****
Unable to contact any Network Server
```

it means that there are authentication problems between the UI and the network server (check your proxy or have the site administrator check the certificate of the server).

Many options are available to edg-job-submit.

If the user's proxy does not have VOMS extensions, he can specify his virtual organization with the `--vo <vo_name>` option; otherwise the default VO specified in the standard configuration file (`$GLITE_WMS_LOCATION/etc/glite_wmsui_cmd_var.conf` for a gLite UI or `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` for a LCG UI) is used.

Note: The above mentioned configuration file can leave the default VO with a value of `"unspecified"`. In that case, if the `--vo` option is not used with `edg-job-submit` AND a proxy without VOMS extension is used, the command will return an error message. In case of a gLite UI:

```
**** Error: UI_NO_VOMS ****
Unable to determine a valid user's VO
```

The useful `-o <file_path>` option allows users to specify a file to which the jobId of the submitted job will be appended. This file can be given to other job management commands to perform operations on more than one job with a single command.

The `-r <CE_Id>` option is used to directly send a job to a particular CE. The drawback is that the match making functionality (see Section 6.3.3) will not be carried out. That is, the BrokerInfo file, which provides information about the evolution of the job, will not be created.

The CE is identified by `<CE_Id>`, which is a string with the following format:

```
<full_hostname>:<port_number>/jobmanager-<service>-<queue_name>     (for a LCG CE)
<full_hostname>:<port_number>/blah-<service>-<queue_name>     (for a gLite CE)
```

where `<full_hostname>` and `<port>` are the hostname of the machine and the port where the Globus Gatekeeper (for LCG CE) or CondorC+BLAH (gLite CE) is running (the Grid Gate), `<queue_name>` is the name of one of the available queue of jobs in that CE, and the `<service>` could refer to the LRMS, such as `lsf`, `pbs`, `condor`, but can also be a different string as it is freely set by the site administrator when the queue is set-up.

An example of CE Id is:

```
adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite      (for LCG CE)
prep-ce-01.pd.infn.it:2119/blah-lsf-atlas            (for gLite CE)
```

Similarly, the `-i <file_path>` allows users to specify a list of CEs from where the user will have to choose a target CE interactively.

Note: The LCG Resource Broker is able to submit jobs ONLY to LCG Computing Elements. The gLite WMS instead is capable to submit jobs to both the LCG and gLite CEs.

Lastly, the `--nomsgi` option makes the command display neither messages nor errors on the standard output. Only the `jobId` assigned to the job is printed to the user if the command was successful. Otherwise the location of the generated log file containing error messages is printed on the standard output. This option has been provided to make easier use of the `glite-job-submit` command inside scripts as an alternative to the `-o` option.

Example 6.3.1.2 (Listing Computing Elements that match a job description)

It is possible to see which CEs are eligible to run a job specified by a given JDL using the `glite-job-list-match` (for gLite WMS) or `edg-job-list-match` (for LCG RB). The `--rank` option can be used to display the ranking value of each matching resource.

```
$ glite-job-list-match --rank Hello.jdl

Selected Virtual Organisation name (from proxy certificate extension): atlas
Connecting to host cert-rb-01.cnaf.infn.it, port 7772

***************************************************************************
                         COMPUTING ELEMENT IDs LIST
 The following CE(s) matching your job requirements have been found:

                   *CEId*                                  *Rank*

 ce03.pic.es:2119/jobmanager-lcgpbs-pps                      -58
 prep-ce-01.pd.infn.it:2119/blah-lsf-atlas                   -3810
 cg01.ific.uv.es:2119/blah-pbs-atlas                    -9850
 ce02.pic.es:2119/blah-pbs-pps                          -999999
**********************************************************************
```

The `-o <file path>` option can be used to store the CE list on a file, which can later be used with the `-i <file path>` option of `glite-job-submit`.

6.3.2. Job Operations

After a job is submitted, it is possible to see its status and its history, and to retrieve logging information about it. Once the job is finished the job's output can be retrieved, although it is also possible to cancel it previously. The following examples explain how.

Example 6.3.2.1 (Retrieving the status of a job)

Given a submitted job whose job identifier is `<jobId>`, the command is:

```
$ edg-job-status <jobId>            (for the LCG RB)
$ glite-job-status <jobId>          (for the gLite WMS)
```

And an example of a possible output from the gLite LB is:

```
$ glite-job-status  https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA


*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA
Current Status:     Ready
Status Reason:      unavailable
Destination:        cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps
Submitted:          Mon May 15 15:14:55 2006 CEST
*************************************************************
```

where the current status of the job is shown, along with the time when that status was reached, and the reason for being in that state (which may be especially helpful for the ABORTED state). The possible states in which a job can be found were introduced in Section 3.3.1, and are summarized in Appendix C. Finally, the `destination` field contains the ID of the CE where the job has been submitted.

Much more information is provided if the verbosity level is increased by using `-v 1`, `-v 2` or `-v 3` with the command. See [25] for detailed information on each of the fields that are returned then.

Many job identifiers can be given as arguments of the `glite-job-status` command, i.e.:

```
glite-job-status <jobId1> ... <jobIdN>
```

The option `-i <file path>` can be used to specify a file with a list of job identifiers (saved previously with the `-o` option of `glite-job-submit`). In this case, the command asks the user interactively the status of which job(s) should be printed. Subsets of jobs can be selected (e.g. 1-2,4).

```
$ glite-job-status -i jobs.list

-------------------------------------------------------------------
1 : https://cert-rb-01.cnaf.infn.it:9000/ma46vg-cgV2SzdkmCI-CTw
2 : https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA
a : all
q : quit
-------------------------------------------------------------------

Choose one or more jobId(s) in the list - [1-2]all:
```

If the `--all` option is used instead, the status of all the jobs owned by the user submitting the command is retrieved. As the number of jobs owned by a single user may be large, there are some options that limit that job selection. The `--from` / `--to [MM:DD:]hh:mm[:[CC]YY]` options make the command query LB for jobs that were submitted after/before the specified date and time. The `--status <state>` (`-s`) option makes the command retrieve only the jobs that are in the specified state, and the `--exclude <state>` (`-e`) option makes it retrieve jobs that are not in the specified state. This two lasts options are mutually exclusive, although they can be used with `--from` and `--to`.

In the following examples, the first command retrieves all jobs of the user that are in the state DONE or RUNNING, and the second retrieves all jobs that were submitted before the 17:35 of the current day, and that were not in the CLEARED state.

```
$ glite-job-status --all -s DONE -s RUNNING
$ glite-job-status --all -e CLEARED --to 17:35
```

NOTE: for the `--all` option to work, it is necessary that an index by owner is created in the LB server; otherwise, the command will fail, since it will not be possible for the LB server to identify the user's jobs. Such index can only be created by the LB server administrator, as explained in section 5.2.2 of [25].

With the option `-o <file path>` the command output can be written to a file.

Example 6.3.2.2 (Canceling a job)

A job can be canceled before it ends using the command `glite-job-cancel` (for the gLite WMS) or `edg-job-cancel` (for the LCG RB). This command requires as arguments one or more job identifiers. For example:

```
$ glite-job-cancel  https://cert-rb-01.cnaf.infn.it:9000/-IQM-Vq20r9Rzgc5tUMWdg
```

```
Are you sure you want to remove specified job(s)? [y/n]n :y

=============================  glite-job-cancel Success  ==============================
 The cancellation request has been successfully submitted for the following job(s):

 - https://cert-rb-01.cnaf.infn.it:9000/-IQM-Vq20r9Rzgc5tUMWdg

=====================================================================================
```

All the command options work exactly as in `glite-job-status` and `edg-job-status`.

Note: If the job has not reached the CE yet (i.e.: its status is WAITING or READY), the cancellation request could be ignored, and the job may continue running, although a message of successful cancellation is returned to the user. In such cases, just cancel the job again when its status is SCHEDULED or RUNNING.

Example 6.3.2.3 (Retrieving the output of a job)

After the job has finished (it reaches the DONE status), its output can be copied to the UI with the command `glite-job-output` (for the gLite WMS) or `edg-job-get-output` (for the LCG RB), which takes a list of jobs as argument. For example:

```
$ glite-job-output  https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA

Retrieving files from host: cert-rb-01.cnaf.infn.it ( for https:... )

*********************************************************************************
                        JOB GET OUTPUT OUTCOME

 Output sandbox files for the job:
 - https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA
 have been successfully retrieved and stored in the directory:
 /tmp/scampana_55YfzeDigWeoHbpHxx1BQA

*********************************************************************************
```

By default, the output is stored under `/tmp`, but it is possible to specify in which directory to save the output using the `--dir <path_name>` option.

All command options work exactly as in `glite-job-status` and `edg-job-status`.

NOTE: The output of a job will in principle be removed from the RB after a certain period of time. How long this period is may vary depending on the administrator of the RB, but the currently suggested time is 10 days, so users should try always to retrieve their jobs within one week after job completion (to have a safe margin).

Example 6.3.2.4 (Retrieving logging information about submitted jobs)

The `glite-job-logging-info` (for gLite LB) and `edg-job-get-logging-info` (for LCG LB) commands query the LB persistent database for logging information about previously submitted jobs. The job's logging information is stored permanently by the LB service and can be retrieved also after the job has terminated its life-cycle. This is especially useful in the analysis of job failures.

The argument of this command is a list of one or more job identifiers. The `-i` and `-o` options work as in the previous commands. As an example for the gLite LB consider:

```
$ glite-job-logging-info -v 1  https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA


**********************************************************************
LOGGING INFORMATION:

Printing info for the Job : https://cert-rb-01.cnaf.infn.it:9000/55YfzeDigWeoHbpHxx1BQA

        ---
Event: RegJob
- source                   =    UserInterface
- timestamp                =    Mon May 15 15:14:55 2006 CEST
        ---
Event: Transfer
- destination              =    NetworkServer
- result                   =    START
- source                   =    UserInterface
- timestamp                =    Mon May 15 15:14:55 2006 CEST
        ---
Event: Accepted
- source                   =    NetworkServer
- timestamp                =    Mon May 15 15:15:42 2006 CEST
        ---
Event: Transfer
- destination              =    NetworkServer
- result                   =    OK
- source                   =    UserInterface
- timestamp                =    Mon May 15 15:15:43 2006 CEST
        ---
Event: EnQueued
- result                   =    OK
- source                   =    NetworkServer
- timestamp                =    Mon May 15 15:15:44 2006 CEST
        ---
Event: DeQueued
- source                   =    WorkloadManager
- timestamp                =    Mon May 15 15:15:44 2006 CEST
        ---
Event: Match
```

```
- dest_id                =    cert-ce-03.cnaf.infn.it:2119/blah-lsf-pps
- source                 =    WorkloadManager
- timestamp              =    Mon May 15 15:15:45 2006 CEST
          ---
[...]
```

6.3.3. The BrokerInfo

The ***BrokerInfo file*** is a mechanism by which the user job can access, at execution time, certain information concerning the job, for example the name of the CE, the files specified in the `InputData` attribute, the SEs where they can be found, etc.

The BrokerInfo file is created in the job working directory (that is, the current directory on the WN for the executable) and is named `.BrokerInfo`. Its syntax is, as in job description files, based on Condor ClassAds and the information contained is not easy to read; however, it is possible to get it by means of a CLI, whose description follows.

NOTE: Remember that, as explained previously, if the option `-r` is used when submitting a job, in order to make the job end up in a particular CE, the BrokerInfo file will not be created.

The information about the BrokerInfo file, the `glite-brokerinfo` (or `edg-brokerinfo` for LCG WMS) CLI, and its respective API can be found in [36].

The `glite-brokerinfo` command has the following syntax:

```
edg-brokerinfo [-v] [-f <filename>] function [parameter] [parameter] ...
```

where `function` is one of the following:

- `getCE`: returns the name of the CE the job is running on;
- `getDataAccessProtocol`: returns the protocol list specified in the `DataAccessProtocol` JDL attribute;
- `getInputData`: returns the file list specified in the `InputData` JDL attribute;
- `getSEs`: returns the list of the Storage Elements with contain a copy of at least one file among those specified in `InputData`;
- `getCloseSEs`: returns a list of the Storage Elements close to the CE;
- `getSEMountPoint <SE>`: returns the access point for the specified `<SE>`, if it is in the list of close SEs of the WN.
- `getSEFreeSpace <SE>`: returns the free space on `<SE>` at the moment of match-making;
- `getLFN2SFN <LFN>`: returns the storage file name of the file specified by `<LFN>`, where `<LFN>` is a logical file name of a GUID specified in the `InputData` attribute;
- `getSEProtocols <SE>`: returns the list of the protocols available to transfer data in the Storage Element `<SE>`;

- `getSEPort <SE> <Protocol>`: returns the port number used by `<SE>` for the data transfer protocol `<Protocol>`;
- `getVirtualOrganization`: returns the name of the VO specified in the `VirtualOrganisation` JDL attribute;

The `-v` option produced a more verbose output, and the `-f <filename>` option tells the command to parse the BrokerInfo file specified by `<filename>`. If the `-f` option is not used, the command tries to parse the file `$GLITE_WL_RB_BROKERINFO`.

There are basically two ways for parsing elements from a BrokerInfo file.

The first one is directly from the job, and therefore from the WN where the job is running. In this case, the `$GLITE_WL_RB_BROKERINFO` variable is defined as the location of the `.BrokerInfo` file, in the working directory of the job, and the command will work without problems. This can be accomplished for instance by including a line like the following in a submitted shell script:

```
glite-brokerinfo getCE
```

where the `glite-brokerinfo` command is called with any desired function as its argument.

If, on the contrary, `glite-brokerinfo` is invoked from the UI, the `$GLITE_WL_RB_BROKERINFO` variable will be usually undefined, and an error will occur. The solution to this is to include an instruction to generate the `.BrokerInfo` file as output of the submitted job, and retrieve it with the rest of generated output (by specifying the file in the Output Sandbox), when the job finishes. This can be done for example by including the following lines:

```
#!/bin/sh
cat $GLITE_WL_RB_BROKERINFO
```

in a submitted shell script.

Then, the file can be accessed locally with the `-f` option commented above.

6.3.4. Interactive Jobs

NOTE: Interactive jobs are not supported in WLCG/EGEE, and that functionality is not part of the official distribution of the current release. Interactive jobs in the gLite 3.0 middleware have never been tested and will not be described here. Any site installing or using it will do it only under its own responsibility.

This section gives an overview of how interactive jobs should work in the LCG Workload Management System.

Interactive jobs are specified setting the JDL `JobType` attribute to `Interactive`. When an interactive job is submitted, the `edg-job-submit` command starts a Grid console shadow process in the background, which listens on a port for the job standard streams. Moreover, the `edg-job-submit` command opens a new window where the incoming job streams are forwarded. The port on which the shadow process listens is assigned by the Operating System (OS), but can be forced through the `ListenerPort` attribute in the JDL.

As the command in this case opens an X window, the user should make sure the `DISPLAY` environment variable is correctly set, an X server is running on the local machine and, if he is connected to the UI node from a remote machine (e.g. with ssh), secure X11 tunneling is enabled. If this is not possible, the user can specify the `--nogui` option, which makes the command provide a simple standard non-graphical interaction with the running job.

Example 6.3.4.1 (Simple interactive job)

The following `interactive.jdl` file contains the description of a very simple interactive job. Please note that the `OutputSandbox` is not necessary, since the output will be sent to the interactive window (it could be used for further output, though).

```
[
JobType = "Interactive" ;
Executable = "interactive.sh" ;
InputSandbox = {"interactive.sh"} ;
]
```

The executable specified in this JDL is the `interactive.sh` script, which follows:

```
#!/bin/sh
echo "Welcome!"
echo -n "Please tell me your name: "
read name
echo "That is all, $name."
echo "Bye bye."
exit 0
```

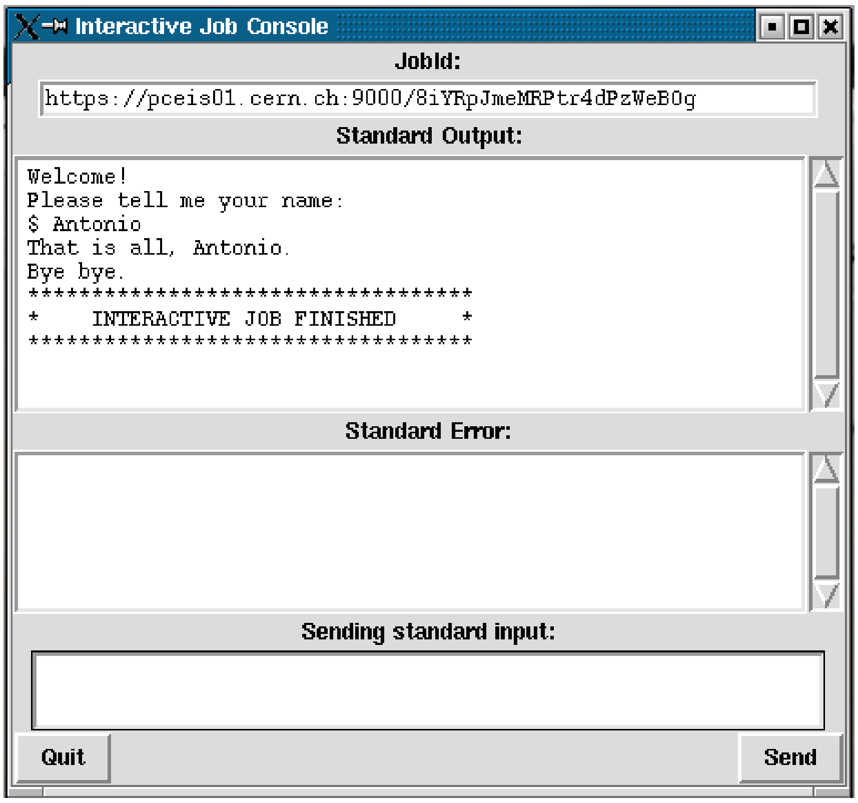
The `interactive.sh` script just presents a welcome message to the user, and then asks and waits for an input. After the user has entered a name, this is shown back just to check that the input was received correctly. Figure 10 shows the result of the program (after the user has entered his name) in the generated X window.

Another option that is reserved for interactive jobs is `--nolisten`: it makes the command forward the job standard streams coming from the WN to named pipes on the UI machine, whose names are returned to the user together with the OS id of the listener process. This allows the user to interact with the job through his own tools. It is important to note that when this option is specified, the UI has no more control over the launched listener process that has hence to be killed by the user (through the returned process id) when the job is finished.

Example 6.3.4.2 (Interacting with the job through a bash script)

A simple script (`dialog.sh`) to interact with the job is presented in this section. It is assumed that the `--nolisten` option was used when submitting the job. The function of the script is to get the information sent by the interactive job, present it to the user, and send the user's response back to the job.

As arguments, the script accepts the names of the three pipes (input, output, and error) that the job will use,



```
https://pceis01.cern.ch:9000/8iYRpJmeMRPtr4dPzWeB0g
```

```
Welcome!
Please tell me your name:
$ Antonio
That is all, Antonio.
Bye bye.
***********************************
*    INTERACTIVE JOB FINISHED     *
***********************************
```

Figure 10: X window for an interactive job

and the process id (pid) of the listener process. All this information is returned when submitting the job, as can be seen in the returned answer for the submission of the same `interactive.jdl` and `interactive.sh` used before:

```
$ edg-job-submit --nolisten interactive.jdl

Selected Virtual Organisation name (from UI conf file): dteam
Connecting to host pceis01.cern.ch, port 7772
Logging to host pceis01.cern.ch, port 9002

*********************************************************************************************
                              JOB SUBMIT OUTCOME
 The job has been successfully submitted to the Network Server.
 Use edg-job-status command to check job current status.
 Your job identifier (edg_jobId) is:

 - https://pceis01.cern.ch:9000/IxKsoi8I7fXbygN56dNwug

 ----
 The Interactive Streams have been successfully generated
 with the following parameters:

 Host:                          137.138.228.252
 Port:                          37033
 Shadow process Id:             7335
 Input Stream  location:        /tmp/listener-IxKsoi8I7fXbygN56dNwug.in
 Output Stream  location:       /tmp/listener-IxKsoi8I7fXbygN56dNwug.out
 Error Stream  location:        /tmp/listener-IxKsoi8I7fXbygN56dNwug.err
 ----
***************************************************************************
```

Once the job has been submitted, the `dialog.sh` script can be invoked, passing the four arguments as described earlier. The code of the script is quite simple, as it just reads from the output pipe and waits for the user's input, which, in this case, will be just one string. This string (the user's name) is the only thing that our job (`interactive.sh`) needs to complete its work. A more general tool should keep waiting for further input in a loop, until the user instructs it to exit. Of course, some error checking should be also added.

The code of `dialog.sh` follows:

```
#!/bin/bash

# Usage information
if [ $# -lt 4 ]; then
   echo 'Not enough input arguments!'
   echo 'Usage: interaction.sh <input_pipe> <output_pipe> <error_pipe> <listener_pid>'
   exit -1  # some error number
fi

# Welcome message
echo -e "\nInteractive session
started\n----------------------------------\n"

# Read what the job sends and present it to the user
cat < $2 &

# Get the user reply
read userInput
echo $userInput > $1

# Clean up (wait two seconds for the pipes to be flushed out)
sleep 2
rm $1 $2 $3  # Remove the pipes
if [ -n $4 ]; then
   kill $4   # Kill the shadow listener
fi

# And we are done
echo -e "\n----------------------------------"
echo "The temporary files have been deleted, and the listener process killed"
echo "The interactive session ends here "
exit 0
```

Note that, before exiting, the script removes the temporary pipe files and kills the listener process. This must be done either inside the script or manually by the user if the `--nolisten` option is used (otherwise, the X window or text console interfaces created by `edg-job-submit` will do it automatically).

Now, let us see what the result of the interaction is:

```
$ dialog.sh \
/tmp/listener-IxKsoi8I7fXbygN56dNwug.in \
/tmp/listener-IxKsoi8I7fXbygN56dNwug.out \
/tmp/listener-IxKsoi8I7fXbygN56dNwug.err \
7335

Interactive session started
----------------------------------


Welcome!
Please tell me your name: Antonio
That is all, Antonio.
Bye bye.
***********************************
*    INTERACTIVE JOB FINISHED     *
***********************************


----------------------------------
The temporary files have been deleted, and the listener process killed
The interactive session ends here
```

Until now, several options for the `edg-job-submit` command used for interactive jobs have been explained; but there is another command that is used for this kind of jobs. It is the `edg-job-attach` command.

Usually, the listener process and the X window are started automatically by `edg-job-submit`. However, in the case that the interactive session with a job is lost, or if the user needs to follow the job from a different machine (not the UI), or on another port, a new interactive session can be started with the `edg-job-attach` command. This commands starts a listener process on the UI machine that is attached to the standard streams of a previously submitted interactive job and displays them on a dedicated window. The `--port <port_number>` option specifies the port on which the listener is started.

6.3.5. Checkpointable Jobs

NOTE: Checkpointable jobs are not supported in WLCG/EGEE, and that functionality is not part of the official distribution of the current gLite 3.0 release. Checkpointable jobs in the gLite 3.0 middleware have never been tested and will not be described here. Any site installing or using it will do it only under its own responsibility.

This section gives a brief overview of how checkpointable jobs should work in gLite 3.0.

Checkpointable jobs are jobs that can be logically decomposed in several steps. The job can save its state in a particular moment, so that if the job fails, that state can be retrieved and loaded by the job later. In this way, a checkpointable job can start running from a previously loaded state, instead of starting from the beginning again.

Checkpointable jobs are specified by setting the JDL `JobType` attribute to `Checkpointable`. When a checkpointable job is submitted the user can specify the number (or list) of steps in which the job can be decomposed, and the step to be considered as the initial one. This can be done by setting respectively the JDL attributes `JobSteps` and `CurrentStep`. The `CurrentStep` attribute is a mandatory attribute and if not provided by the user, it is set automatically to 0 by the UI.

When a checkpointable job is submitted to be run from the beginning, it is submitted as any other job, using the `edg-job-submit` command. If, on the contrary, the job must start from a intermediate state (e.g., after a crash), the `--chkpt <state_file>` option may be used, where `state_file` must be a valid JDL file, where the state of a previously submitted job was saved. In this way, the job will first load the given state and then continue running until it finishes. That JDL job state file can be obtained by using the `edg-job-get-chkpt <jobid>` command.

6.3.6. MPI Jobs

Message Passing Interface (MPI) applications are run in parallel on several processors.

Note: In WLCG/EGEE, there is no native support for MPI jobs, therefore, computing centers supporting MPI jobs should provide a particular farm configuration. In gLite 3.0, there is notive support for MPI jobs, but since they have never been tested, they will not be described here.

Some WLCG/EGEE sites support jobs that are run in parallel using the MPI standard. It is not mandatory for WLCG/EGEE clusters to support MPI, so those clusters who do must publish this in the IS. They do so by adding the value `"MPICH"` to the `GlueHostApplicationSoftwareRunTimeEnvironment` GLUE attribute. User's jobs can then look for this attribute in order to find clusters that support MPI. This is done transparently for the user by the use of a requirements expression in the JDL file, as shown later.

From the user's point of view, jobs to be run as MPI are specified setting the JDL `JobType` attribute to `MPICH`. When that attribute is included, then the presence of the `NodeNumber` attribute in the JDL is mandatory as well. This variable specifies the required number of CPUs for the application. These two attributes could be specified as follows:

```
JobType="MPICH";
NodeNumber=4;
```

The UI automatically requires the MPICH runtime environment installed on the CE and a number of CPUs at least equal to the required number of nodes. This is done by adding an expression like the following:

```
(other.GlueCEInfoTotalCPUs >= <NodeNumber> ) &&
Member("MPICH",other.GlueHostApplicationSoftwareRunTimeEnvironment)
```

to the the JDL requirements expression (remember that this addition is automatically performed by the UI, so you do not have to do it yourself).

Attention: The executable that is specified in the JDL must not be the MPI application directly, but a wrapper script that invokes this MPI application by calling `mpirun`[3]. This allows the user to perform some pre- and post-execution steps in the script. This usually includes also the compilation of the MPI application, since the resulting binary may be different depending on the MPI version and configuration.

It is good practice to specify the list of machines that `mpirun` will use, not to risk the usage of a default (possibly stale) list. This cannot be the case if `mpiexec` is used, but this command is only available on Torque or PBS systems.

One more thing to say regarding MPI jobs is that some of them require a shared filesystem among the WNs to run. In order to know if a CE offers a shared filesystem, the variable `VO_<name_of_VO>_SW_DIR` defined in each WN can be checked. The variable will contain a name of a directory for CEs with shared file systems, while it will just hold a ".", if there is no shared file systems.

The following example shows a complete example of MPI job.

Example 6.3.6.1 ***(MPI job submission)***

The very simple MPI application could be the following (`MPItest.c`):

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
  int numprocs;  /* Number of processors */
  int procnum;   /* Processor number */
  /* Initialize MPI */
  MPI_Init(&argc, &argv);
  /* Find this processor number */
  MPI_Comm_rank(MPI_COMM_WORLD, &procnum);
  /* Find the number of processors */
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  printf ("Hello world! from processor %d out of %d\n", procnum, numprocs);
  /* Shut down MPI */
  MPI_Finalize();
  return 0;
}
```

The JDL file would be:

```
Type = "Job";
```

[3]This is so because the job type that is used at the globus gatekeeper level is `multiple`. If you bypass the LCG middleware and submit a job using globus directly, and if you specify `mpi` as the job type, then globus calls `mpirun` directly on the specified executable. This is rather limiting because no pre- or post-MPI activity can be performed.

```
JobType = "MPICH";
NodeNumber = 10;
Executable = "MPItest.sh";
Arguments = "MPItest";
StdOutput = "test.out";
StdError = "test.err";
InputSandbox = {"MPItest.sh","MPItest.c"};
OutputSandbox = {"test.err","test.out","mpiexec.out"};
```

And the script send as executable would be the following (`MPItest.sh`):

```
#!/bin/sh -x

# Binary to execute
EXE=$1

echo "***********************************************************************"
echo "Running on: $HOSTNAME"
echo "As:       " `whoami`

echo "***********************************************************************"
echo "Compiling binary: $EXE"
echo mpicc -o ${EXE} ${EXE}.c
mpicc -o ${EXE} ${EXE}.c

if [ "x$PBS_NODEFILE" != "x" ] ; then
  echo "PBS Nodefile: $PBS_NODEFILE"
  HOST_NODEFILE=$PBS_NODEFILE
fi

if [ "x$LSB_HOSTS" != "x" ] ; then
  echo "LSF Hosts: $LSB_HOSTS"
  HOST_NODEFILE=`pwd`/lsf_nodefile.$$
  for host in ${LSB_HOSTS}
  do
    echo $host >> ${HOST_NODEFILE}
  done
fi

if [ "x$HOST_NODEFILE" = "x" ]; then
  echo "No hosts file defined.  Exiting..."
  exit
fi

echo "***********************************************************************"
CPU_NEEDED=`cat $HOST_NODEFILE | wc -l`
echo "Node count: $CPU_NEEDED"
echo "Nodes in $HOST_NODEFILE: "
cat $HOST_NODEFILE
```

```
echo "**********************************************************************"
CPU_NEEDED=`cat $HOST_NODEFILE | wc -l`
echo "Checking ssh for each node:"
NODES=`cat $HOST_NODEFILE`
for host in ${NODES}
do
  echo "Checking $host..."
  ssh $host hostname
done

echo "**********************************************************************"
echo "Executing $EXE with mpirun"
chmod 755 $EXE
mpirun -np $CPU_NEEDED -machinefile $HOST_NODEFILE `pwd`/$EXE
```

In the script, the MPI application is first compiled, then the list of hosts where to run is created by reading from the appropriate batch system information. The variable `CPU_NEEDED` stores the number of nodes that are available. The script also checks that ssh works for all listed nodes. This step should not be required but it is a good safety measure to detect misconfigurations in the site and avoid future problems. Finally, `mpirun` is called with the `-np` and `-machinefile` options specified.

The retrieved output of a job execution follows:

```
***********************************************************************
Running on: node16-4.farmnet.nikhef.nl
As:         dteam005

***********************************************************************
Compiling binary: MPItest
mpicc -o MPItest MPItest.c

PBS Nodefile: /var/spool/pbs/aux/625203.tbn20.nikhef.nl
***********************************************************************
Node count:      10
Nodes in /var/spool/pbs/aux/625203.tbn20.nikhef.nl:
node16-4.farmnet.nikhef.nl
node16-44.farmnet.nikhef.nl
node16-45.farmnet.nikhef.nl
node16-45.farmnet.nikhef.nl
node16-46.farmnet.nikhef.nl
node16-46.farmnet.nikhef.nl
node16-47.farmnet.nikhef.nl
node16-47.farmnet.nikhef.nl
node16-48.farmnet.nikhef.nl
node16-48.farmnet.nikhef.nl
***********************************************************************
Checking ssh for each node:
Checking node16-4.farmnet.nikhef.nl...
```

```
node16-4.farmnet.nikhef.nl
Checking node16-44.farmnet.nikhef.nl...
node16-44.farmnet.nikhef.nl
Checking node16-45.farmnet.nikhef.nl...
node16-45.farmnet.nikhef.nl
Checking node16-45.farmnet.nikhef.nl...
node16-45.farmnet.nikhef.nl
Checking node16-46.farmnet.nikhef.nl...
node16-46.farmnet.nikhef.nl
Checking node16-46.farmnet.nikhef.nl...
node16-46.farmnet.nikhef.nl
Checking node16-47.farmnet.nikhef.nl...
node16-47.farmnet.nikhef.nl
Checking node16-47.farmnet.nikhef.nl...
node16-47.farmnet.nikhef.nl
Checking node16-48.farmnet.nikhef.nl...
node16-48.farmnet.nikhef.nl
Checking node16-48.farmnet.nikhef.nl...
node16-48.farmnet.nikhef.nl
*************************************************************************
Executing MPItest with mpirun
Hello world! from processor 2 out of 10
Hello world! from processor 6 out of 10
Hello world! from processor 3 out of 10
Hello world! from processor 4 out of 10
Hello world! from processor 7 out of 10
Hello world! from processor 8 out of 10
Hello world! from processor 5 out of 10
Hello world! from processor 1 out of 10
Hello world! from processor 9 out of 10
Hello world! from processor 0 out of 10
```

6.3.7. Advanced Command Options

All the `glite-job-*` and `edg-job-*` commands read some configuration files which the user can edit, if he is not satisfied with the default ones.

The main configuration file is located under `$GLITE_WMS_LOCATION/etc/glite_wmsui_cmd_var.conf` (in a gLite 3.0 UI) and `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` (in a LCG UI).

This file sets, among other things, the default VO, the default location for job outputs and command log files and the default values of mandatory JDL attributes.

It also adds by default the requirement `other.GlueCEStateStatus == "Production"`, so that CEs that are not in a condition to accept jobs (e.g. `Closed`) do not match.

It is possible to point to a different configuration file by setting the value of the environment variable

`$GLITE_WMS_UI_CONFIG_VAR` (in a gLite 3.0 UI)

`$EDG_WL_UI_CONFIG_VAR` (in a LCG UI)

to the file path, or by specifying the file in the `--config <file>` option of the `glite-job-*` or `edg-job-*` commands (which takes precedence).

In addition, VO-specific configurations are defined by default in the file

`$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf` (in a LCG UI)

`$GLITE_WMS_LOCATION/etc/<vo>/glite_wmsui.conf` (in a gLite 3.0 UI)

consisting essentially in the list of Network Servers, Proxy Servers and LB servers to be used when submitting jobs. A different file can be specified using the variable

`$EDG_WL_UI_CONFIG_VO` (in a LCG UI)

`$GLITE_WMS_UI_CONFIG_VO` (in a gLite 3.0 UI)

or the `--config-vo <file>` option of the `glite-job-*` `edg-job-*` commands. This can be useful for example to use a WMS that is different from the default one.

Other options present in the `glite-job-*` and `edg-job-*` commands are the following: the `--log <file>` option allows the user to define the log file; the default log file is named `<command_name>_<UID>_<PID>_<date_time>.log` and it is found in the directory specified in the configuration file. The `--noint` option skips all interactive questions and prints all warning and error messages to a log file. The `--help` and `--version` options are self-explanatory.

Example 6.3.7.1 (Changing the default VO)

In a gLite 3.0 UI, the user can change his default VO by performing the following steps:

a. Make a copy of the file `$GLITE_WMS_LOCATION/etc/glite_wms_ui_cmd_var.conf`, for example to `$HOME/my_ui.conf`.
b. Edit `$HOME/my_ui.conf` and change this line:

   ```
   DefaultVo = "cms";
   ```

   if, for example, he wants to set the CMS VO as default.
c. Define in the shell configuration script (`$HOME/.bashrc` for bash and `$HOME/.cshrc` for csh/tcsh) the environment variable

   ```
   setenv GLITE_WMS_UI_CONFIG_VAR $HOME/my_ui.conf      ((t)csh)
   export GLITE_WMS_UI_CONFIG_VAR=$HOME/my_ui.conf      (bash)
   ```

Example 6.3.7.2 (Using several RBs)

Several NSs as well as LBs can be specified in the previously indicated VO-specific configuration file. In this way the submission tool will try to use the first NS specified and will use another one if this attempt fails (e.g.: the NS is not accessible).

The syntax to do this can be deduced from the following example:

```
NSAddresses = { NS_1 , NS_2 };
LBAddresses = { { LB_1a, LB_1b }, { LB_2 } };
```

In this case, the first NS to be contacted is `NS_1`, and either `LB_1a` or `LB_1b` (if the first one is not available) will be used as LB servers. If `NS_1` cannot be contacted, then `NS_2` and `LB_2` will be used instead. In general, it is probably more useful to just specify several NSs and then the associated LB (usually in the same RB or in a close machine) to each one of them (always using curly brackets as shown in the example).

7. Data Management

7.1. Introduction

This chapter describes the client tools that are available to deal with the data in gLite 3.0. An overview of the available Data Management APIs is also is given in Appendix F.

7.2. Storage Elements

The ***Storage Element*** is the service which allows a user or an application to store data for future retrieval. Even if it is foreseen for the future, currently there is no enforcement of policies for space quota management. All data in a SE must be considered permanent (no scratchable via automatic mechanism local on the site) and it is user responsibility to manage the available space in a SE (removing unnecessary data, moving files to mass storage systems etc.).

7.2.1. Data Channel Protocols

The data access protocols supported in current gLite 3.0 are summarized in the next table:

Protocol	Type	GSI secure	Description	Optional
GSIFTP	File Transfer	Yes	FTP-like	No
gsidcap	File I/O	Yes	Remote file access	Yes
insecure RFIO	File I/O	No	Remote file access	Yes
secure RFIO (gsirfio)	File I/O	Yes	Remote file access	Yes

In the current gLite 3.0 release, every SE must have a ***GSIFTP*** server [38][4]. The GSIFTP protocol offers basically the functionality of FTP, i.e. the transfer of files, but enhanced to support GSI security. It is responsible for secure, fast and efficient file transfers to/from Storage Elements. It provides third party control of data transfer as well as parallel streams data transfer.

GSIFTP is currently supported by every Grid SE and it is thus the main file transfer protocol in gLite 3.0. However, for the remote access (and not only copy) of files stored in the SEs, the protocols currently supported by gLite are the ***Remote File Input/Output protocol (RFIO)*** [39] and the ***GSI dCache Access Protocol (gsidcap)***.

RFIO was developed to access tape archiving systems, such as CASTOR (CERN Advanced STORage manager)[5].

CASTOR does not implement yet a GSI enabled version of RFIO (as DPM does) and therefore can only be used to access data from any WN within the Local Area Network (LAN); the only authentication is through UID

[4]In the literature, the terms *GridFTP* and *GSIFTP* are sometimes used interchangeably. Strictly speaking, GSIFTP is a subset of GridFTP. Please refer to [38] for more information.

[5]A CERN hierarchical Storage Manager [40]

and GID. The secure RFIO on the other hands can be used for file access to any remote network storage and also from a UI.

There is some more information about RFIO in Appenddix F.

The gsidcap protocol is the GSI secure version of the dCache[6] access protocol, *dcap*. Being GSI-secure, gsidcap can be used for inter-site remote file access.

It is possible that in the future *kdcap*, the Kerberos secure version of dcap, will be also supported.

The ***file*** protocol was used in the past for local file access to remote network file systems. Currently this option is not supported anymore and the *file* protocol is only used to specify a file on the local machine (i.e. in a UI or a WN), but not stored in a Grid SE.

7.2.2. The Storage Resource Manager interface

The Storage Resource Manager (SRM) has been designed to be the single interface (through the corresponding SRM protocol) for the management of disk and tape storage resources. Any kind of Storage Element will **eventually** offer an SRM interface that will hide the complexity of the resources behind it and allow the user to request files, pin them for a specified lifetime, reserve space for new entries, and so on. Behind this interface, the SE will have a site-specific policy defined, according to which files are migrated from disk to tape, users are allowed to read and write, etc. SRMs will be able also to handle request for transfers from one SRM to another one (third party copy).

It is important to notice that the SRM protocol is a storage management protocol and not a file access or file transfer one. For these tasks the client application will access directly the appropriate file access or transfer server.

Unfortunately, at the moment, not all SEs implement the same version of the SRM interface, and none of these versions offers all of the functionalities that the SRM standard ([19]) defines. The high level Data Management tools and APIs will, in general, interact transparently with SRM.

7.2.3. Types of Storage Elements

There are different types of possible SEs in gLite 3.0.

- ***Classic SE***: it consists of a GridFTP server and an insecure RFIO daemon (***rfiod***) in front of a physical single disk or disk array. The GridFTP server supports *secure* data transfers. The rfiod daemon ensures LAN-limited file access through RFIO. If the same Classic SE serves multiple Virtual Organizations, the only way to allocate disk quota per VO is through physical partitioning of the disk, which is an option that site managers in general do not like. In other words, a single VO might fill up the entire SE. Once again, it is user responsibility to monitor disk usage in the case of a Classic SE. Furthermore, the classic SE **does NOT support the SRM interface** (and never will).

[6]A storage Manager developed by the Deutsches Elektronen-Synchrotron (DESY) and the Fermi National Accelerator Laboratory (FERMI). More information in [41]

- ***Mass Storage System***: it consists of a ***Hierarchical Storage Management (HSM)*** system for files that may be migrated between front-end disk and back-end tape storage hierarchies. The migration of files between disk and tape storage is managed by a ***stager*** process. The stager manages one or more disk pools or groups of one or more UNIX file systems, residing on one or more disk servers. The stager is responsible for space allocation and for file migration between the disk pool and tape storage. The MSS exposes to the user a virtual file system which hides the complexity of the internal details.

  A *classic* interface to a MSS consists in a GridFTP front-end (also a load-share balance solution has been deployed) which ensures file transfer capabilities. The files access protocol depends instead on the type of Mass Storage System: insecure RFIO for CASTOR, gsidcap for a dCache disk pool manager front-end, etc. It is responsibility of the application to figure out which type of protocol is supported (for instance querying the information system) and access files accordingly. Nevertheless, the ***GFAL*** API does this transparently (see Appendix F).

  The CASTOR MSS can also expose an SRM interface. This is a desired solutions since it hides internal complexities inherent to access and transfer protocols. In addition, it makes it possible to pre-stage files (migrate them on the stager beforehand, so that they are available on disk to an application at runtime), to pin and unpin files (ensure the persistency of files on disk until they are *released*), to reserve space in the stager, etc.

  SRM interfaces to other MSS rather than CASTOR are not supported by gLite 3.0 although they are strongly desirable. It is therefore up to the sites to provide an SRM implementation for their specific MSS.

- **dCache Disk pool manager**: it consists of a dCache server and one or more pool nodes. The server represents the single point of access to the SE and presents files in the pool disks under a single virtual file system tree. Nodes can be dynamically added to the pool. File transfer is managed through GridFTP while the native gsidcap protocol allows POSIX-like data access. It presents an SRM interface which allows to overcome the limitations of the Classic SE.

- **LCG Disk pool manager**: is the LCG lightweight alternative to dCache; easy to install and, although not so powerful as dCache, offers all the functionality required by small sites. Disks can be added dynamically to the pool at any time. Like in dCache, a virtual file system hides the complexity of the disk pool architecture. The LCG DPM includes a GridFTP server for file transfer and ensures file access through secure RFIO. It also presents an SRM interface. In addition, disk quota allocation per VO is supported. For these reasons, once the DPM is deployed, it will replace the Classic SE. Old Classic SEs will be converted to DPMs with only one disk in the pool.

7.3. Files Naming Convention in gLite 3.0

As an extension of what was introduced in Chapter 3, the different types of names that can be used within the gLite 3.0 files catalogs are summarized below:

- The ***Grid Unique IDentifier (GUID)***, which identifies a file uniquely, is of the form:

  ```
  guid:<40_bytes_unique_string>
  guid:38ed3f60-c402-11d7-a6b0-f53ee5a37e1d
  ```

- The ***Logical File Name (LFN)*** or User Alias, which can be used to refer to a file in the place of the GUID (and which should be the normal way for a user to refer to a file), has this format:

```
lfn:<anything_you_want>
lfn:importantResults/Test1240.dat
```

In case the LCG File Catalog is used (see Section 7.4), the LFNs are organized in a hierarchical directory-like structure, and they will have the following format:

```
lfn:/grid/<MyVO>/<MyDirs>/<MyFile>
```

- The ***Storage URL (SURL)***, also known as ***Physical File Name (PFN)***, which identifies a replica in a SE, is of the general form:

  ```
  <sfn | srm>://<SE_hostname>/<some_string>
  ```

  where the prefix will be `sfn` for files located in SEs without SRM interface and `srm` for SRM-managed SEs.

  In the case of `sfn` prefix, the string after the hostname is the path to the location of the file and can be decomposed in the SE's access-point (path to the storage area of the SE), the relative path to the VO of the file's owner and the relative path to the file.

  ```
  sfn://<SE_hostname><SE_Accesspoint><VO_path><filename>
  sfn://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/file3596e86f-c402-1
  1d7-a6b0-f53ee5a37e1d
  ```

  In the case of SRM-managed SEs, one cannot assume that the SURL will have any particular format, other than the `srm` prefix and the hostname. In general, SRM-managed SEs can use virtual file systems and the name a file receives may have nothing to do with its physical location (which may also vary with time). An example of this kind of SURL follows:

  ```
  srm://castorgrid.cern.ch/castor/cern.ch/grid/dteam/generated/2004-09-15/file24e3227a
  -cb1b-4826-9e5c-07dfb9f257a6
  ```

- The ***Transport URL (TURL)***, which is a valid URI with the necessary information to access a file in a SE, has the following form:

  ```
  <protocol>://<some_string>
  gsiftp://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/file3596e86f-c40
  2-11d7-a6b0-f53ee5a37e1d
  ```

  where `<protocol>` must be a valid protocol (supported by the SE) to access the contents of the file (GSIFTP, RFIO, gsidcap), and the string after the double slash may have any format that can be understood by the SE serving the file. While SURLs are in principle invariable (they are entries in the file catalog, see Section 7.4), TURLs are obtained dynamically from the SURL through the Information System or the SRM interface (for SRM managed SEs). The TURL therefore can change with time and should be considered only valid for a relatively small period of time after it has been obtained.

7.4. File Catalogs in gLite 3.0

Users and applications need to locate files (or replicas) on the Grid. The ***File Catalog*** is a service which fulfills such requirement, maintaining mappings between LFN(s), GUID and SURL(s).

In gLite 3.0, two types of file catalogs are currently deployed: the old ***Replica Location Server (RLS)*** and the new ***LCG File Catalog (LFC)***. Both of them are deployed as centralized catalogs.

The catalogs publish their ***endpoints*** (service URL) in the Information Service so that the LCG Data Management tools and any other interested services (the RB for example) can find the way to them (and then to the Grid files information). Be aware that for the RLS there are two different endpoints (one for LRC and one for RMC) while for LFC, being it a single catalog, there is only one.

The user can decide which catalog to use by setting the environmental variable `LCG_CATALOG_TYPE` equal to `edg` for RLS or `lfc` for the LFC. Since gLite 3.0 the default file catalog is LFC.

Attention: the RLS and LFC are not respectively mirrored. Entries in the LFC will not appear in RLS and vice-versa. Choose to use any of the two catalogs but be consistent with your choice.

The RLS in fact it consists of two catalogs: the ***Local Replica Catalog (LRC)*** and the ***Replica Metadata Catalog (RMC)***.

The LRC keeps mappings between GUIDs and SURLs, while the RMC keeps mappings between GUIDs and LFNs. Both RMC and LRC support the use of metadata. User metadata should all be confined in RMC while LRC should contain only system metadata (file size, creation date, checksum etc.). Please refer to the LCG-2 User Guide [18] for further readings and acknowledgments about RLS structure and usage.

The LFC was developed to overcome some serious performance and security problems of the old RLS catalogs; it also adds some new functionalities such as transactions, roll-backs, sessions, bulk queries and a hierarchical name-space for LFNs. It consists of a unique catalog, where the LFN is the main key, see Figure 11. Further LFNs can be added as symlink to the main LFN. System metadata is supported while for user metadata only a single string entry is available (rather a *description field* than real metadata).

Migrations from RLS to LFC , made possible through specific tools, have permitted all VOs to gradually adopt LFC as unique File Catalog in LCG and to abandon definitely RLS.

IMPORTANT: A file is considered to be a ***Grid file*** if it is **both** physically present in a SE **and** registered in the file catalog. In this chapter several tools will be described. In general high level tools like lcg_utils (see Sec. 7.6.1) will ensure consistency between files in the SEs and entries in the File Catalog. However, usage of low level tools, for both data transfer and catalog entries management could cause inconsistencies between SEs physical files files and catalog entries; what would imply the corruption of GRID files. This is why the usage of low level tools is strongly discouraged unless really necessary.

7.4.1. LFC Commands

In general terms, the user should usually interact with the file catalog through high level utilities (lcg-utils, see Section 7.6.1). The CLIs and APIs that are available for catalog interaction provide further functionality and more fine-grained control for the operations with the catalog. In some situations, they represent the only possible way to achieve the desired functionality with the LFC.

With gLite 3.0 the variable `$LFC_HOST` must be set to hold the hostname of the LFC server.

Attention: For GFAL and the lcg-utils, the `$LFC_HOST` variable is only another way to define the location

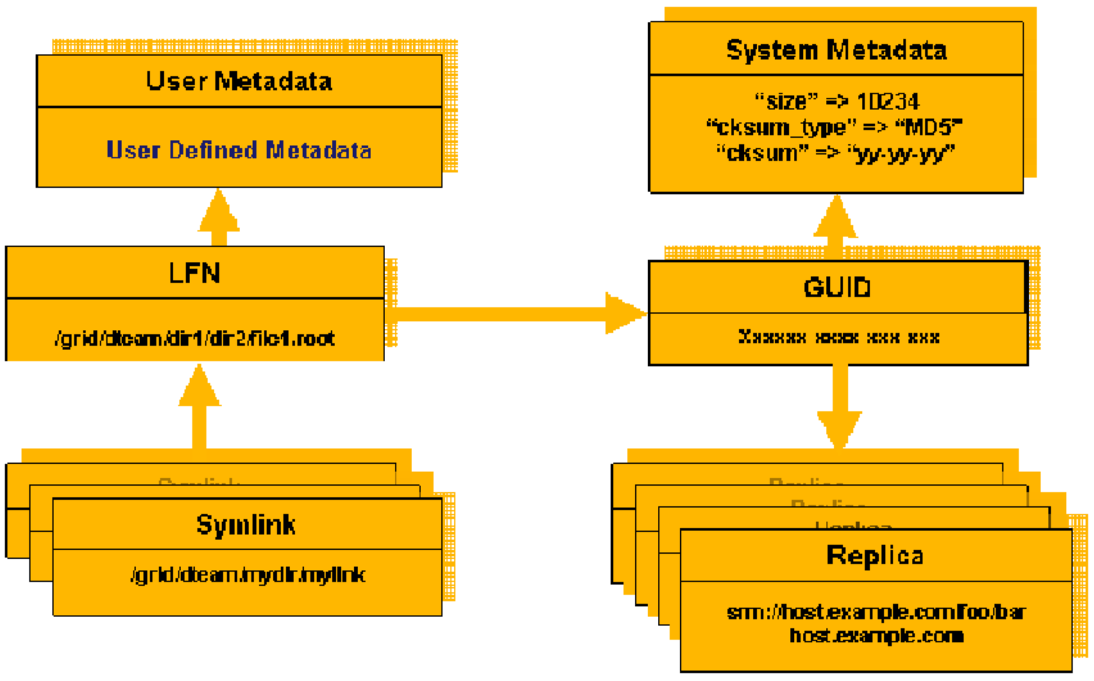


Figure 11: Architecture of the LFC

of the LFC, but for the `lfc-*` commands the variable is required, since these tools do not use the information published in the IS.

The directory structure of the LFC name-space initiates with the `grid` directory. Under this one, there is a directory for each one of the supported VOs. Users of a VO will have read and write permissions only under the directory of their VO (e.g.: `/grid/lhcb` for users of LHCb VO). If such a directory does not exist, then this means that this LFC server does not support that VO.

Once the correct environment has been set, the following commands can be used:

`lfc-chmod`	Change access mode of a LFC file/directory.
`lfc-chown`	Change owner and group of a LFC file/directory.
`lfc-delcomment`	Delete the comment associated with a file/directory.
`lfc-getacl`	Get file/directory access control lists.
`lfc-ln`	Make a symbolic link to a file/directory.
`lfc-ls`	List file/directory entries in a directory.
`lfc-mkdir`	Create directory.
`lfc-rename`	Rename a file/directory.
`lfc-rm`	Remove a file/directory.
`lfc-setacl`	Set file/directory access control lists.
`lfc-setcomment`	Add/replace a comment.
`lfc-entergrpmap`	Defines a new group entry in the Virtual ID table.
`lfc-enterusrmap`	Defines a new user entry in Virtual ID table.
`lfc-modifygrpmap`	Modifies a group entry corresponding to a given virtual gid.
`lfc-modifyusrmap`	Modifies a user entry corresponding to a given virtual uid.
`lfc-rmgrpmap`	Suppresses group entry corresponding to a given virtual gid or group name
`lfc-rmusrmap`	Suppresses user entry corresponding to a given virtual uid or user name.

Many of the commands provide administrative functionality. Their names indicate what their functions are. Man pages are available for all the commands. Most of the commands work in a very similar way to their Unix equivalents, but operating on directories and files of the catalog name-space. Where the path of a file/directory is required, an absolute path can be specified (starting by `/`) or, otherwise, the path is prefixed by the contents of the `$LFC_HOME` environment variable.

Users should use these commands carefully, keeping in mind that the operations they are performing affect the catalog, but not the physical files that the entries represent. This is especially true in the case of the `lfc-rm` command, since it can be used to remove the LFN for a file, but it does not affect the physical files. If all the LFNs pointing to a physical Grid file are removed, then the file is no longer visible to gLite users (although it may still be accessed by using its SURL directly).

Example 7.4.1.1 (Creating directories in the LFC)

Users cannot implicitly create directories in the LFNs name-space when registering new files with LCG Data Management tools (see for example the `lcg-cr` command in Section 7.6.1). Directories must be created in advance using the `lfc-mkdir` command. This is currently the only main use of the `lfc-*` commands for the average user. For other tasks, the lcg-utils should be normally used.

```
$ lfc-mkdir /grid/lhcb/test_roberto/MyTest
$ lfc-ls -l /grid/lhcb/test_roberto
drwxrwxr-x   0 santinel z5                       0 Feb 21 16:50 MyTest
-rwxrwxr-x   1 santinel z5                    1183 Oct 14 11:33 gfal_test
-rwxrwxr-x   1 santinel z5                     270 Jan 09 10:18 input_data
drwxr-xr-x  50 santinel z5                       0 Jun 09  2005 insert
-rwxrwxr-x   1 santinel z5                     215 Jun 22  2005 lfc_test_pic2
-rwxrwxr-x   1 santinel z5                     319 Oct 03 14:49 orriginal.cern-rwxrwxr-x   1 santinel
-rwxrwxr-x   1 santinel z5                     734 Oct 12 11:17 using_project
```

Example 7.4.1.2 (Listing the entries of a LFC directory)

The `lfc-ls` command lists the LFNs of a directory. The command supports several options, out of them the `-l` for long format and `--comment` for showing user-defined metadata information.

Attention: The `-R` option, for recursive listing, is also available for the command, but **it should not be used**. It is a very expensive operation on the catalog and should be avoided.

In the following example the directory `/grid/dteam/MyExample` and its subdirectory `day1` are listed (after being populated).

```
$ lfc-ls /grid/dteam/MyExample

/grid/dteam/MyExample:
day1
day2
day3
day4
interesting


$ lfc-ls /grid/dteam/MyExample/day1

/grid/dteam/MyExample/day1:
measure1
measure2
measure3
measure4
```

Example 7.4.1.3 (Creation of symbolic links)

The `lfc-ln` may be used to create a symbolic link to a file. In this way two different LFNs will point to the same file.

In the following example, we create a symbolic link `/grid/lhcb/test_roberto/MyTest/MyLink` to the original file `/grid/lhcb/test_roberto/lfc_test_pic2`

```
$ lfc-ln -s /grid/lhcb/test_roberto/lfc_test_pic2 /grid/lhcb/test_roberto/MyTest/MyLink
```

Now we can check that link was created with a long listing.

```
$ lfc-ls -l /grid/lhcb/test_roberto/MyTest/MyLink
lrwxrwxrwx   1 santinel    z5              0 Feb 21 16:54 /grid/lhcb/test_roberto/MyTest/MyLink \
  -> /grid/lhcb/test_roberto/lfc_test_pic2
```

Remember that links created with `lfc-ln` are soft. If the LFN they are pointing to is removed, the links themselves are not deleted, but keep existing as broken links.

Example 7.4.1.4 (Adding metadata information to LFC entries)

The `lfc-setcomment` and `lfc-delcomment` commands allow the user to associate a comment with a catalog entry and delete that comment respectively. This is the only user-defined metadata that can be associated with catalog entries. The comments for the files may be listed using the `--comment` option of the `lfc-ls` command. This is shown in the following example:

```
$ lfc-setcomment /grid/dteam/MyExample/interesting/file1 "Most promising measure"

$ lfc-ls --comment /grid/dteam/MyExample/interesting/file1
> /grid/dteam/MyExample/interesting/file1 Most promising measure
```

Example 7.4.1.5 (Removing LFNs from the LFC)

As explained before, the `lfc-rm` will only delete catalog entries, but not physical files. In principle, the deletion of replicas (`lcg-del`) should be used instead. When the last replica is removed, the entry is also removed from the catalog. Indeed, the `lfc-rm` commands will not allow a user to remove an LFN for which there are still SURLs associated (i.e.: physical replicas exist).

That said, there might be some use cases of the command, being the most important one the deletion of directories, by using the `-r` option. This action should be of course performed only in rare occasions and probably by only special users within a VO with administrative privileges.

In the next example, the directory `trash` that has been previously created is removed.

```
$ lfc-ls -l -d /grid/dteam/MyExample/trash
> drwxr-xrwx   0 dteam004 cg                 0 Jul 06 11:13 /grid/dteam/MyExample/trash

$ lfc-rm  /grid/dteam/MyExample/trash
> /grid/dteam/MyExample/trash: Is a directory


$ lfc-ls -l -d /grid/dteam/MyExample/trash
> /grid/dteam/MyExample/trash: No such file or directory
```

7.5. File Transfer Service

FTS (*File Transfer Service*) is the lowest-level data movement service defined in the gLITE architecture and integrated within the gLite 3.0. It's responsible for moving in a reliable way sets of files from one site to another, allowing participant sites to control the network and disk resources usage. FTS is designed for moving physical files point to point. Together with the File Catalog, the data movement protocol GSIFTP and the Storage Element Service it represents the forth pillar that completes the Data Management System picture on LCG/gLite.

7.5.1. Basic Concepts

In order to understand the terminology currently used by the FTS community the following concepts are defined:

- ***Job:*** A transfer job consists of a set of files to be transferred in a source/destination pairs format. A job may optionally have parameters key/value pair block used for passing options to the underlying transport layer (gridFTP) and contains a MyProxy password used together with the client's subject to retrieve the user's proxy used for the transfer
- ***File:*** Refers to the source/destination physical name pairs to be transferred . Both source and destination should be defined in a Storage URL format, suitable for interaction with an SRM or in a Gridftp format, i.e. used by globus-url-copy
- ***Job State:*** the job's state is a function of whole individual File states constituting the Job
- ***File State:*** is the state of an individual file transfer
- ***Channel:*** is a specific network pipe used for transferring files. There are two distinguished types of channels:
  production channels, for ensuring bulk distribution of files within a production job. These channels are dedicated network pipe between Tier-0, Tier-1's and other major Tier-2's centers
  non production channels, assigned typically to open network;these channels do not assure production QOS

The jobs and their constituent files are processed asynchronously. Upon submission the client is handed a job identifier that can later be used to query the status of the job as it progresses through the system. The job identifier allows the user for querying the various statuses of his jobs and for canceling them. Once a job has been submitted to the system it is assigned a transfer channel based on the files that it containsand accordingly the VO, the site and the network provider policies.

Attention! FTS only deals with physical files and does not provide any facility for dealing with logical files or collection-like entities. This means that a user cannot expect FTS understands a task like: "*Put this LFN to RAL and registers the new replica on LFC*" which is rather supposed to be a task for an higher level service (File Placement Service). Intelligent job submission and reorganization for an optimal use of the network is targeted for higher level service as well. Each VO currently provides its own FPS-like service.

7.5.2. States

The possible states a job can assume are:

- ***Submitted*** The job has been submitted to FTS but not yet assigned to a channel
- ***Pending*** The job has been assigned to a channel and its files waiting for being transferred
- ***Active*** The transfer for some of the job's files is ongoing

- ***Canceling*** The job is going to be canceled
- ***Done*** All files in a job get transferred successfully
- ***Failed*** Some transfers of the files in a job are failed
- ***Canceled*** The job has been canceled
- ***Hold*** Job in this state require manual interventions because the state for some of the files cannot be resolved automatically (they might have been retried many times before this state)

The job's files can assume the states of the job they belong to with the exception of the "*wait*" state , indicating the file transfer has failed once and it is candidate for another try.

7.5.3. FTS Commands

Before submitting a job, the user is expected to upload an appropriate credential (the grid certificate) into the MyProxy server used by FTS. The upload has to be ade using the DN-as-username mode, since the DN of the client who submits the job is later used by FTS to retrieve the credential form MyProxy. This is achieved with the following options of the `myproxy-init` command.

```
$ myproxy-init -s myproxy-fts.cern.ch -d
```

The same password used is passed to FTS at the Job submission time.
The following user-level commands for submitting, querying and canceling jobs are described here:

`glite-transfer-submit`	Submits a data transfer job.
`glite-transfer-status`	Displays the status of an ongoing data transfer job
`glite-transfer-list`	Lists all submitted transfer jobs owned by that user.
`glite-transfer-cancel`	Cancels a file transfer job.

For matter of completeness the following administrative commands are also briefly described. Only few people within a VO, explicitly configured on the FTS server, are eligible to run them.

`glite-transfer-channel-add`	Creates a new channel with defined parameters on FTS.
`glite-transfer-channel-list`	Displays details of a given channel defined on FTS
`glite-transfer-channel-set`	Allows administrators to set a channel Active or Inactive
`glite-transfer-channel-signal`	Allows to change the status of all transfers of a given job (or of a given channel).

Man pages are available for all these commands.

Example 7.5.3.1 (Submitting a job to FTS)

Once a user has successfully registered his proxy to a MyProxy server he has to submit to FTS a transfer job. It can do it either by specifying the source-destination pair in the command line:

```
$glite-transfer-submit -m myproxy-fts.cern.ch  -s https://fts-sc.cr.cnaf.infn.it:8443/ \
 sc3infn/glite-data-transfer-fts/services/FileTransfer srm://srm.grid.sara.nl:8443/srm/ \
 managerv1?SFN=/pnfs/grid.sara.nl/data/lhcb/test/roberto/zz_zz.f \
 srm://sc.cr.cnaf.infn.it:8443/srm/managerv1?SFN=/castor/cnaf.infn.it/grid/lcg/lhcb/test/ \
 roberto/SARA_1.25354

Enter MyProxy password:

Enter MyProxy password again:

c2e2cdb1-a145-11da-954d-944f2354a08b
```

or by specifying all source-destination pairs in an input file (bulk submission). The `-m` option specifies the my-proxy server to use; the `-s` option specifies the FTS service endpoint to be contacted. If the service starts with **http://**, **https://** or **httpg://** it is taken as a direct service endpoint URL otherwise is taken as a service instance name and Service Discovery is invoked to look up the endpoints. If not specified the first available transfer service from the Service Discovery utility will be used. This is true for all subsequent examples.

```
$glite-transfer-submit -m "myproxy-fts.cern.ch"  -s https://fts-sc.cr.cnaf.infn.it:8443/sc3infn\
 /glite-data-transfer-fts/services/FileTransfer SARA-CNAF.in -p $passwd
```

where the input file `SARA-CNAF.in` looks like:

```
$cat SARA-CNAF.in
srm://srm.grid.sara.nl:8443/srm/managerv1?SFN=/pnfs/grid.sara.nl/data/lhcb/test/roberto/zz_zz.f \
 srm://sc.cr.cnaf.infn.it:8443/srm/managerv1?SFN=/castor/cnaf.infn.it/grid/lcg/lhcb/test/roberto/SARA_1
srm://srm.grid.sara.nl:8443/srm/managerv1?SFN=/pnfs/grid.sara.nl/data/lhcb/test/roberto/zz_zz.f \
 srm://sc.cr.cnaf.infn.it:8443/srm/managerv1?SFN=/castor/cnaf.infn.it/grid/lcg/lhcb/test/roberto/SARA_2
srm://srm.grid.sara.nl:8443/srm/managerv1?SFN=/pnfs/grid.sara.nl/data/lhcb/test/roberto/zz_zz.f \
 srm://sc.cr.cnaf.infn.it:8443/srm/managerv1?SFN=/castor/cnaf.infn.it/grid/lcg/lhcb/test/roberto/SARA_3
.....
```

The `$passwd` in the example is an environment variable opportunely set to the value of the password to be passed to FTS.
Attention! The transfers handled by FTS within a single job bulk-submission must all refer to the same channel otherwise FTS will not process such transfers and will return the message *Inconsistent channel*.

Example 7.5.3.2 (Querying the status of a job)
The following example shows a query to FTS for inferring information about the states of a transfer job.

```
$glite-transfer-status -s https://fts-sc.cr.cnaf.infn.it:8443/sc3infn/glite-data-transfer-fts\
 /services/FileTransfer -l c2e2cdb1-a145-11da-954d-944f2354a08b
Pending
  Source:      srm://srm.grid.sara.nl:8443/srm/managerv1?SFN=/pnfs/grid.sara.nl/data/lhcb/test/roberto/
  Destination: srm://sc.cr.cnaf.infn.it:8443/srm/managerv1?SFN=/castor/cnaf.infn.it/grid/lcg/lhcb/test/
  State:       Pending
  Retries:     0
  Reason:      (null)
  Duration:    0
```

Attention: The verbosity level of the status of a given job is set with the `-v` option; the individual transfer status is however available through the option `-l`.

Example 7.5.3.3 ***(Listing ongoing data transfers)***

The following example allows for querying all ongoing data transfers with specified (intermediate) state in a defined FTS service. The user willing to know ongoing transfer jobs on a channel has to specify instead the channel with `-c` option.

```
$glite-transfer-list -s s https://fts-sc.cr.cnaf.infn.it:8443/ \
 sc3infn/glite-data-transfer-fts/services/FileTransfer Pending
 |grep c2e2cdb1-a145-11da-954d-944f2354a08b

c2e2cdb1-a145-11da-954d-944f2354a08b Pending
```

Example 7.5.3.4 ***(Canceling a job)***

The use of the FTS user command for canceling a data transfer job previously submitted is shown there.

```
 glite-transfer-cancel -s https://fts-sc.cr.cnaf.infn.it:8443/sc3infn/glite-data-transfer-fts \
 /services/FileTransfer c2e2cdb1-a145-11da-954d-944f2354a08b
```

7.6. File and Replica Management Client Tools

While FTS is intended for transferring large amount of data (e.g. production data) in a reliable and asynchronous way, the average grid-user has to deal with his own (relatively) small data for private analysis. gLite 3.0 offers a variety of Data Management Client tools to upload/download files to/from the Grid, replicate data and locate the best replica available, and interact with the file catalogs. Every user should deal with data management through the LCG Data Management tools (usually referred to as ***lcg-utils*** or `lcg-*` commands). They provide a high level interface (both command line and APIs) to the basic DM functionality, hiding the complexities of catalog and SEs interaction. Furthermore, such high level tools ensure the consistency between Storage Elements and catalog in DM operations and try to minimize the risk of grid files corruption.

The same functionalities are exploited by the `edg-replica-manager` wrapper. More details on this are given below.

Some lower level tools (like `edg-gridftp-*` commands, globus-url-copy and srm-* dedicated commands) are also available. These low level tools are quite helpful in some particular cases (see examples for more details). Their usage however is strongly discouraged for non expert users, since such tools do not ensure consistency between physical files in the SE and entries in the file catalog and their usage might result very dangerous.

7.6.1. LCG Data Management Client Tools

The LCG Data Management tools (usually called ***lcg-utils***) allow users to copy files between UI, CE, WN and a SE, to register entries in the File Catalog and replicate files between SEs.

NOTE: Up to the LCG release 2.3.0, the `edg-replica-manager` command (also in abbreviated `edg-rm` form) provided the same functionality than the current lcg_utils offer. For performance reasons, the `edg-rm` was dismissed in favor of the lcg_utils. The current `edg-replica-manager` command is just a wrapper script around lcg_utils. In this way, it ensures the performance and functionalities of lcg_utils, maintaining the interface of the old Java CLI. More information about the `edg-replica-manager` wrapper script can be found in [43]

The name and functionality overview of the available commands is shown in the following table.

Replica Management

`lcg-cp`	Copies a Grid file to a local destination (download).
`lcg-cr`	Copies a file to a SE and registers the file in the catalog (LFC or LRC) (upload).
`lcg-del`	Deletes one file (either one replica or all replicas).
`lcg-rep`	Copies a file from one SE to another SE and registers it in the catalog (LFC or LRC) (replicate).
`lcg-gt`	Gets the TURL for a given SURL and transfer protocol.
`lcg-sd`	Sets file status to ”Done” for a given SURL in an SRM’s request.

File Catalog Interaction

`lcg-aa`	Adds an alias in the catalog (LFC or RMC) for a given GUID.
`lcg-ra`	Removes an alias in the catalog (LFC or RMC) for a given GUID.
`lcg-rf`	Registers in the the catalog (LFC or LRC/RMC), a file residing on an SE.
`lcg-uf`	Unregisters in the the catalog (LFC or LRC) a file residing on an SE.
`lcg-la`	Lists the aliases for a given LFN, GUID or SURL.
`lcg-lg`	Gets the GUID for a given LFN or SURL.
`lcg-lr`	Lists the replicas for a given LFN, GUID or SURL.

Each command has a different syntax (arguments and options), but the `--vo <vo_name>` option, to specify the virtual organization of the user, is present and mandatory in all the commands, except for `lcg-gt`, unless the environment variable `LCG_GFAL_VO` has been set; in this case the VO for the user is taken from the value of that variable[7].

The `--config <file>` option (allows to specify a configuration file) and the `-i` option (allows to connect insecurely to the File Catalog) are currently ignored.

Timeouts: The commands `lcg-cr`, `lcg-del`, `lcg-gt`, `lcg-rf`, `lcg-sd` and `lcg-rep` all have timeouts implemented. By using the option `-t`, the user can specify a number of seconds for the tool to time out. The default is 0 seconds, i.e.: no timeout. If a tool times out in the middle of operations, all actions performed till that moment are undone, so no broken files are left on a SE and not unexisting files are registered in the catalogs.

Environment:

- For all `lcg-*` commands to work, the environment variable `LCG_GFAL_INFOSYS` must be set to point to the IS provider (the BDII) in the format `hostname.domain:port`, so that the commands can retrieve the necessary information for their operation. Remember that the BDII read port is 2170.
- The endpoint(s) for the catalogs can also be specified (taking precedence over that published in the IS) through environmental variables: `LRC_ENDPOINT`, `RMC_ENDPOINT` for the RLS and `LFC_HOST` for the LFC. If no endpoints are specified, the ones published in the Information System are taken[8].
- If the variable `LCG_GFAL_VO` is set in the environment, then the `--vo` option is not required for the `lcg-*` commands, since they take the value of this variable.

In respect of authentication and authorization, lcg-utils manage data transfer securely through gsiftp For this reason, the user must have a valid proxy and must appear in the grid-mapfile of the SE in order to use `lcg-cr`, `lcg-cp`, `lcg-rep` and `lcg-del`.

On the other side, **the information in the LRC and RMC catalogs is not protected** (RLS allows insecure access) and no proxy certificate is required for the rest of `lcg-*` commands, which do not deal with physical replicas. Although this situation is different for the LFC, currently the information stored in the RLS file catalogs **can be altered by anyone**.

NOTE: The user will often need to gather information on the existing Grid resources in order to perform DM operations. For instance, in order to specify the destination SE for the upload of a file, the information about the available SEs must be retrieved in advance. There are several ways to retrieve information about the resources on the Grid. The Information System may be queried directly through the `ldapsearch` command or via the `lcg-info` wrapper, the `lcg-infosites` wrapper may be used, or a monitoring tool (e.g. a web page displaying information on Grid resources) can be checked. All these methods are described in Chapter 5.

In what follows, some usage examples are given. Most command can run in verbose mode (`-v` or `--verbose` option). For details on the options of each command, please use the man pages of the commands.

Example 7.6.1.1 (Uploading a file to the Grid)

[7]This variable can also be used by GFAL when resolving LFNs and GUIDs, as described in Appendix F

[8]As discussed in 7.4.1, these is not true for the LFC interaction commands (`lfc-*`), which require that the `LFC_HOST` variable is defined explicitly.

In order to upload a file to the Grid; i.e.: to transfer it from the local machine to a Storage Element and register it in the catalog, the `lcg-cr` command (which stands for copy and register) can be used:

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch file:/home/antonio/file1
> guid:6ac491ea-684c-11d8-8f12-9c97cebf582a
```

where the only argument is the local file to be uploaded (a fully qualified URI) and the `-d <destination>` option indicates the SE used as the destination for the file. The command returns the unique GUID.

If no destination is given, the SE specified by the `VO_<VO-name>_DEFAULT_SE` environmental variable is taken. Such variable should be set in all WNs and UIs.

The `-P` option allows the user to specify a relative path name for the file in the SE. The absolute path is built appending the relative path to a root directory which is VO and SE specific and is determined through the Information System. If no `-P` option is given, the relative path is automatically generated following a certain schema.

There is also the possibility to specify the destination as a complete SURL, including SE hostname, the path, and a chosen filename. The action will only be allowed if the specified path falls under the user's VO directory.

The following are examples of the different ways to specify a destination:

```
-d lxb0710.cern.ch
-d sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/my_file
-d lxb0710.cern.ch -P my_dir/my_file
```

The option `-l <lfn>` can be used to specify a LFN:

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch -l lfn:my_alias1 file:/home/antonio/file1
> guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
```

REMINDER: If the RLS catalog is used, the LFN takes the form `lfn:<someLFN>` where `<someLFN>` can be any string. In the case that the LFC is used, the LFNs are organized to a hierarchical namespace (like UNIX directory trees). So the LFN will take the form `lfn:/grid/<voname>/<dir1>/...`. Remember that subdirectories in the name-space are **not** created automatically by `lcg-cr` and you should manage them yourself through the `lfc-mkdir` and `lfc-rmdir` command line tools described in the previous section.

The `-g` option allows to specify a GUID (otherwise automatically created):

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch \
-g guid:baddb707-0cb5-4d9a-8141-a046659d243b file:`pwd`/file2
> guid:baddb707-0cb5-4d9a-8141-a046659d243b
```

Attention! This option should not be used except for expert users and in very particular cases. Because the specification of an existing GUID is also allowed, a misuse of the tool may end up in a corrupted GRID file in which replicas of the same file are in fact different from each other.

Finally, in this and other commands, the `-n <#streams>` options can be used to specify the number of parallel streams to be used in the transfer (default is one).

Known problem: When multiple streams are requested, the GridFTP protocol establishes that the GridFTP server must open a new connection back to the client (the original connection, and only one in the case of one stream, is opened from the client to the server). This may become a problem when a file is requested from a WN and this WN is firewalled to disable inbound connections (which is usually the case). The connection will in this case fail and the error message returned (in the logging information of the job performing the data access) will be ”`425 can't open data connection`”.

Example 7.6.1.2 (Replicating a file)

Once a file is stored on an SE and registered within the catalog, the file can be replicated using the `lcg-rep` command, as in:

```
$ lcg-rep -v --vo dteam -d lxb0707.cern.ch guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24

> Source URL: sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb4
6-2214-4db8-9ee8-2930de1a6bef
File size: 30
Destination specified: lxb0707.cern.ch
Source URL for copy: gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/f
ile0dcabb46-2214-4db8-9ee8-2930de1a6bef
Destination URL for copy: gsiftp://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07
-09/file50c0752c-f61f-4bc3-b48e-af3f22924b57
# streams: 1
Transfer took 2040 ms
Destination URL registered in LRC: sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2
004-07-09/file50c0752c-f61f-4bc3-b48e-af3f22924b57
```

where the file to be replicated can be specified using a LFN, GUID or even a particular SURL, and the `-d` option is used to specify the SE where the new replica will be stored. This destination can be either an SE hostname or a complete SURL, and it is expressed in the same format as with `lcg-cr`. The command also admits the `-P` option to add a relative path to the destination (as with `lcg-cr`).

For one GUID, there can be only one replica per SE. If the user tries to use the `lcg-rep` command with a destination SE that already holds a replica, the command will exit successfully, but no new replica will be created.

Example 7.6.1.3 (Listing replicas, GUIDs and aliases)

The lcg-lr command allows users to list all the replicas of a file that have been successfully registered in the File Catalog:

```
$ lcg-rep --vo dteam lfn:my_alias1
```

```
> sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-09/file79aee616-6cd7-4b75-
8848-f09110ade178
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-
9ee8-2930de1a6bef
```

Again, LFN, GUID or SURL can be used to specify the file for which all replicas must be listed. The SURLs of the replicas are returned.

The `lcg-lg` command (list GUID) returns the GUID associated with a specified LFN or SURL:

```
$ lcg-lg --vo dteam sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-09/file7
9aee616-6cd7-4b75-8848-f09110ade178
> guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
```

The `lcg-la` command (list aliases) can be used to list the LFNs associated with a particular file, which can be, identified by its GUID, any of its LFNs, or the SURL of one of its replicas.

```
$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
> lfn:my_alias1
```

The `lfc-*` commands for LFC and the tools `edg-lrc` and `edg-rmc` for the RLS offer more functionalities for catalog interaction, although the ones provided by the `lcg-*` commands should be enough for a normal user.

Example 7.6.1.4 (Copying files out of the Grid)

The `lcg-cp` command can be used to copy a Grid file to a non-grid storage resource. The first argument (source file) can be a LFN, GUID or one SURL of a valid Grid file, the second argument (destination file) must be a local filename or valid TURL. In the following example, the verbose mode is used and a timeout of 100 seconds is specified:

```
$ lcg-cp --vo dteam -t 100 -v lfn:/grid/dteam/hosts file:/tmp/f2
Source URL: lfn:/grid/dteam/hosts
File size: 104857600
Source URL for copy:
gsiftp://lxb2036.cern.ch/storage/dteam/generated/2005-07-17/fileea15c9c9-abcd-4e9b-8724-1
ad60c5afe5b
Destination URL: file:///tmp/f2
# streams: 1
# set timeout to  100 (seconds)
     85983232 bytes   8396.77 KB/sec avg   9216.11
Transfer took 12040 ms
```

Notice that although this command is designed to copy files from a SE to non-grid resources, if the proper TURL is used (using the `gsiftp:` protocol), a file could be transferred from one SE to another, or from out of the Grid to a SE. **This should not be done**, since it has the same effect as using `lcg-rep` BUT **skipping the file registration**, making in this way this replica invisible to Grid users.

Example 7.6.1.5 (Obtaining a TURL for a replica)

The `lcg-gt` allows to get a TURL from a SURL and a supported protocol. The command behaves very differently if the Storage Element exposes an SRM interface or not. The command always returns three lines of output: the first is always the TURL of the file, the last two are meaningful only in case of SRM interface.

- In case of classic SE or a MSS without SRM interface, the command obtains the TURL by simple string manipulation of the SURL (obtained from the File Catalog) and the protocol (checking in the Information System if it is supported by the Storage Element). No direct interaction with the SE is involved. The last two lines of output are always zeroes.

  ```
  $ lcg-gt sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb4
  6-2214-4db8-9ee8-2930de1a6bef gsiftp
  > gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-22
  14-4db8-9ee8-2930de1a6bef
  > 0
  > 0
  ```

  Be aware that in case of MSS, the file could be not staged on disk but only stored on tape. For this reason, an operation like

  ```
  $ lcg-cp gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dca
  bb46-2214-4db8-9ee8-2930de1a6bef file://tpm/somefile.txt
  ```

  could hang forever, waiting for the file to be staged (a timeout mechanism is not implemented in lcg-utils).

- In the case of SRM interface, the TURL is returned to the `lcg-gt` command by the SRM itself. In case of MSS, the file will be staged on disk (if not present already) before a valid TURL is returned. It could take `lcg-gt` quite a long time to return the TURL (depending on the conditions of the stager) but a successive `lcg-cp` of such TURL will not hang. This is one of the reasons for which a SRM interface is desirable for all MSS.

  The second and third lines of output represent the requestID and fileID for the `srm_put` request (hidden to the user) which will remain open unless explicitly closed (at least in SRM v1 currently deployed). It is important to know that some SRM Storage Elements are limited in the maximum number of open requests. Further requests will fail, once this limit has been reached. It is therefore good practice to close the request once the TURL is not needed anymore. This can be done with the `lcg-sd` command which needs as arguments the TURL of the file, the requestID and fileID.

  ```
  $ lcg-gt srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/generated/2005-04-12/file
  fad1e7fb-9d83-4050-af51-4c9af7bb095c gsiftp
  > gsiftp://castorgrid.cern.ch:2811//shift/lxfsrk4705/data02/cg/stage/filefad1e7fb-9d
  83-4050-af51-4c9af7bb095c.43309
  > -337722383
  > 0
  ```

```
[ ... do something with the TURL ... ]

$ lcg-sd gsiftp://castorgrid.cern.ch:2811//shift/lxfsrk4705/data02/cg/stage/filefad1
e7fb-9d83-4050-af51-4c9af7bb095c.43309 -337722383 0
```

Example 7.6.1.6 (Deleting replicas)

A file that is stored on a Storage Element and registered with a catalog can be deleted using the `lcg-del` command. If a SURL is provided as argument, then that particular replica will be deleted. If a LFN or GUID is given instead, then the `-s <SE>` option must be used to indicate which one of the replicas must be erased, unless the `-a` option is used, in which case all replicas of the file will be deleted and unregistered (on a best-effort basis). If all the replicas of a file are removed, the corresponding GUID-LFN mappings are removed as well.

```
$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file78ef5a13-166f-4701-
8059-e70e397dd2ca
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file21658bfb-6eac-409b-
9177-88c07bb1a57c

$ lcg-del --vo=dteam -s lxb0707.cern.ch guid:91b89dfe-ff95-4614-bad2-c538bfa28fac

$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file21658bfb-6eac-409b-
9177-88c07bb1a57c

$ lcg-del --vo dteam -a guid:91b89dfe-ff95-4614-bad2-c538bfa28fac

$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> lcg_lr: No such file or directory
```

The last error indicates that the GUID is no longer registered within the catalogs of LCG-2, as the last replica was deleted.

Example 7.6.1.7 (Registering and unregistering Grid files)

The `lcg-rf` (register file) command allows to register a file physically present in a SE, creating a GUID-SURL mapping in the catalog. The `-g <GUID>` allows to specify a GUID (otherwise automatically created).

```
$ lcg-rf -v --vo dteam -g guid:baddb707-0cb5-4d9a-8141-a046659d243b sfn://lxb0710.cern.ch/
flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1a6bef
> guid:baddb707-0cb5-4d9a-8141-a046659d243b
```

Likewise, `lcg-uf` (unregister file) allows to delete a GUID-SURL mapping (respectively the first and second argument of the command) from the catalog.

```
$ lcg-uf --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b sfn://lxb0710.cern.ch/flatfil
es/SE00/dteam/generated/2004-07-12/file04eec6b2-9ce5-4fae-bf62-b6234bf334d6
```

If the last replica of a file is unregistered, the corresponding GUID-LFN mapping is also removed.

WARNING: `lcg-uf` just removes entries from the catalog, it does not remove any physical replica from the SE. Watch out for consistency.

Example 7.6.1.8 (Managing aliases)

The `lcg-aa` (add alias) command allows the user to add a new LFN to an existing GUID:

```
$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
> lfn:my_alias1

$ lcg-aa --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b lfn:my_new_alias

$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
> lfn:my_alias1
> lfn:my_new_alias
```

Correspondingly, the `lcg-ra` command (remove alias) allows a user to remove an LFN from an existing GUID:

```
$ lcg-ra --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b lfn:my_alias1

$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
> lfn:my_new_alias
```

In order to list the aliases of a file, the `lcg-la` command, discussed previously, has been used.

7.6.2. Low Level Data Management Tools: GSIFTP

The following low lever user tools are intended for being used against GSIFTP servers on SEs.

`edg-gridftp-exists URL`	Checks the existence of a file or directory on a SE.
`edg-gridftp-ls URL`	Lists a directory on a SE.
`edg-gridftp-mkdir URL`	Creates a directory on a SE.
`edg-gridftp-rename sourceURL destURL`	Renames a file on a SE.
`edg-gridftp-rm URL`	Removes a file from a SE.
`edg-gridftp-rmdir URL`	Removes a directory on a SE.
`globus-url-copy sourceURL destURL`	Copies files between SEs.

The commands `edg-gridftp-rename`, `edg-gridftp-rm`, and `edg-gridftp-rmdir` should be used with extreme care and only in case of serious problems. In fact, these commands do not interact with any of the catalogs and therefore they can compromise the consistency/coherence of the information contained in the Grid. Also `globus-url-copy` is dangerous since it allows the copy of a file into the Grid without enforcing its registration.

All the `edg-gridftp-*` commands accept `gsiftp` as the only valid protocol for the TURL.

Some usage examples are shown. They are by no means exhaustive. To obtain help on these commands use the option `--usage` or `--help`. General information on gridFTP is available in [38].

Example 7.6.2.1 (Listing and checking the existence of Grid files)

The `edg-gridftp-exists` and `edg-gridftp-ls` commands can be useful in order to check if a file is physically in a SE, regardless of its presence in the Grid catalogs.

```
$ lcg-lr --vo dteam guid:27523374-6f60-44af-b311-baa3d29f841a
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/file42ff7086-8063-414d-
9000-75c459b71296

$ edg-gridftp-exists gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/f
ile42ff7086-8063-414d-9000-75c459b71296

$ edg-gridftp-exists gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/my
_fake_file
> error gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/my_fake_file
does not exist


$ edg-gridftp-ls gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13
> file42ff7086-8063-414d-9000-75c459b71296
```

Example 7.6.2.2 (Copying a file with `globus-url-copy`)

The `globus-url-copy` command can be used to copy files between any two Grid resources, and from/to a non-grid resource. Its functionality is similar to that of `lcg-cp`, but source and destination must be specified as TURLs.

```
globus-url-copy gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/file42
ff7086-8063-414d-9000-75c459b71296 file://`pwd`/my_file
```

7.6.3. Low Level Data Management Tools: SRM

The following commands, originally intended for interacting with SRM servers running upon d-cache storages, offer a general way for interacting with any SRM server on LCG/gLite.

`srmcp`	Copies a file on a SRM SE.
`srm-get-metadata`	Gathers information about a file stored into a SRM SE.
`srm-advisory-delete`	Remove a file from a SRM storage.
`srm-storage-element-info`	Retrieves information about a SRM managed storage.
`srm-get-request-status`	Retrieves status about a given requestID

All these commands do not come with man pages available even though basic functionalities can be displayed using the option `-h`.

Examples of their use are thereafter described.

Example 7.6.3.1 (Copy a file to a SRM Storage)

The srmcp command allows for copying files from a local disk to a SRM storage or between two SRM storage elements (two party or third party transfers). It can be invoked either by specifying the SRM URLs source/destination pair or by passing a list of URLs pairs in a files through the `-copyjobfile` option. It can also accept gsiftp TURLs. The following example shows how to correctly invoke the command for copying a local file to a remote SRM storage.

```
$srmcp file:///‘pwd‘/zz_zz.f srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb/zz_zz
```

Please note that there are **four** slashes between file protocol and the input file expressed in absolute path. The output looks like:

```
>...
execution of CopyJob, source = file:////afs/cern.ch/user/s/santinel/zz_zz.f destination =
gsiftp://srm.grid.sara.nl:2811//pnfs/grid.sara.nl/data/lhcb/zz_zz completed
setting file request -2147196960 status to Done
doneAddingJobs is true
copy_jobs is empty
stopping copier
```

which clearly indicates that the file has been copied successfully. The command returns 0 for success, 1 for a general error, 2 for an existing file that can not be overwritten and 3 for user permission error.

Example 7.6.3.2 (Retrieving information about a file)

`srm-get-metadata` is a useful command for grid users to get the metadata of SRM stored data. The following example shows how to infer such information.

```
$srm-get-metadata srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/zz_zz
> [...]
SRMClientV1 :  getFileMetaData, contacting service httpg://srm.grid.sara.nl:8443/srm/managerv1
FileMetaData(srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/zz_zz)=
RequestFileStatus         SURL :srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/zz_zz
                     size :12553
                     owner :1781
                     group :1164
                     permMode :420
                     checksumType :adler32
                     checksumValue :65aa6b8f
                     isPinned :false
                     isPermanent :true
                     isCached :true
                     state :
                     fileId :0
                     TURL :
                     estSecondsToStart :0
                     sourceFilename :
                     destFilename :
                     queueOrder :0
```

Attention: On d-cache, the `srmcp` command grants by default read and write permissions to any member of the group the user belongs to. Nobody else is granted to access the file. The globus-url-copy has the same behavior. On CASTOR based storages `srmcp` sets read-write access to the owner and to the members of the same group and read access to the others. Globus-url-copy by default sets read-write access to the owner and only read access to both members of the group and the others.

Example 7.6.3.3   (Removing files from SRM: CASTOR Vs d-cache)

The current interface of SRM in LCG/gLite does not offer a functionality for physical file deletion in a homogeneous way with respect of the back end (CASTOR,dcache,DPM).

The srm-advisory-delete command is currently used for removing files om SRM based storages. Its behavior is however depending on the configuration and settings of the storage it self. By design, this command is not intended for this purpose but rather for asking SRM server to prepare for deleting a given file. Users willing to scratch their files on the grid achieve different results by issuing the command against different implementations of SRM. The command works as expected if run against d-cache and DPM storages. It removes the entry from the name space of the storage server which, will take care of removing physical entries on its disk pools. If `srm-advisory-delete` is however run against CASTOR, the file is kept on the name space and not physical removal is done on the disk pools. Any file is however over writable in CASTOR.

The following examples help understanding this difference between the d-cache and CASTOR SRM imple-

mentations.
The file zz_globus is present on the name space of a CASTOR storage.

```
$ edg-gridftp-ls -v gsiftp://castorsrm.cern.ch/castor/cern.ch/grid/lhcb

>-rw-rw-r--   1 lhcb001  z5                    2317 Aug 15  2005 test-acsmith-009.dat
>mrw-r--r--   1 lhcb001  z5                   12553 Feb 20 12:02 zz_globus
> -rw-rw-r--   1 lhcb001  z5                   12553 Nov 04 14:00 zz_zz.f
```

The `srm-get-metadata` returns the following information:

```
$ srm-get-metadata srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus
>....
SRMClientV1 :  getFileMetaData, contacting service httpg://castorgrid06.cern.ch:8443/srm/managerv1
FileMetaData(srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus)=
RequestFileStatus         SURL :srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus
                     size :12553
                     owner :lhcb001
                     group :z5
                     permMode :33188
                     checksumType :null
                     checksumValue :null
                     isPinned :false
                     isPermanent :true
                     isCached :false
                     state :
                     fileId :0
                     TURL :
                     estSecondsToStart :0
                     sourceFilename :
                     destFilename :
```

The `srm-advisory-delete` command for deleting the entry zz_globus **does not** work in this case:

```
$ srm-advisory-delete srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus
```

The file is not removed as `srm-get-metadata` is still reporting the same information:

```
$ srm-get-metadata srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus
> ...
SRMClientV1 :  getFileMetaData, contacting service httpg://castorgrid01.cern.ch:8443/srm/managerv1
FileMetaData(srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus)=
RequestFileStatus         SURL :srm://castorsrm.cern.ch:8443/castor/cern.ch/grid/lhcb/zz_globus
```

```
                    size :12553
                    owner :lhcb001
                    group :z5
                    permMode :33188
                    checksumType :null
                    checksumValue :null
                    isPinned :false
                    isPermanent :true
                    isCached :false
                    state :
                    fileId :0
                    TURL :
                    estSecondsToStart :0
                    sourceFilename :
                    destFilename :
                    queueOrder :0

$ edg-gridftp-ls -v gsiftp://castorsrm.cern.ch/castor/cern.ch/grid/lhcb

>-rw-rw-r--   1 lhcb001  z5                     2317 Aug 15  2005 test-acsmith-009.dat
>mrw-r--r--   1 lhcb001  z5                    12553 Feb 20 12:02 zz_globus
> -rw-rw-r--   1 lhcb001  z5                    12553 Nov 04 14:00 zz_zz.f
```

Attention! On CASTOR based storages the space occupied by a file is freed only through the command `edg-gridftp-rm` or the native CASTOR command (`nsrm`).

```
$ edg-gridftp-rm gsiftp://castorsrm.cern.ch/castor/cern.ch/grid/lhcb/zz_globus
$ edg-gridftp-ls -v gsiftp://castorsrm.cern.ch/castor/cern.ch/grid/lhcb
>-rw-rw-r--   1 lhcb001  z5                     2317 Aug 15  2005 test-acsmith-009.dat
> -rw-rw-r--   1 lhcb001  z5                    12553 Nov 04 14:00 zz_zz.f
```

By issuing `srm-advisory-delete` versus a d-cache storage the user can however also remove multiple files in one single operation (bulk removal). The following example shows the content of a directory on a remote d-cache storage:

```
$ edg-gridftp-ls -v gsiftp://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb
>drw                 512         test
>drw                 512         sc3
>-rw                 12553       from_CNAF
>drw                 512         production
>-rw                 12553       zz_zzRAL.f
>drw                 512         disk
>-rw                 12553       gsi_zz
>-rw                 12553       zz_zz
>drw                 512         user
>-rw                 12553       zz_zzPIC.f
```

```
>drw                512         generated
>-rw                12553       zz_zz.f
>-rw                12553       zz_zz2.f
```

invoking srm-advisory-delete for two of these entries:

```
$ srm-advisory-delete srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/gsi_zz \
   srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/zz_zz
```

the files are removed successfully

```
$ srm-get-metadata srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/gsi_zz \
   srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lhcb/zz_zz
```

as it is shown in the output of the srm-get-metadata command for both of these entries.

```
>SRMClientV1 : getFileMetaData failed to parse SURL: java.lang.RuntimeException: could not get storage
>CacheException(rc=666;msg=Pnfs error : can't get pnfsId (not a pnfsfile))
>SRMClientV1 : copy: try again
```

Example 7.6.3.4 ***(Retrieving info about a Storage Element)***

The command `srm-storage-element-info` provided with a valid endpoint-wsdl-url allows user for retrieving information about the status of a storage system managed by SRM.

```
$srm-storage-element-info httpg://srm.grid.sara.nl:8443/srm/infoProvider1_0.wsdl
StorageElementInfo :
                     totalSpace     =5046586572800 (4928307200 KB)
                     usedSpace      =1933512144967 (1888195454 KB)
                     availableSpace =3476820574537 (3395332592 KB)
```

7.7. Job Services and Data Management

With both LCG and gLite WMS, some specific JDL attributes allow the user to specify requirements on the input data.

Example 7.7.1 (Specifying input data in a job)

If a job requires one or more input file stored in an LCG Storage Element, the `InputData` JDL attribute list can be used. Files can be specified by both by LFN and GUID.

An example of JDL specifying input data looks like:

```
Executable = "/bin/hostname";
StdOutput = "sim.out";
StdError = "sim.err";
DataCatalog = "http://lfc-lhcb-ro.cern.ch:8085";
InputData = {"lfn:/grid/lhcb/test_roberto/preproduction"};
DataAccessProtocol = {"rfio", "gsiftp", "gsidcap"};
OutputSandbox = {"sim.err","sim.out"};
```

The `InputData` field may also be specified through guids. This attribute is used only during the match making process (to match CEs and SEs). It has nothing to do with the real access to files that the job can do while running. However, it is obviously reasonable that files listed in the attribute are really accessed by the job and vice-versa.

The `DataAccessProtocol` attribute is used to specify the protocols that the application can use to access the file and is mandatory if `InputData` is present. Only data in SEs which support one or more of the listed protocols are considered. The Resource Broker will schedule the job to a *close* CE to the SE holding the largest number of input files requested. In case several CEs are suitable, they will be ranked according to the ranking expression.

Warning A Storage Element or a list of Storage Elements is published as “close” to a given CE via the GlueCESEBindGroupSEUniqueID attribute of the GlueCESEBindGroup object class. For more information see Appendix G

Warning: Sometimes using the `InputData` attribute might result to be a bad idea: if many jobs need to access a single file and a unique replica of such file exists in the Grid, all jobs will land at the site containing the replica. Very soon all the CPUs of the site will be filled and jobs will be put in waiting state. It would be more efficient to schedule the job in a site with free CPUs and replicate the file in the *close SE* of the site.

Example 7.7.2 (Specifying a Storage Element)

With the LCG RB the user can ask the job to run close a specific Storage Element, in order to store there the output data, using the attribute `OutputSE`.

For example:

```
OutputSE = "castorsrm.cern.ch";
```

The Resource Broker will abort the job if there is no CE close to the `OutputSE` specified by the user.

NOTE: The same is not true with the gLite RB. Even if there are CEs close to a given `OutputSE` specified by the user, no resources get matched when this field is defined on the JDL.

Example 7.7.3 (Automatic upload and registration of output files)

The `OutputData` attribute list allows the user to automatically upload and register files produced by the job on the WN. For each file, three attributes can be set:

- The `OutputFile` attribute is mandatory and specifies the name of the generated file to be uploaded to the Grid.
- The `StorageElement` attribute is an optional string indicating the SE where the file should be stored, if possible. If unspecified, the WMS automatically choses a SE defined as close to the CE.
- The `LogicalFileName` attribute (also optional) represents a LFN the user wants to associate to the output file.

The following code shows an example of JDL requiring explicitely an `OutputData` attribute:

```
Executable    = "test.sh";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"test.sh"};
OutputSandbox = {"std.out","std.err",".BrokerInfo"};
OutputData = {
[
OutputFile="my_out";
LogicalFileName="lfn:/grid/lhcb/test_roberto/outputdata";
StorageElement = "castorsrm.pic.es";
]
};
```

Once the job is terminated and the user retrieve his output, the Output Sandbox downloaded will contain a further file, automatically generated by the JobWrapper, containing the logs of the output upload.

```
$ cat DSUpload_ZboHMYWoBsLVax-nUCmtaA.out
#
# Autogenerated by JobWrapper!
#
# The file contains the results of the upload and registration
# process in the following format:
# <outputfile> <lfn|guid|Error>


my_out    guid:2a14e544-1800-4257-afdd-7031a6892ef7
```

Warning: This mechaism for automatic upload of a specified output by using the *OutputData* is not longer supported by the gLite WMS.

Example 7.7.4 (Selecting the file catalog to use for match making)

In order for the RB to select CEs that are close to the files required by a job (by the `InputData` attribute), it has to locate the SEs where these files are stored. In order to do this, the RB uses the Data Location Interface service, which in turn contacts some file catalog. Since it is possible that several different file catalogs exist on the Grid, the user has the possibility to select which one he wants to talk to by using the JDL attribute `DataCatalog`. The user will specify the attribute and the endpoint of the catalog as value, as in this example:

```
DataCatalog = "http://lfc-lhcb-ro.cern.ch:8085/";
```

If no value is specified, then the first catalog supporting the DLI interface that is found in the information system is used (which should probably be the right choice for the normal user).

NOTE: Only catalogs that present a DLI interface can be talked to by the DLI service. Currently, the RLS catalog does not provide such interface, and so in order to use this catalog, the RB is configured to interface directly with it (without using the DLI). Thus, summarizing, the RB can be configured to either talk to the RLS catalog or to use the DLI. In the second possibility, the user has the option to manually specify the endpoint of the catalog he wants to use.

7.8. ACCESSING FILES ON GLITE

7.8.1. VOMS interaction

While the WMS relies on the LCMAPS mechanism for getting local UID/GID on the user voms proxy presented basis, the DMS implements the concept of Virtual Ids. They are site independant identifiers and do not require any administrator action. A DN is mapped to a a virtual uid and a VOMS group or role to a virtual gid. They are currently used by LFC and DPM. Only te virtual uid and the primary virtual gid (i.e. the ID mapped to the primary group as intended in [chapter for VOMS]) are used to control access to the File Catalog entries or the file on the Storage Elements.
Unless changed with *chown* the files are owned by the DN of the user who created the file; the group ownership depends on the *S_ISGID* value of the parent directory: if it is set, the file group ownership is the same as the parent one, if not the primary group of the user is used.

The LCG Data Managements is also designed for supporting secondary groups. Secondary groups are useful as explained by this example: Let's suppose a given user dressing the "production" role needs to access some collective files whose group ownership is "lhcb". Let suppose these files are not world readable. In this case, if secondary groups are not supported (and then the user is not seen from the "lhcb" group), - unless not explicitely set by an accessing ACLs giving "lhcb/Role=production" as supplementary group - the user cannot access these files.

7.8.2. Posix Access Control Lists

There are two different “level” of ACLs: base and extended. The former map directly to standard Unix permission (owner,group owner, others), the latter correspond to lists of supplementary users/groups. The existence of supplementary users/groups does require also an ACL mask be defined. Two different types of extended ACLs: access and default:

- access ACLs: can be set on both files and directories and are used to control the access
- default ACLs can be set on directories and are inhereted as access ACLs by every file or sub-directory underneath unless explicitely specified differently. The default ACLs are also inhereted as default ACLs by every sub-directory. LFC and any SE (HPSS) with SRM v2 interface support Posix ACLs.

7.8.3. Protection of files

Permission to delete a file depends on the *S_ISVTX* bit in the parent directory: one always needs to have write permission on the parent directory. Furthermore, if the *S_ISVTX* bit in the parent directory is set, one further needs to be the owner of the file or write permission on the files.

Hint In case of user analysis files, in order to guarantee privacy, it is suggested to set the S_ISVTX bit on parent directory and the mode of the file to 0644 or even to 0600 if the file must only be seen by the owner (DN). Conversely, in case of production files, the bit of the parent directory must be set and the permission mode to 0644 (the file must been accessible at least by the members of the group).

A special VOMS role “lhcbdataadmin” can finally be defined to grant users with this role, accessing all data of the Virtual Organization. This can be realized in two different ways:

a. An ACL entry is opportunely added to every file or directory to give the role ”lhcbodataadmin” all permissions on them.
b. The role “lhcbdataadmin” can be recognized by all LCG Data Management services as being a special role to get all permissions on files or directories owned by the VO (without using ACLs).

7.9. POOL AND LCG-2

The ***Pool Of persistent Objects for LHC (POOL)*** tool is used by most of the LHC experiments as a common persistency framework for the LCG application area. It is through POOL that they store their data.

Objects created by users using POOL are stored into its own File Catalog (XML Catalog). In order to be able to operate in the Grid, it is certainly very important for these experiments to have an interaction between the POOL catalog and the LCG-2 file catalogs.

Currently, there is a satisfactory interoperability between the POOL catalogs and the RLS catalogs. That is, there is a way to migrate POOL catalog entries to the RLS catalogs (i.e.: register those files within LCG-2) and also the LCG-2 Data Management tools can access those files as with any other Grid file. A way to achieve the same kind of interoperability between POOL and the LFC has not been implemented yet.

Example 7.9.1 (Migration from POOL(XML) to LCG(RLS))

We assume that the user has used POOL and as result has created a file which has been registered into the XML catalog of POOL. Now the point is how to register this file into the LCG catalog, the RLS.

In the first place, it is necessary to obtain the connection to the RLS catalog. A contact string has to be specified through the environment variable `POOL_CATALOG` as follows:

```
$ export POOL_CATALOG=edgcatalog_http://<host>:<port>/<path>   (bash shell)
$ setenv POOL_CATALOG edgcatalog_http://<host>:<port>/<path>   (csh shell)
```

For example into LCG-2 this environment variable may be set to:

```
$ export POOL_CATALOG=edgcatalog_http://rlscert01.cern.ch:7777/$VO/v2.2
/edg-local-replica-catalog/services/edg-local-replica-catalog
```

In the case that the user has specified the file as a SURL into POOL, he can assign it an LFN with POOL as follows:

```
$ FCregisterLFN -p <SURL> -l <LFN>
```

Now the user can make some test to check whether the file is into the LRC with the RLS client:

```
$ edg-lrc mappingsByPfn <SURL> --endpoint <LRC>
```

Or into the RMC:

```
$ edg-rmc mappingsByAlias <LFN> --endpoint <RMC>
```

Finally he can check if the Data Management tools are able to find the file:

```
$ lcg-lr --vo <VO> lfn:<LFN>
```

Note that in case that the POOL user has defined the SURL entry following a ROOT format, he must use the command `FCrenamePFN` to create a SURL entry compatible with the RLS catalog.

A complete list of POOL commands can be found into [42]. And in a machine where the interface is installed, the user can see them just by typing `FC<tab>` (or `FC<Ctrl-D>`, if that does not work).

APPENDIX A THE GRID MIDDLEWARE

The operating systems supported by LCG-2 are Red Hat 7.3 and Scientific Linux 3, while the supported architectures are IA32 and IA64.

The LCG-2 middleware layer uses components from several Grid projects, including DataTag (EDT), DataGrid (EDG), EGEE, INFN-GRID, Globus and Condor.

In some cases, LCG patches are applied to the components, so the final software used is not exactly the same as the one distributed by the original project.

The components which are currently used in LCG-2 are listed in table 1.

Component	LCG	EGEE	EDG	EDT	INFN-GRID	Globus	Condor	Other
Basic middleware								
Globus 2.4.3						√		
ClassAds 0.9.4							√	
Security								
MyProxy								√
VO management								
LDAP-based			√					
VOMS	√	√	√					
Workload management								
Condor/Condor-G 6.6.5							√	
EDG WMS	√		√					
Data management								
Replica Manager	√		√					
Replica Location Service			√			√		
LCG File Catalog	√						√	
Disk Pool Manager	√							
GFAL	√							
LCG DM tools	√							
Fabric management								
LCFG	√		√					√
Quattor	√		√					
YAIM	√							
LCAS/LCMAPS			√					
Monitoring								
GridICE					√			
Information system								
MDS						√		
Glue Schema				√				√
BDII	√							
R-GMA		√	√					
LCG Information tools	√							

Table 1: Software components of LCG-2 and projects that contributed to them.

APPENDIX B CONFIGURATION FILES AND VARIABLES

Some of the configuration files and environmental variables that may be of interest for the Grid user are listed in the following tables. Unless explicitly stated, they are all located/defined in the User Interface.

Environmental variables:

Variable	Notes
`$EDG_LOCATION`	Base of the installed EDG software.
`$EDG_TMP`	Temporary directory.
`$EDG_WL_JOBID`	Job id (defined for a running job). **In a WN.**
`$EDG_WL_LIBRARY_PATH`	Library path for EDG's WMS commands.
`$EDG_WL_LOCATION`	Base of the EDG's WMS software.
`$EDG_WL_PATH`	Path for EDG's WMS commands.
`$EDG_WL_RB_BROKERINFO`	Location of the .BrokerInfo file. **In a WN.**
`$EDG_WL_UI_CONFIG_VAR`	May be used to specify a configuration different that `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`.
`$EDG_WL_UI_CONFIG_VO`	May be used to specify a configuration different that `$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf`.
`$LCG_CATALOG_TYPE`	Type of file catalog used (`edg` or `lfc`) for lcg-utils and GFAL.
`$LCG_GFAL_INFOSYS`	Location of the BDII for lcg-utils and GFAL.
`$LCG_GFAL_VO`	May be used to tell lcg-utils or GFAL about a user's VO. **To set in a UI or ship with a job's JDL.**
`$LFC_HOST`	Location of the LFC catalog (only for catalog type `lfc`).
`$LCG_LOCATION`	Base of the installed LCG software.
`$LCG_RFIO_TYPE`	Type of RFIO (secure or insecure) to use by GFAL.
`$LCG_TMP`	Temporary directory.
`$LRC_ENDPOINT`	May be used to define th LRC endpoint for lcg-utils and GFAL (overriding the IS).
`$RMC_ENDPOINT`	May be used to define the RMC endpoint for lcg-utils and GFAL (overriding the IS).
`$VO_<VO_name>_DEFAULT_SE`	Default SE defined for a CE. **In a WN.**
`$VO_<VO_name>_SW_DIR`	Base directory of the VO's software or "." for WNs with no shared file system among WNs. **In a WN.**
`$X509_USER_PROXY`	Location of the user proxy certificate (default is /tmp/x509up_u¡uid¿).

Configuration files:

Configuration File	Notes
`$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`	WMS command line tools settings: Retry count, error and output directory, default VO...
`$EDG_WL_LOCATION/etc/<VO>/edg_wl_ui.conf`	VO's specific WMS settings: Resource Broker, Logging and Bookkeeping server and Proxy Server to use.
`$EDG_LOCATION/var/edg-rgma/rgma-defaults`	RGMA default values.
`$EDG_LOCATION/etc/edg_wl.conf`	**In the RB!*** Resource Broker's configuration file. It may be useful to find out which BDII the RB is using.

* This file can not be edited by the user (since it is located in the Resource Broker) but may be retrieved via GridFTP to look at its contains.

APPENDIX C JOB STATUS DEFINITION

As it was already mentioned in Chapter 6, a job can find itself in one of several possible states. Also, only some transitions between states are allowed. These transitions are depicted in Figure 12.

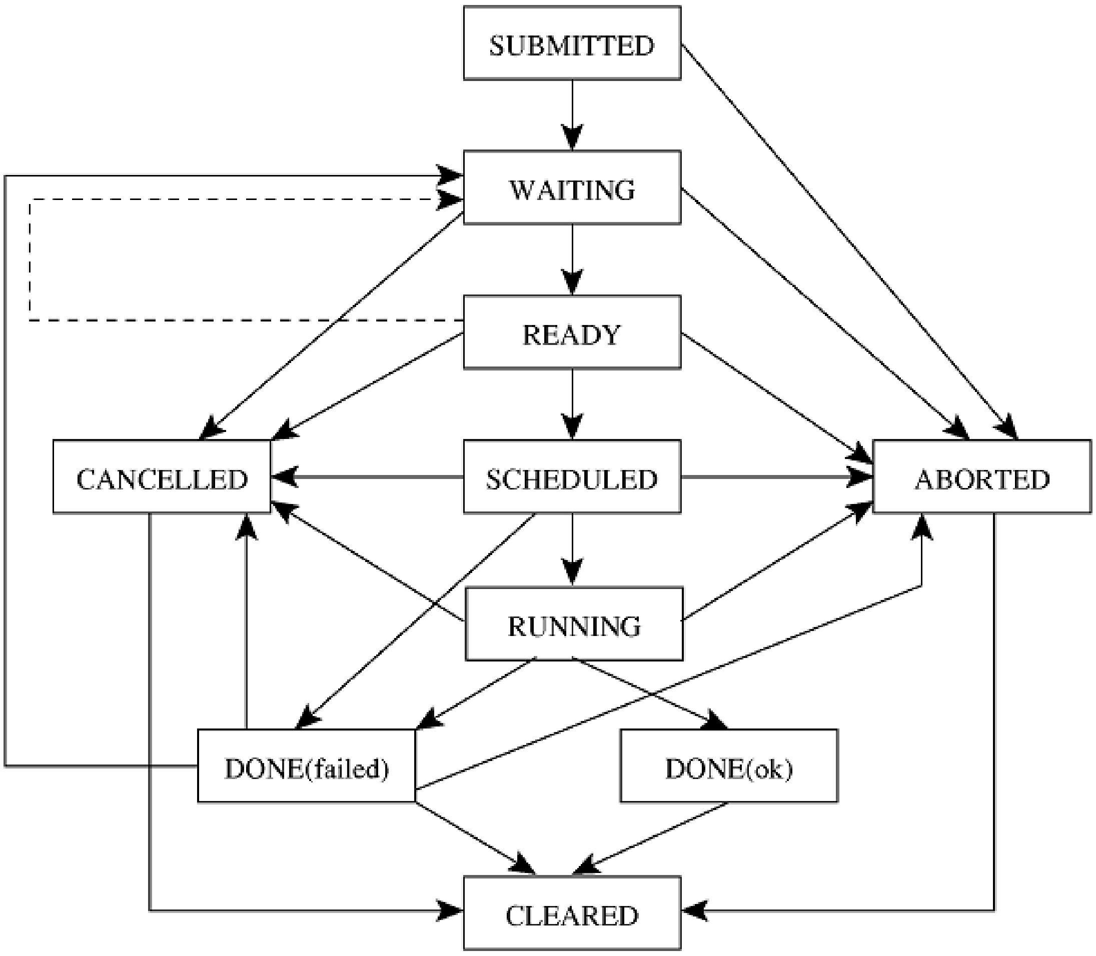


Figure 12: Possible job states in the LCG-2

And the definition of the different states is given in this table.

Status	Definition
SUBMITTED	The job has been submitted by the user but not yet processed by the Network Server
WAITING	The job has been accepted by the Network Server but not yet processed by the Workload Manager
READY	The job has been assigned to a Computing Element but not yet transferred to it
SCHEDULED	The job is waiting in the Computing Element's queue
RUNNING	The job is running
DONE	The job has finished
ABORTED	The job has been aborted by the WMS (e.g. because it was too long, or the proxy certificated expired, etc.)
CANCELED	The job has been canceled by the user
CLEARED	The Output Sandbox has been transferred to the User Interface

APPENDIX D USER TOOLS

D.1. INTRODUCTION

This section introduces some tools that are not really part of the middleware stack of LCG-2, but that can be very useful for user activities on the Grid. Certainly, there are potentially tens of tools and it is impossible to cover them all in this guide. Basically, the purpose of this guide is to introduce the functionality of some of them and to point to other sources of documentation where more detailed information about the tools can be found.

This section will probably evolve to include information of new tools, as they appear or we gain knowledge of them.

Detailed information on the different tools here summarised is provided in Wiki, under the following URL:

http://goc.grid.sinica.edu.tw/gocwiki/User_tools

D.2. JOB MANAGEMENT FRAMEWORK

The submission of large bunches of jobs to LCG resources is a common practice of the VOs during their production phases (software implementation, data production, analysis, etc). The monitoring of these bunches of jobs can become a difficult task without adapted tools that are able to handle the submission and retrieval of the outputs.

Most of the VOs have developed their own tools to perform such job. Here, a framework to automatically submit large bunches of jobs and keep track of their outputs is proposed. Its aim is to assist and guide the users or VOs with the intention of developing their own tools. They could seize parts of this framework to include them in their own applications.

The framework consists mainly of two tools:

- ***submitter_general***: It perform the automatic job submission
- ***get_output***: It retrieves and handles the corresponding outputs

Information on this tool can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/Job_Management_Framework

D.3. JOB MONITORING (LCG-JOB-MONITOR)

The `lcg-job-monitor` command can be used to monitor the process of a job currently running in a WN. The command is intended to be run on a UI. This tool gives information about all statistics for a given jobId, e.g.: memory, virtual memory, real memory, CPU time, DN etc...

The information is retrieved by querying the JobMonitor table (published via R-GMA). The command can return information either for a single job (given the job id), for a user (specifying the DN) or for a whole VO (by name). Standard R-GMA type of queries are supported: `LATEST`, `HISTORY`, `CONTINUOUS`.

Usage is:

```
lcg-job-monitor [-j <jobid>] [-v <VO>] [-u <DN>] [-q <query_type>]
```

Information on this tool can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/Job_Monitoring

D.4. JOB STATUS MONITORING (LCG-JOB-STATUS)

This tool provides information about the status of a running job. It is intended to be run on a UI. The information is retrieved by querying the `JobStatusRaw` table (published via R-GMA). The command returns information for a specified job (given the job id).

Usage:

```
lcg-job-status.py [-j <jobid>] [-q <type>]
```

where the query type can be either `LATEST`, `CONTINUOUS` or `HISTORY` (these are standard R-GMA query types). For `LATEST` and `HISTORY` queries, some seconds are waited after the query is made. After that, the returned results, or a ”no events” message, if none is got, are printed. In the case of `CONTINUOS` queries, the status is checked every 5 seconds until the program is exited via `Ctrl-C` or the `Done` status is reached.

Information on this tool can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/Job_Status_Monitoring

D.5. TIME LEFT UTILITY (LCG-GETJOBSTATS, LCG_JOBSTATS.PY)

The tools described in this section can be invoked by a running job to determine some statistics about itself. Right now the parameters that can be retrieved are:

- How much CPU or wall clock time it has consumed
- How long it can still run (before reaching the CPU or wall clock time limits).

There are scripts (CLI) and python modules (API) that can be used for these purposes. They are described in detail later. They work by querying the CE's batch system (using different ways depending on the batch system that the given CE uses).

The results returned by the tools are not always trustable, since some sites do not set time limits and some LRMSs cannot provide resource usage information to the tool. The underlying idea of these CLI and API is that the experiment decides how to use them and when to trust their results. Please read the documentation regarding the different batch systems supported.

ATTENTION!!: This command executes heavy processes on the Computing Elements, please do not use it too often (not every minutes or event processed!). Rather, limit the usage to something like once every hour, or when a sensitive percentage of your job is accomplished.

The following files should be present in the WN (either already in the release or shipped with the job in a tarball):

- `lcg-getJobStats` (small wrapper bash script around the corresponding python script)
- `lcg-getJobTimes` (small wrapper bash script around the corresponding python script)
- `lcg-getJobStats.py` (python script: executable)
- `lcg-getJobTimes.py` (python script: executable)
- `lcg_jobConsumedTimes.py` (python module)
- `lcg_jobStats.py` (python module)

In principle, one should only deal with `lcg-getJobStats` or with `lcg_jobStats.py` (for python API). `lcg-getJobTimes` (and `lcg_jobConsumedTimes.py`) provide a way to estimate the used CPU and wall clock time without querying the batch system, but by parsing the proc filesystem. It is internally called by `lcg-getJobStats` in the cases where it cannot get the information from the batch system (like when Condor is used).

Information on this tool can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/Time_Left_Utility

D.6. Information System Reader (lcg-info)

This command was already discussed in Section 5.1.2.

Nevertheless, the more up-to-date information can be always got from:

http://goc.grid.sinica.edu.tw/gocwiki/Information_System_Reader

APPENDIX E VO-WIDE UTILITIES

E.1. INTRODUCTION

This section introduces some utilities that are only relevant to certain people in a VO (VO managers, experiment software managers...). They are basically administrative tools. The purpose of this section is to introduce the functionality of some of them and to point to other sources of documentation where more detailed information about the utilities can be found.

Detailed information on the different tools here summarised is provided in Wiki, under the following URL:

http://goc.grid.sinica.edu.tw/gocwiki/VoDocs

E.2. FREEDOM OF CHOICE FOR RESOURCES

The ***Freedom of Choice for Resources (FCR)*** Pages is a web interface for VO Software Managers. The tool gives a possibility to set up selection rules for Computing and Storage Elements, which will affect the information published by top level BDIIs configured accordingly.

Resources can be permanently in- or excluded or be used in case if the execution of the ***Sites Functional Tests (SFT)*** was successful. There is also a possibility to use VO-specific test results to be taken in account on the top of the SFT results.

Certificate-based authentication method protects the pages. Access has to be requested.

Information on this can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/FCR_Pages

E.3. THE VO BOX

The VO-BOX is a type of node, which will be deployed in all sites, where the experiment can run specific agents and services. The acess to the VO-BOX is restricted to the SOFTWARE GROUP MANAGER of the VO. If you are not your VO SGM, you will not be interested in the VO-BOX.

A description of the VO-BOX functionalities and usage is described in WIKI indicated below.

Information on this can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/VOBOX_HowTo

E.4. EXPERIMENTS SOFTWARE INSTALLATION

Authorized users can install software in the computing resources of LCG-2. The installed software, which we will call Experiment Sofware, is also published in the Information Service, so that user jobs can run on nodes where the software they need is installed.

The Experiment Software Manager (ESM) is the member of the experiment VO entitled to install Application Software in the different sites. The ESM can manage (install, validate, remove...) Experiment Software on a site at any time through a normal Grid job, without previous communication to the site administrators. Such job has in general no scheduling priorities and will be treated as any other job of the same VO in the same queue. There would be therefore a delay in the operation if the queue is busy.

The site provides a dedicated space where each supported VO can install or remove software. The amount of available space must be negotiated between the VO and the site.

Information on this can be found under:

http://goc.grid.sinica.edu.tw/gocwiki/Experiments_Software_Installation

APPENDIX F DATA MANAGEMENT AND FILE ACCESS THROUGH AN APPLICATION PROGRAMMING INTERFACE

The development of code for jobs submitted to LCG-2 is out of the scope of this guide, and therefore the different APIs for Data Management and Grid File Access will not be covered in full detail. This section just summarizes what APIs exist and gives some examples of lcg_util and GFAL use.

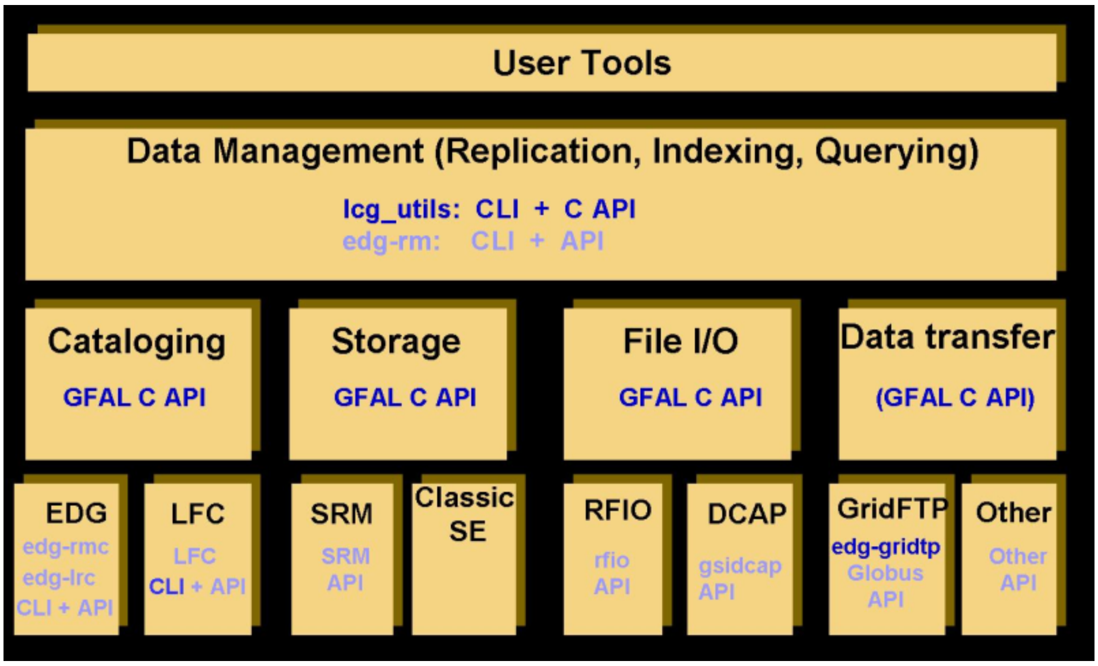


Figure 13: Layered view of the Data Management APIs and CLIs

Figure 13 shows a layered view of the different APIs that are available for Data Management operations in LCG-2. In the figure the CLIs and APIs whose use is discouraged are shadowed. It also includes the already described CLIs, which can be usually related to one of the APIs (as *being in the same layer*).

On the top, just below the tools developed by the users, we find the ***lcg_util*** API. This is a C API that provides the same functionality as the `lcg-*` commands (lcg-utils). In fact, the commands are just a wrapper around the C calls. This layer should cover most basic needs of user applications. It is abstract in the sense that it is independent from the underlying technology, since it will transparently interact with either the RLS or the LFC catalog and will use the correct protocol (usually GSIFTP) for file transfer.

Apart from the basic calls `lcg_cp`, `lcg_cr`, etc., there are other calls that enhance this with a buffer for complete error messages (`lcg_cpx`, `lcg_crx`, ...), that include timeouts (`lcg_cpt`, `lcg_crt`, ...), and both (`lcg_cpxt`, `lcg_crxt`). Actually, all calls use the most complete version (i.e. `lcg_cpxt`...) with default values for the arguments that were not provided.

Below the lcg_util API, we find ***the Grid File Access Library (GFAL)***. GFAL provides calls for catalog interaction, storage management and file access and can be very handy when an application requires access to some part of a big Grid file but does not want to copy the whole file locally. The library hides the interactions with the LCG-2 catalogs and the SEs and SRMs and presents a POSIX interface for the I/O operations on the files. The function names are obtained by prepending `gfal_` to the POSIX names, for example `gfal_open`, `gfal_read`, `gfal_close`...

GFAL accepts GUIDs, LFNs, SURLs and TURLs as file names, and, in the first two cases, it tries to find the closest replica of the file. Depending on the type of storage where the file's replica resides in, GFAL will use one protocol or another to access it. GFAL can deal with GSIFTP, secure and insecure RFIO, or gsidcap in a transparent way for the user (unless a TURL is used, in which case the protocol is explicitly indicated).

NOTE: In the case where LFNs or GUIDs are used, the library needs to contact the file catalogs to get the corresponding TURL. Since the catalogs are VO dependant, and since the calls do not include any argument to specify the VO, GFAL requires the `LCG_GFAL_VO` environment variable to be set; along with the pointer for the Information Service: `LCG_GFAL_INFOSYS`. Alternatively, the endpoints of the catalogs may be specified directly, by setting: `LFC_HOST` (LFC) or `RMC_ENDPOINT` and `LRC_ENDPOINT` (RLS).

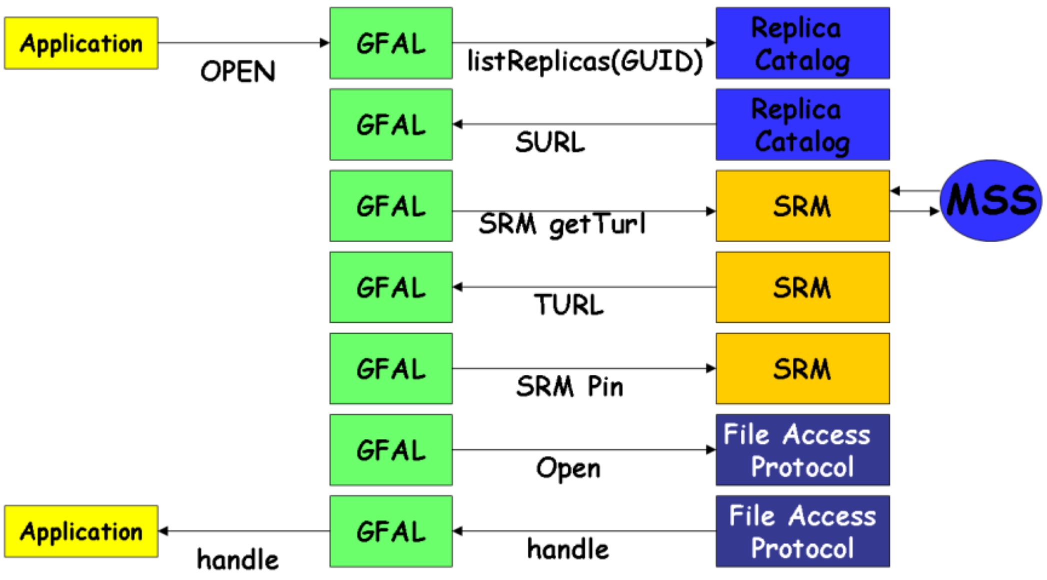


Figure 14: Flow diagram of a GFAL call

This behaviour is illustrated in Figure 14, which shows the flow diagram of a `gfal_open` call. This call will locate a Grid file and return a remote file descriptor so that the caller can read or write file remotely, as it would do for a local file. As shown in the figure, first, if a GUID is provided, GFAL will contact a file catalog to retrieve the corresponding SURL. Then, it will access the SRM interface of the SE that the SURL indicates, it will get a valid TURL and also pin the file so that it is there for the subsequent access. Finally, with the TURL and using the appropriate protocol, GFAL will open the file and return a file handle to the caller.

Nevertheless, GFAL sometimes exposes functionality applicable only to a concrete underlying technology (or protocol) if this is considered useful. A good example of this is the exposed SRM interface that GFAL provides. Some code exploiting this functionality is shown later.

Finally, below GFAL, we find some other CLIs and APIs which are technology dependent. Their direct use is in general discouraged (except for the mentioned cases of the LFC client tools and the `edg-gridftp-*` commands). Nonetheless, some notes on the RFIO API are given later on.

Example F.0.1 (Using lcg_util API to transfer a file)

The following example copies a file from a SE to the WN where our job is running, using the call with timeout. The file can be then accessed locally with normal file I/O calls.

The source code follows:

```
#include <iostream>
#include <unistd.h> // For the unlink function
#include <fstream>  // For file access inside "doSomethingWithFile" (ifstream)

extern "C"{
   #include "lcg_util.h"
   }

   using namespace std;

/* A function to further process the copied file... */
bool doSomethingWithFile(const char * pFile2){
   //....
}

int main(){
/*
 * Parameters of the lcg_cp call
 * int  lcg_cpt (char  *src_file,  char  *dest_file,  char  *vo,  int nbstreams,
 *               char *conf_file, int insecure, int verbose, int timeout);
 */
   char * src_file="lfn:my_lcg_cr_example";
   char * dest_file=new char[200];
   char * vo="dteam";
   int nbstreams=1;
   // conf_file=0  (currently ignored)
   // insecure=0   (currently ignored)
   int verbose=1;
   int timeout=180;

/* Form the name of the destination file */
   char * my_file="my_retrieved_file";
   char * pwd=getenv("PWD");
   strcpy(dest_file,"file:");
   strcat(dest_file,pwd);
   strcat(dest_file,"/");
   strcat(dest_file,my_file);
/* The lcg_cp call itself */
   if(lcg_cp(src_file, dest_file, vo, nbstreams, 0, 0, verbose, timeout)==0){
      cout << "File correctly copied to local filesystem " << endl;
   }
   else{
      perror("Error with lcg_cp!");
   }

/* Further processing */
   doSomethingWithFile(my_file);

/* Cleaning... */

  // Delete the temporary file
  unlink(my_file);


/* That was it */
   cout << endl;
   return 0;

}//end of main
```

The program should be compilable with the following line:

```
$ /opt/gcc-3.2.2/bin/c++-3.2.2 -I$LCG_LOCATION/include \
   -L$LCG_LOCATION/lib -L$GLOBUS_LOCATION/lib  \
   -llcg_util -lgfal -lglobus_gass_copy_gcc32 -o lcg_cp_example lcg_cp_example.cpp
```

Note: The linkage with `libglobus_gass_copy_gcc32.so` should not be necessary, and also the one with `libgfal.so` should done transparently when linking `liblcg_util.so`. Nevertheless, their explicit linkage as shown in the example was necessary for the program to compile in the moment that this guide was written. At time of reading, it may not be necessary anymore.

Example F.0.2 (Using GFAL to access a file)

The following C++ code uses the `libgfal` library to access a Grid file, whose name is specified as a command line argument. The program opens the files, writes a set of numbers into it, and closes it. Afterwards, the files is opened again, and the previously written numbers are read and shown to the user. The source code (`gfal_example.cpp`) follows:

```
#include<iostream>
#include <fcntl.h>
```

```
#include <stdio.h>
extern "C" {
#include "/opt/lcg/include/gfal_api.h"
}

using namespace std;

/* Include the gfal functions (are C and not C++, therefore are 'extern') */
extern "C" {
 int gfal_open(const char*, int, mode_t);
 int gfal_write(int, const void*, size_t);
 int gfal_close(int);
 int gfal_read(int, void*, size_t);
}

/***************  MAIN  *************/
main(int argc, char **argv)
{
 int fd;  // file descriptor
 int rc;  // error codes
 size_t INTBLOCK=40; // how many bytes we will write each time (40 = 10 int a time)

 /* Check syntax (there must be 2 arguments) */
 if (argc != 2) {
   cerr << "Usage: " << argv[0]<< "filename\n";
   exit (1);
 }

 /* Declare and initialize the array of input values (to be written in the file) */
 int* original = new int[10];
 for (int i=0; i<10; i++) original[i]=i*10; // just: 0, 10, 20, 30...

 /* Declare and give size for the array that will store the values read from the file */
 int* readValues = new int[10];

 /* Create the file for writing with the given name */
 cout << "\nCreating file " << argv[1] << endl;
 if ((fd = gfal_open (argv[1], O_WRONLY | O_CREAT, 0644)) < 0) {
   perror ("gfal_open");
   exit (1);
 }
 cout << " ... Open successful ... " ;

 /* Write into the file (reading the 10 integers at once from the int array) */
   if ((rc = gfal_write (fd, original, INTBLOCK )) != INTBLOCK) {
   if (rc < 0)   perror ("gfal_write");
   else  cerr << "gfal_write returns " << rc << endl;
   (void) gfal_close (fd);
   exit (1);
```

```
  }
  cout << "Write successful ... ";

  /* Close the file */
  if ((rc = gfal_close (fd)) < 0) {
    perror ("gfal_close");
    exit (1);
  }
  cout << "Close successful" << endl;

  /* Reopen the file for reading */
  cout << "\nReading back " << argv[1] << endl;
  if ((fd = gfal_open (argv[1], O_RDONLY, 0)) < 0) {
    perror ("gfal_open");
    exit (1);
  }
  cout << " ... Open successful ... ";

  /* Read the file (40 bytes directly into the readValues array) */
  if ((rc = gfal_read (fd, readValues, INTBLOCK )) != INTBLOCK) {
    if (rc < 0)  perror ("gfal_read");
    else  cerr << "gfal_read returns " << rc << endl;
    (void) gfal_close (fd);
    exit (1);
  }
  cout << "Read successful ...";

  /* Show what has been read */
  for(int i=0; i<10; i++)
    cout << "\n\tValue of readValues[" << i << "] = " << readValues[i];

  /* Close the file */
  if ((rc = gfal_close (fd)) < 0) {
    perror ("gfal_close");
    exit (1);
  }
  cout << "\n ... Close successful";
  cout << "\n\nDone" << endl;

}//end of main
```

The command used to compile and link the previous code (it may be different in your machine) is:

```
$ /opt/gcc-3.2.2/bin/c++-3.2.2 -L /opt/lcg/lib/ -l gfal -o gfal_example gfal_example.cpp
```

As temporary file, we may specify one in our local filesystem, by using the `file://` prefix. In that case we get the following output:

```
$ ./gfal_example file://`pwd`/test.txt

Creating file file:///afs/cern.ch/user/d/delgadop/gfal/test.txt
 ... Open successful ... Write successful ... Close successful

Reading back file:///afs/cern.ch/user/d/delgadop/gfal/test.txt
 ... Open successful ... Read successful ...
        Value of readValues[0] = 0
        Value of readValues[1] = 10
        Value of readValues[2] = 20
        Value of readValues[3] = 30
        Value of readValues[4] = 40
        Value of readValues[5] = 50
        Value of readValues[6] = 60
        Value of readValues[7] = 70
        Value of readValues[8] = 80
        Value of readValues[9] = 90
 ... Close successful

Done
```

We may define a JDL file and access a Grid file from a job. Indeed, a Grid file cannot be accessed directly from the UI if insecure RFIO is the protocol used for that access. However, the access from a job running in the same site where the file is stored is allowed. The reason for this is that insecure RFIO does not handle Grid certificates (secure RFIO does), and while the UID a user's job is mapped to will be allowed to access a file in a SE, the user's UID in the UI is different, and will not be allowed to perform that access.

In opposition to the insecure RFIO, the secure version, also called ***gsirfio***, includes all the usual GSI security, and so it can deal with certificates rather than with users' UIDs. For this reason, it can be used with no problem to access files from UIs or in remote SEs. Just as gsidcap can.

Attention: Some SEs support only insecure RFIO (classic SEs and CASTOR), while others support only secure RFIO (dpm), but they all publish `rfio` as the supported protocol in the IS. The result is that currently GFAL has to figure out which one of the two RFIO versions it uses basing on an environmental variable. This variable is called `LCG_RFIO_TYPE`. If its value is `dpm`, the secure version of RFIO will be used, if its value is `castor` or the variable it is undefined, then insecure RFIO will be the one chosen.

Unfortunately, an insecure RFIO client cannot talk to a secure server and viceversa. Therefore, the user must correctly define the indicated variable depending on the SE he wants to talk to before using GFAL calls. Otherwise, the calls will not work.

Another important issue is that of the names used to access files.

For classic SEs, both the SURL and TURL names of the files must include a double slash between the hostname of the SE and the path of the file. This is needed by GFAL for RFIO. An example of correct SURL and TURL is:

```
sfn://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
rfio://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
```

These name requirements are imposed by the use of RFIO as access protocol. As seen in previous examples, the `lcg-*` commands will work with SURLs and TURLs registered in the catalogs, even if they do not follow this rules. This does not happen with RFIO. Therefore, it is always better to use LFNs or GUIDs when dealing with files, not to have to deal with SURL and TURL naming details.

IMPORTANT!: Nevertheless, the entries in the RMC and LRC may contain SURLs which do not comply with the described rules. As a result, when GFAL uses the GUID or LFN to retrieve the SURL of the file, it will get an incorrect one, and the call will fail. In those cases, using the correct SURL (which usually means doubling the slash after the hostname), instead of the GUID or LFN, is the only way to access the file.

Having this all in mind, let us build a JDL file to create and read a Grid file with our C++ program:

```
Executable="gfal_example";
StdOutput="std.out";
StdError="std.err";
Arguments="sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file";
InputSandbox={"gfal_example"};
OutputSandbox={"std.out","std.err"};
```

After submitting the job, the output retrieved in `std.out` is as follows:

```
Creating file sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file
 ... Open successful ... Write successful ... Close successful

Reading back sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file
 ... Open successful ... Read successful ...
        Value of readValues[0] = 0
        Value of readValues[1] = 10
        Value of readValues[2] = 20
        Value of readValues[3] = 30
        Value of readValues[4] = 40
        Value of readValues[5] = 50
        Value of readValues[6] = 60
        Value of readValues[7] = 70
        Value of readValues[8] = 80
        Value of readValues[9] = 90
 ... Close successful

Done
```

It is important to notice that the creation of a new file using GFAL does not imply the registration of that file. This means that if the created file has to be used as a Grid file, then it should be manually registered using `lcg-rf`. Otherwise, the file should be deleted using `edg-gridftp-rm`.

As seen, by using GFAL, an application can access a file remotely almost as if it was local (substituting POSIX calls by those of GFAL). For more information on GFAL, refer to the manpages of the library (`gfal`) and of the different calls (`gfal_open`, `gfal_write`...).

In addition to GFAL, there is also the possibility to use the RFIO's C and C++ APIs, which also give applications the possibility to open and read a file remotely.

Nevertheless, RFIO presents several limitations in comparison to GFAL. First of all, it can only be used to access those SEs or SRMs that support the RFIO protocol, while GFAL will deal with any of the other supported protocols also. Secondly, RFIO does not understand GUIDs, LFNs or SURLs, and it can only operate with RFIO's TURLs.

Finally, as was explained previously, insecure RFIO can only be used in order to access files that are located in the same local area network where the CE holding the job is located. In order to access or move files between different sites, the user should use a different method. Of course, if the only way to remotely access a file is insecure RFIO (as is the case for classic SEs or CASTOR), then GFAL calls will also use insecure RFIO as the protocol for the interaction and therefore this last limitation will also apply.

Although direct use of RFIO's APIs is discouraged, information on it and its APIs can be found in [39].

Example F.0.3 (Explicit interaction with the SRM using GFAL)

The following example program can be useful for copying a file that is stored in a MSS. It asks for the file to be staged from tape to disk first, and only tries to copy it when the file has been migrated.

The program uses both the lcg_util and the GFAL APIs. From lcg_util, just the `lcg_cp` call is used. From GFAL, `srm_get`, which requests a file to be staged from tape to disk, and `srm_get_status`, which checks the status of the previous request, are used.

The source code follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <iostream>
#include <sstream>   // for the integer to string conversion
#include <unistd.h>  // for the sleep function
#include <fstream>   // for the local file access
extern "C"{
   #include "gfal_api.h"
   #include "lcg_util.h"
}

using namespace std;

main(int argc, char ** argv){
/* Check arguments */
 if ((argc < 2) || (argc > 2)) {
   cerr << "Usage: " << argv[0] << " SURL\n";
        exit (1);
 }
```

```
/*
 * Try to get the file (stage in)
 * int srm_get (int nbfiles, char **surls, int nbprotocols, char **protocols, int *reqid,
 *              char **token, struct srm_filestatus **filestatuses, int timeout);
 *
 * struct srm_filestatus{
 *   char  *surl;
 *   char  *turl;
 *   int   fileid;
 *   int   status;};
 */
 int nbreplies;     //number of replies returned
 int nbfiles=1;       // number of files
 char **surls;      // array of SURLs
 int nbprotocols;   // number of bytes of the protocol array
 char * protocols[] = {"rfio"};     // protocols
 int reqid;         // request ID
 //char **token=0;  // unused
 struct srm_filestatus *filestatuses;   // status of the files
 int timeout=100;

/* Set the SURL and the nbprotocols */
 surls = &argv[1];
 nbprotocols = sizeof(protocols) / sizeof(char *);

/* Make the call */
 if ((nbreplies = srm_get (nbfiles, surls, nbprotocols, protocols,
          &reqid, 0, &filestatuses, timeout)) < 0) {
    perror ("Error in srm_get");
         exit (-1);
 }

/* Show the retrieved information */
 cout << "\nThe status of the file is: " << endl;
 cout << endl << filestatuses[0].status << "  --  " << filestatuses[0].surl;
 free(filestatuses[0].surl);
 if(filestatuses[0].status == 1){
    cout << "  (" << filestatuses[0].turl << ")" << endl;
    free(filestatuses[0].turl);
 }
 else {cout << endl;}
 free(filestatuses);

 if(filestatuses[0].status == -1){
    cout << endl << "Error when trying to stage the file. Not waiting..." << endl;
    exit(-1);
 }
```

```
/*
 * Now watch the status until it gets to STAGED (1)
 * int srm_getstatus (int nbfiles, char **surls, int reqid, char **token,
 *                     struct srm_filestatus **filestatuses, int timeout);
 */
 cout << "\nWaiting for the file to be staged in..." << endl;
 int numiter=1;
 int filesleft=1;

 char * destfile = new char[200];
 while((numiter<50) && (filesleft>0)){
    //sleep longer each iteration
    sleep(numiter++);
    cout << "#";  // just to show we are waiting and not dead
    cout.flush();

    if ((nbreplies = srm_getstatus (nbfiles, surls, reqid, NULL, &filestatuses, timeout))
         < 0) {
       perror ("srm_getstatus");
       exit (-1);
    }

    if (filestatuses[0].status == 1){
       cout << "\nREADY  --  " << filestatuses[0].surl << endl;
       filesleft--;
       // Create a name for the file to be saved
       strcpy(destfile, "file:/tmp/srm_gfal_retrieved");
       cout << "\nCopying " << filestatuses[0].surl << " to " << destfile << "...\n";
       // Copy the file to the local filesystem
       if(lcg_cp(filestatuses[0].surl, destfile, "dteam", 1, 0, 0 , 1)!=0){
           perror("Error in lcg_cp");
       }
    }
    free(filestatuses[0].surl);
    if(filestatuses[0].status == 1)   free(filestatuses[0].turl);
    free(filestatuses);
 }

 if(numiter>49){
    cout << "\nThe file did not reach the READY status. It could not be copied." << endl;
 }


/* Cleaning */
 delete [] destfile;

/* That was all */
 cout << endl;
 return reqid;  // return the reqid, so that it can be used by the caller

}//end of main
```

The `srm_get` function is called once to request the staging of the file. In this call, we retrieve the corresponding TURL and some numbers identifying the request. If a LFN was provided, several TURLs (from several replicas) could be retrieved. In this case, only one TURL will be returned (stored in the first position of the `filestatuses` array).

The second part of the program is a loop that will repeatedly call `srm_getstatus` in order to get the current status of the previous request, until the status is equal to 1 (ready). There is a `sleep` call to let the program wait some time (time increasing with each iteration) for the file staging. Also a maximum number of iterations is set (50), so that the program does not wait for ever, but rather ends finally with an aborting message.

When the file is ready, it is copied using `lcg_cp` in the same way as seen in a previous example. This or other application should then perform some operation on the file (this is not shown here).

A possible output of this program is the following:

```
The status of the file is:

0  --  srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Waiting for the file to be staged in...
##################

READY  --  srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Copying srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1 to
file:/tmp/srm_gfal_retrieved...
Source URL: srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
File size: 2331
Source URL for copy:
gsiftp://castorgrid.cern.ch:2811//shift/lxfs5614/data03/cg/stage/test_1.172962
Destination URL: file:/tmp/srm_gfal_retrieved
# streams: 1
Transfer took 590 ms
```

Where the `0` file status means that the file exists but it lays on the tape (not staged yet), the hash marks show the iterations in the looping and finally the `READY` indicates that the file has been staged in and it can be copied (what it is done afterwards as shown by the normal verbose output).

If the same program was run again, passing the same SURL as argument, it would return almost immediately, since the file has been already staged. This is shown in the following output:

```
The status of the file is:
```

```
1  --  srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
(rfio://lxfs5614//shift/lxfs5614/data03/cg/stage/test_1.172962)

Waiting for the file to be staged in...
#
READY  --  srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Copying srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1 to
file:/tmp/srm_gfal_retrieved...
Source URL: srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
File size: 2331
Source URL for copy:
gsiftp://castorgrid.cern.ch:2811//shift/lxfs5614/data03/cg/stage/test_1.172962
Destination URL: file:/tmp/srm_gfal_retrieved
# streams: 1
Transfer took 550 ms
```

Where the `1` file status means that the file is already in disk.

APPENDIX G   THE GLUE SCHEMA

As explained earlier, the GLUE Schema describes the Grid resources information that is stored by the Information System. This section gives information about the MDS, namely the LDAP implementation of the GLUE Schema, which is currently used in the LCG-2 IS. For information on the abstract GLUE Schema definition, please refer to [27].

First of all, the tree of object classes definitions is shown. Then, the attributes for each one of the objectclasses (where the data are actually stored) are presented. Some of the attributes may be currently empty, even if they are defined in the schema. Furthermore, some new attributes may be published, although not yet collected in the schema. Finally, the DITs currently used in the IS for the publishing of these attributes are shown.

G.1. THE GLUE SCHEMA LDAP OBJECT CLASSES TREE

```
Top
 |
 ---- GlueTop 1.3.6.1.4.1.8005.100
             |
             ---- .1. GlueGeneralTop
             |      |
             |      ---- .1. ObjectClass
             |      |      |
             |      |      ---- .1 GlueSchemaVersion
             |      |      |
             |      |      ---- .2 GlueCESEBindGroup
             |      |      |
             |      |      ---- .3 GlueCESEBind
             |      |      |
             |      |      ---- .4 GlueKey
             |      |      |
             |      |      ---- .5 GlueInformationService
             |      |      |
             |      |      ---- .6 GlueService
             |      |      |
             |      |      ---- .7 GlueServiceData
             |      |      |
             |      |      ---- .8 GlueSite
             |      |
             |      ---- .2. Attributes
             |             |
             |             ---- .1. Attributes for GlueSchemaVersion
             |             |              . . .
             |             |
             |             ---- .8. Attributes for GlueSiteTop
             |
             ---- .2. GlueCETop
```

```
|       |
|       ---- .1. ObjectClass
|       |       |
|       |       ---- .1  GlueCE
|       |       |
|       |       ---- .2  GlueCEInfo
|       |       |
|       |       ---- .3  GlueCEState
|       |       |
|       |       ---- .4  GlueCEPolicy
|       |       |
|       |       ---- .5  GlueCEAccessControlBase
|       |       |
|       |       ---- .6  GlueCEJob
|       |       |
|       |       ---- .7  GlueVOView
|       |
|       ---- .2. Attributes
|       |       |
|       |       ---- .1.  Attributes for GlueCE
|       |       |              . . .
|       |       |
|       |       ---- .7.  Attributes for GlueVOView
|       |
|       ---- .3. MyObjectClass
|       |
|       ---- .4. MyAttributes
|
---- .3. GlueClusterTop
|       |
|       ---- .1. ObjectClass
|       |       |
|       |       ---- .1  GlueCluster
|       |       |
|       |       ---- .2  GlueSubCluster
|       |       |
|       |       ---- .3  GlueHost
|       |       |
|       |       ---- .4  GlueHostArchitecture
|       |       |
|       |       ---- .5  GlueHostProcessor
|       |       |
|       |       ---- .6  GlueHostApplicationSoftware
|       |       |
|       |       ---- .7  GlueHostMainMemory
|       |       |
|       |       ---- .8  GlueHostBenchmark
|       |       |
|       |       ---- .9  GlueHostNetworkAdapter
|     |     |
|     |     ---- .10 GlueHostProcessorLoad
|     |     |
|     |     ---- .11 GlueHostSMPLoad
|     |     |
|     |     ---- .12 GlueHostOperatingSystem
|     |     |
|     |     ---- .13 GlueHostLocalFileSystem
|     |     |
|     |     ---- .14 GlueHostRemoteFileSystem
|     |     |
|     |     ---- .15 GlueHostStorageDevice
|     |     |
|     |     ---- .16 GlueHostFile
|     |     |
|     |     ---- .17 GlueLocation
|     |
|     ---- .2. Attributes
|     |     |
|     |     ---- .1. Attributes for GlueCluster
|     |     |            . . .
|     |     |
|     |     ---- .17  Attributes for GlueLocation
|     |
|     ---- .3. MyObjectClass
|     |
|     ---- .4. MyAttributes
|
---- .4. GlueSETop
|     |
|     ---- .1. ObjectClass
|     |     |
|     |     ---- .1  GlueSE
|     |     |
|     |     ---- .2  GlueSEState
|     |     |
|     |     ---- .3  GlueSEAccessProtocol
|     |     |
|     |     ---- .4  GlueSEControlProtocol
|     |
|     ---- .2. Attributes
|     |     |
|     |     ---- .1.  Attributes for GlueSE
|     |     |             . . .
|     |     |
|     |     ---- .4.  Attributes for GlueSEControlProtocol
|     |
|     ---- .3. MyObjectClass
|     |
|      ---- .4. MyAttributes
|
---- .5. GlueSLTop
|      |
|      ---- .1. ObjectClass
|      |      |
|      |      ---- .1  GlueSL
|      |      |
|      |      ---- .2  GlueSLLocalFileSystem
|      |      |
|      |      ---- .3  GlueSLRemoteFileSystem
|      |      |
|      |      ---- .4  GlueSLFile
|      |      |
|      |      ---- .5  GlueSLDirectory
|      |      |
|      |      ---- .6  GlueSLArchitecture
|      |      |
|      |      ---- .7  GlueSLPerformance
|      |
|      ---- .2. Attributes
|      |      |
|      |      ---- .1. Attributes for GlueSL
|      |      |             . . .
|      |      |
|      |      ---- .7  Attributes for GlueSLPerformance
|      |
|      ---- .3. MyObjectClass
|      |
|      ---- .4. MyAttributes
|
---- .6. GlueSATop
       |
       ---- .1. ObjectClass
       |      |
       |      ---- .1  GlueSA
       |      |
       |      ---- .2  GlueSAPolicy
       |      |
       |      ---- .3  GlueSAState
       |      |
       |      ---- .4  GlueSAAccessControlBase
       |
       ---- .2. Attributes
       |      |
       |      ---- .1. Attributes for GlueSA
       |      |             . . .
       |      |
       |      ---- .4  Attributes for GlueSAAccessControlBase
```

```
|
---- .3. MyObjectClass
|
---- .4. MyAttributes
```

G.2. General Attributes

This group includes some base (top) object classes, which have no attributes, and, thus, no actual resource data, and some other that include general attributes that are defined in entries of both CEs and SEs. These are the version of the schema that is used, the URL of the IS server and finally the `GlueKey`, which is used to relate different entries of the tree and in this way overcome OpenLDAP limitations in query flexibility.

- **Base class (objectclass `GlueTop`)**
  - No attributes
- **Base class for general object classes, attributes, matching rules, etc. (objectclass `GlueGeneralTop`)**
  - No attributes
- **Schema Version Number (objectclass `GlueSchemaVersion`)**
  - `GlueSchemaVersionMajor`: Major Schema version number
  - `GlueSchemaVersionMinor`: Minor Schema version number
- **Internal attributes to express object associations (objectclass `GlueKey`)**
  - `GlueChunkKey`: Relative DN (AttributeType=AttributeValue) to reference a related entry in the same branch than this DN
  - `GlueForeignKey`: Relative DN (AttributeType=AttributeValue) to reference a related entry in a different branch
- **Information for the Information Service (objectclass `GlueInformationService`)**
  - `GlueInformationServiceURL`: The Information Service URL publishing the related information
- **Service entity (objectclass `GlueService`)**
  - `GlueServiceUniqueID`: unique identifier of the service
  - `GlueServiceName`: human-readable name of the service
  - `GlueServiceType`: type of service
  - `GlueServiceVersion`: version of the service: major.minor.patch
  - `GlueServiceEndpoint`: network endpoint for the service
  - `GlueServiceStatus`: status of the service (OK, Warning, Critical, Unknown, Other)
  - `GlueServiceStatusInfo`: textual explanation of the status
  - `GlueServiceWSDL`: URI of the service WSDL
  - `GlueServiceSemantics`: URL of a detailed description of the service

- GlueServiceStartTime: time of last service start
- GlueServiceOwner: owner of the service (e.g. the VO)

The attributes GlueServicePrimaryOwnerName, GlueServicePrimaryOwnerContact, GlueServiceHostingOrganization, GlueServiceMajorVersion, GlueServiceMinorVersion, GlueServicePatchVersion, GlueServiceAccessControlRule and GlueServiceInformationServiceURL are deprecated from version 1.2 of the GLUE schema.

G.3. Attributes for the Computing Element

These are attributes that give information about a CE and its composing WNs. In the GLUE Schema, they are defined in the UML diagram for the Computing Element.

- **Base class for the CE information (objectclass `GlueCETop`)**
  - No attributes
- **Base class for the cluster information (objectclass `GlueClusterTop`)**
  - No attributes
- **Computing Element (objectclass `GlueCE`)**
  - GlueCEUniqueID: unique identifier for the CE
  - GlueCEName: human-readable name of the service
- **General Info for the queue associated to the CE (objectclass `GlueCEInfo`)**
  - GlueCEInfoLRMSType: name of the local batch system
  - GlueCEInfoLRMSVersion: version of the local batch system
  - GlueCEInfoGRAMVersion: version of GRAM
  - GlueCEInfoHostName: fully qualified name of the host where the gatekeeper runs
  - GlueCEInfoGateKeeperPort: port number for the gatekeeper
  - GlueCEInfoTotalCPUs: number of CPUs in the cluster associated to the CE
  - GlueCEInfoContactString: contact string for the service
  - GlueCEInfoJobManager: job manager used by the gatekeeper
  - GlueCEInfoApplicationDir: path of the directory for application installation
  - GlueCEInfoDataDir: path a shared the directory for application data
  - GlueCEInfoDefaultSE: unique identifier of the default SE
- **CE State (objectclass `GlueCEState`)**
  - GlueCEStateStatus: queue status: queueing (jobs are accepted but not run), production (jobs are accepted and run), closed (jobs are neither accepted nor run), draining (jobs are not accepted but those in the queue are run)

- GlueCEStateTotalJobs: total number of jobs (running + waiting)
- GlueCEStateRunningJobs: number of running jobs
- GlueCEStateWaitingJobs: number of jobs not running
- GlueCEStateWorstResponseTime: worst possible time between the submission of a job and the start of its execution, in seconds
- GlueCEStateEstimatedResponseTime: estimated time between the submission of a job and the start of its execution, in seconds
- GlueCEStateFreeCPUs: number of CPUs available to the scheduler
- GlueCEStateFreeJobSlots: number of jobs that could start, given the current number of jobs submitted

- **CE Policy (objectclass `GlueCEPolicy`)**
  - GlueCEPolicyMaxWallClockTime: maximum wall clock time available to jobs submitted to the CE, in minutes
  - GlueCEPolicyMaxCPUTime: maximum CPU time available to jobs submitted to the CE, in minutes
  - GlueCEPolicyMaxTotalJobs: maximum allowed total number of jobs in the queue
  - GlueCEPolicyMaxRunningJobs: maximum allowed number of running jobs in the queue
  - GlueCEPolicyPriority: information about the service priority
  - GlueCEPolicyAssignedJobSlots: maximum number of single-processor jobs that can be running at a given time
- **Access control (objectclass `GlueCEAccessControlBase`)**
  - GlueCEAccessControlBaseRule: a rule defining any access restrictions to the CE. Current semantic: VO = a VO name, DENY = an X.509 user subject
- **Job (currently not filled, the Logging and Bookkeeping service can provide this information) (objectclass `GlueCEJob`)**
  - GlueCEJobLocalOwner: local user name of the job's owner
  - GlueCEJobGlobalOwner: GSI subject of the real job's owner
  - GlueCEJobLocalID: local job identifier
  - GlueCEJobGlobalId: global job identifier
  - GlueCEJobGlueCEJobStatus: job status: SUBMITTED, WAITING, READY, SCHEDULED, RUNNING, ABORTED, DONE, CLEARED, CHECKPOINTED
  - GlueCEJobSchedulerSpecific: any scheduler specific information
- **Cluster (objectclass `GlueCluster`)**
  - GlueClusterUniqueID: unique identifier for the cluster
  - GlueClusterName: human-readable name of the cluster

  The attribute GlueClusterService is deprecated from version 1.2 of the GLUE schema.
- **Subcluster (objectclass `GlueSubCluster`)**
  - GlueSubClusterUniqueID: unique identifier for the subcluster

- GlueSubClusterName: human-readable name of the subcluster
- GlueSubClusterTmpDir: path of temporary directory shared among worker nodes
- GlueSubClusterWNTmpDir: path of temporary directory local to the worker nodes
- GlueSubClusterPhysicalCPUs: total number of real CPUs in the subcluster
- GlueSubClusterLogicalCPUs: total number of logical CPUs (e.g. with hyperthreading)

- **Host (objectclass `GlueHost`)**
  - `GlueHostUniqueId`: unique identifier for the host
  - `GlueHostName`: human-readable name of the host
- **Architecture (objectclass `GlueHostArchitecture`)**
  - `GlueHostArchitecturePlatformType`: platform description
  - `GlueHostArchitectureSMPSize`: number of CPUs in a SMP node
- **Processor (objectclass `GlueHostProcessor`)**
  - `GlueHostProcessorVendor`: name of the CPU vendor
  - `GlueHostProcessorModel`: name of the CPU model
  - `GlueHostProcessorVersion`: version of the CPU
  - `GlueHostProcessorClockSpeed`: clock speed of the CPU
  - `GlueHostProcessorInstructionSet`: name of the instruction set architecture of the CPU
  - `GlueHostProcessorOtherProcessorDescription`: other description for the CPU
  - `GlueHostProcessorCacheL1`: size of the unified L1 cache
  - `GlueHostProcessorCacheL1I`: size of the instruction L1 cache
  - `GlueHostProcessorCacheL1D`: size of the data L1 cache
  - `GlueHostProcessorCacheL2`: size of the unified L2 cache
- **Application software (objectclass `GlueHostApplicationSoftware`)**
  - `GlueHostApplicationSoftwareRunTimeEnvironment`: list of software installed on this host
- **Main memory (objectclass `GlueHostMainMemory`)**
  - `GlueHostMainMemoryRAMSize`: physical RAM
  - `GlueHostMainMemoryRAMAvailable`: unallocated RAM
  - `GlueHostMainMemoryVirtualSize`: size of the configured virtual memory
  - `GlueHostMainMemoryVirtualAvailable`: available virtual memory
- **Benchmark (objectclass `GlueHostBenchmark`)**
  - `GlueHostBenchmarkSI00`: SpecInt2000 benchmark
  - `GlueHostBenchmarkSF00`: SpecFloat2000 benchmark
- **Network adapter (objectclass `GlueHostNetworkAdapter`)**
  - `GlueHostNetworkAdapterName`: name of the network card
  - `GlueHostNetworkAdapterIPAddress`: IP address of the network card

  - `GlueHostNetworkAdapterMTU`: the MTU size for the LAN to which the network card is attached
  - `GlueHostNetworkAdapterOutboundIP`: permission for outbound connectivity
  - `GlueHostNetworkAdapterInboundIP`: permission for inbound connectivity

- **Processor load (objectclass `GlueHostProcessorLoad`)**
  - `GlueHostProcessorLoadLast1Min`: one-minute average processor availability for a single node
  - `GlueHostProcessorLoadLast5Min`: 5-minute average processor availability for a single node
  - `GlueHostProcessorLoadLast15Min`: 15-minute average processor availability for a single node

- **SMP load (objectclass `GlueHostSMPLoad`)**
  - `GlueHostSMPLoadLast1Min`: one-minute average processor availability for a single node
  - `GlueHostSMPLoadLast5Min`: 5-minute average processor availability for a single node
  - `GlueHostSMPLoadLast15Min`: 15-minute average processor availability for a single node

- **Operating system (objectclass `GlueHostOperatingSystem`)**
  - `GlueHostOperatingSystemOSName`: OS name
  - `GlueHostOperatingSystemOSRelease`: OS release
  - `GlueHostOperatingSystemOSVersion`: OS or kernel version

- **Local file system (objectclass `GlueHostLocalFileSystem`)**
  - `GlueHostLocalFileSystemRoot`: path name or other information defining the root of the file system
  - `GlueHostLocalFileSystemSize`: size of the file system in bytes
  - `GlueHostLocalFileSystemAvailableSpace`: amount of free space in bytes
  - `GlueHostLocalFileSystemReadOnly`: true if the file system is read-only
  - `GlueHostLocalFileSystemType`: file system type
  - `GlueHostLocalFileSystemName`: the name for the file system
  - `GlueHostLocalFileSystemClient`: host unique identifier of clients allowed to remotely access this file system

- **Remote file system (objectclass `GlueHostRemoteFileSystem`)**
  - `GlueHostLRemoteFileSystemRoot`: path name or other information defining the root of the file system
  - `GlueHostRemoteFileSystemSize`: size of the file system in bytes
  - `GlueHostRemoteFileSystemAvailableSpace`: amount of free space in bytes
  - `GlueHostRemoteFileSystemReadOnly`: true if the file system is read-only
  - `GlueHostRemoteFileSystemType`: file system type
  - `GlueHostRemoteFileSystemName`: the name for the file system
  - `GlueHostRemoteFileSystemServer`: host unique id of the server which provides access to the file system

- **Storage device (objectclass `GlueHostStorageDevice`)**
  - `GlueHostStorageDeviceName`: name of the storage device

- `GlueHostStorageDeviceType`: storage device type
- `GlueHostStorageDeviceTransferRate`: maximum transfer rate for the device
- `GlueHostStorageDeviceSize`: size of the device
- `GlueHostStorageDeviceAvailableSpace`: amount of free space

- **File (objectclass `GlueHostFile`)**
  - `GlueHostFileName`: name for the file
  - `GlueHostFileSize`: file size in bytes
  - `GlueHostFileCreationDate`: file creation date and time
  - `GlueHostFileLastModified`: date and time of the last modification of the file
  - `GlueHostFileLastAccessed`: date and time of the last access to the file
  - `GlueHostFileLatency`: time taken to access the file, in seconds
  - `GlueHostFileLifeTime`: time for which the file will stay on the storage device
  - `GlueHostFileOwner`: name of the owner of the file
- **Location (objectclass `GlueLocation`)**
  - `GlueLocationLocalID`: local ID for the location
  - `GlueLocationName`: name
  - `GlueLocationPath`: path
  - `GlueLocationVersion`: version
- **VO View (objeclass `GlueVOView`)**
  - `GlueVOViewLocalID`: local ID for this VO view

G.4. Attributes for the Storage Element

These are attributes that give information about a SE and the corresponding storage space. In the GLUE Schema, they are defined in the UML diagram for the Storage Element.

It is worth noting that the `GlueSE` object class, which maps to the `StorageService` element in the GLUE Schema, publishes information of the manager service of a SE; the `GlueSL` object class, which maps to the `StorageLibrary` element, publishes information related to the access node for the SE; and the `GlueSA` object class, which maps to the `StorageSpace` element, gives information about the available space in the SE.

- **Base Class for the storage service (objectclass `GlueSETop`)**
  - No attributes
- **Base Class for the storage library (objectclass `GlueSLTop`)**
  - No attributes
- **Base Class for the storage space (objectclass `GlueSATop`)**

- No attributes

- **Storage Service (objectclass `GlueSE`)**
  - `GlueSEUniqueId`: unique identifier of the storage service (URI)
  - `GlueSEName`: human-readable name for the service
  - `GlueSEPort`: port number that the service listens
  - `GlueSEHostingSL`: unique identifier of the storage library hosting the service
  - `GlueSESizeTotal`: the total size of the storage space managed by the service
  - `GlueSESizeFree`: the size of the storage capacity that is free for new areas for any VO/user
  - `GlueSEArchitecture`: underlying architectural system category

  The attribute `GlueSEType` is deprecated from version 1.2 of the GLUE schema.

- **Storage Service State (objectclass `GlueSEState`)**
  - `GlueSEStateCurrentIOLoad`: system load (for example, number of files in the queue)

- **Storage Service Access Protocol (objectclass `GlueSEAccessProtocol`)**
  - `GlueSEAccessProtocolType`: protocol type to access or transfer files
  - `GlueSEAccessProtocolPort`: port number for the protocol
  - `GlueSEAccessProtocolVersion`: protocol version
  - `GlueSEAccessProtocolSupportedSecurity`: security features supported by the protocol
  - `GlueSEAccessProtocolAccessTime`: time to access a file using this protocol
  - `GlueSEAccessProtocolLocalID`: local identifier
  - `GlueSEAccessProtocolEndpoint`: network endpoint for this protocol
  - `GlueSEAccessProtocolCapability`: function supported by this control protocol

- **Protocol details (objectclass `GlueSEControlProtocol`)**
  - `GlueSEControlProtocolType`: protocol type (e.g. srmv1)
  - `GlueSEControlProtocolVersion`: protocol version
  - `GlueSEControlProtocolLocalID`: local identifier
  - `GlueSEControlProtocolLocalID`: network endpoint for this protocol
  - `GlueSEControlProtocolCapability`: function supported by this control protocol

- **Storage Library (objectclass `GlueSL`)**
  - `GlueSLName`: human-readable name of the storage library
  - `GlueSLUniqueID`: unique identifier of the machine providing the storage service
  - `GlueSLService`: unique identifier for the provided storage service

- **Local File system (objectclass `GlueSLLocalFileSystem`)**
  - `GlueSLLocalFileSystemRoot`: path name (or other information) defining the root of the file system
  - `GlueSLLocalFileSystemName`: name of the file system
  - `GlueSLLocalFileSystemType`: file system type (e.g. NFS, AFS, etc.)

  - `GlueSLLocalFileSystemReadOnly`: true is the file system is read-only
  - `GlueSLLocalFileSystemSize`: total space assigned to this file system
  - `GlueSLLocalFileSystemAvailableSpace`: total free space in this file system
  - `GlueSLLocalFileSystemClient`: unique identifiers of clients allowed to access the file system remotely

- **Remote File system (objectclass `GlueSLRemoteFileSystem`)**
  - `GlueSLRemoteFileSystemRoot`: path name (or other information) defining the root of the file system
  - `GlueSLRemoteFileSystemSize`: total space assigned to this file system
  - `GlueSLRemoteFileSystemAvailableSpace`: total free space in this file system
  - `GlueSLRemoteFileSystemReadOnly`: true is the file system is read-only
  - `GlueSLRemoteFileSystemType`: file system type (e.g. NFS, AFS, etc.)
  - `GlueSLRemoteFileSystemName`: name of the file system
  - `GlueSLRemoteFileSystemServer`: unique identifier of the server exporting this file system

- **File Information (objectclass `GlueSLFile`)**
  - `GlueSLFileName`: file name
  - `GlueSLFileSize`: file size
  - `GlueSLFileCreationDate`: file creation date and time
  - `GlueSLFileLastModified`: date and time of the last modification of the file
  - `GlueSLFileLastAccessed`: date and time of the last access to the file
  - `GlueSLFileLatency`: time needed to access the file
  - `GlueSLFileLifeTime`: file lifetime
  - `GlueSLFilePath`: file path

- **Directory Information (objectclass `GlueSLDirectory`)**
  - `GlueSLDirectoryName`: directory name
  - `GlueSLDirectorySize`: directory size
  - `GlueSLDirectoryCreationDate`: directory creation date and time
  - `GlueSLDirectoryLastModified`: date and time of the last modification of the directory
  - `GlueSLDirectoryLastAccessed`: date and time of the last access to the directory
  - `GlueSLDirectoryLatency`: time needed to access the directory
  - `GlueSLDirectoryLifeTime`: directory lifetime
  - `GlueSLDirectoryPath`: directory path

- **Architecture (objectclass `GlueSLArchitecture`)**
  - `GlueSLArchitectureType`: type of storage hardware (i.e. disk, RAID array, tape library, etc.)

- **Performance (objectclass `GlueSLPerformance`)**
  - `GlueSLPerformanceMaxIOCapacity`: maximum bandwidth between the service and the network

- **Storage Space (objectclass `GlueSA`)**

  - `GlueSARoot`: pathname of the directory containing the files of the storage space
  - `GlueSALocalID`: local identifier
  - `GlueSAPath`: root path of the area
  - `GlueSAType`: guarantee on the lifetime for the storage area (permanent, durable, volatile, other)
  - `GlueSAUniqueID`: unique identifier

- **Policy (objectclass `GlueSAPolicy`)**
  - `GlueSAPolicyMaxFileSize`: maximum file size
  - `GlueSAPolicyMinFileSize`: minimum file size
  - `GlueSAPolicyMaxData`: maximum allowed amount of data that a single job can store
  - `GlueSAPolicyMaxNumFiles`: maximum allowed number of files that a single job can store
  - `GlueSAPolicyMaxPinDuration`: maximum allowed lifetime for non-permanent files
  - `GlueSAPolicyQuota`: total available space
  - `GlueSAPolicyFileLifeTime`: lifetime policy for the contained files

- **State (objectclass `GlueSAState`)**
  - `GlueSAStateAvailableSpace`: total space available in the storage space (in kilobytes)
  - `GlueSAStateUsedSpace`: used space in the storage space (in kilobytes)

- **Access Control Base (objectclass `GlueSAAccessControlBase`)**
  - `GlueSAAccessControlBase Rule`: list of the access control rules

G.5. Attributes for the CE-SE Binding

The CE-SE binding schema represents a mean for advertising relationships between a CE and a SE (or several SEs). This is defined by site administrators and is used when scheduling jobs that must access input files or create output files from or to SEs. In the GLUE Schema, they are defined in the UML diagram for the Computing Element - Storage Service - Bind.

- **Associations between an CE and one or more SEs (objectclass `GlueCESEBindGroup`)**
  - `GlueCESEBindGroupCEUniqueID`: unique ID for the CE
  - `GlueCESEBindGroupSEUniqueID`: unique ID for the SE

- **Associations between an SE and a CE (objectclass `GlueCESEBind`)**
  - `GlueCESEBindCEUniqueID`: unique ID for the CE
  - `GlueCESEBindCEAccesspoint`: access point in the cluster from which CE can access a local SE
  - `GlueCESEBindSEUniqueID`: unique ID for the SE
  - `GlueCESEBindMountInfo`: information about the name of the mount directory on the worker nodes of the CE and the exported directory from the SE
  - `GlueCESEBindWeight`: it expresses a preference when multiple SEs are bound to a CE

G.6. The DIT used by the MDS

The DITs used in the local BDIIs, the GRISes of a CE and a SE are shown in the Figures 15, 16, 17, 18 and 19. The GRIS of a CE contains information for computing resources (different entries for the different queues) and also for the computing-storage service relationships. A GRIS located in a SE publishes information for the storage resources.

The DITs of every GRIS in a site are included in the DIT of the site BDII, grouping all the information of the Grid resources in that site. The information of the different site BDIIs is compiled in a global BDII, as described in Section 5.1.5.

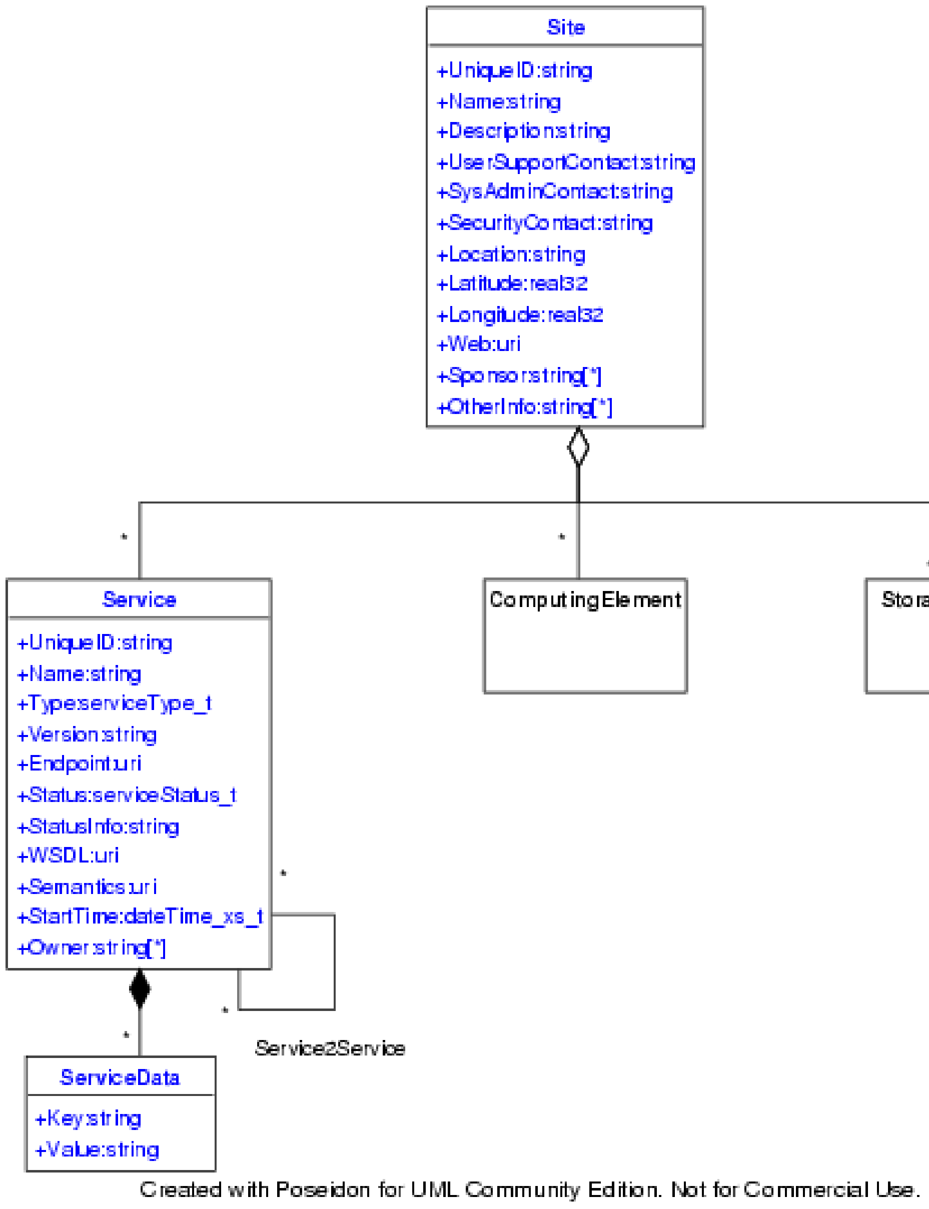


Figure 15: DIT for the core information

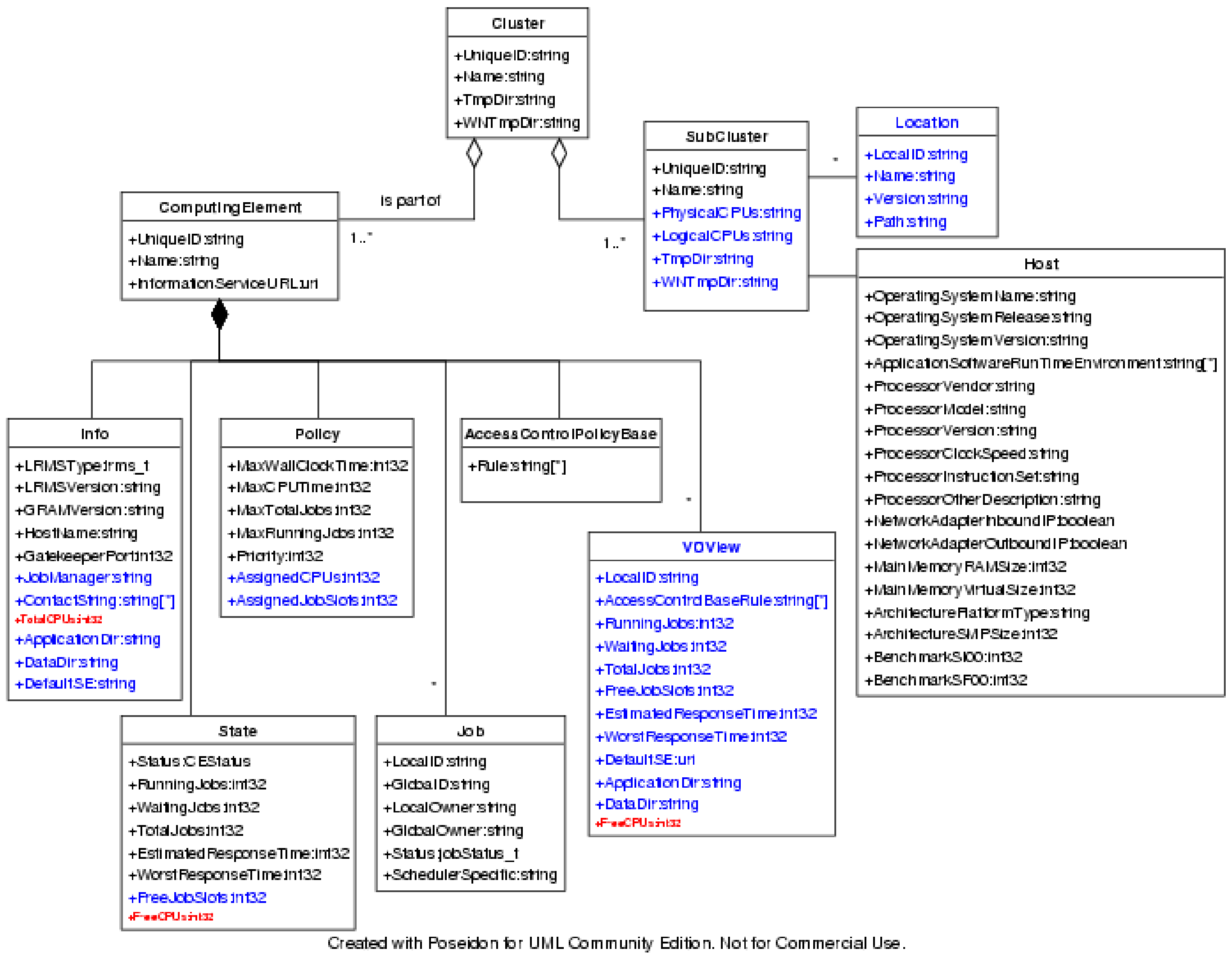


Figure 16: DIT for the computing resources

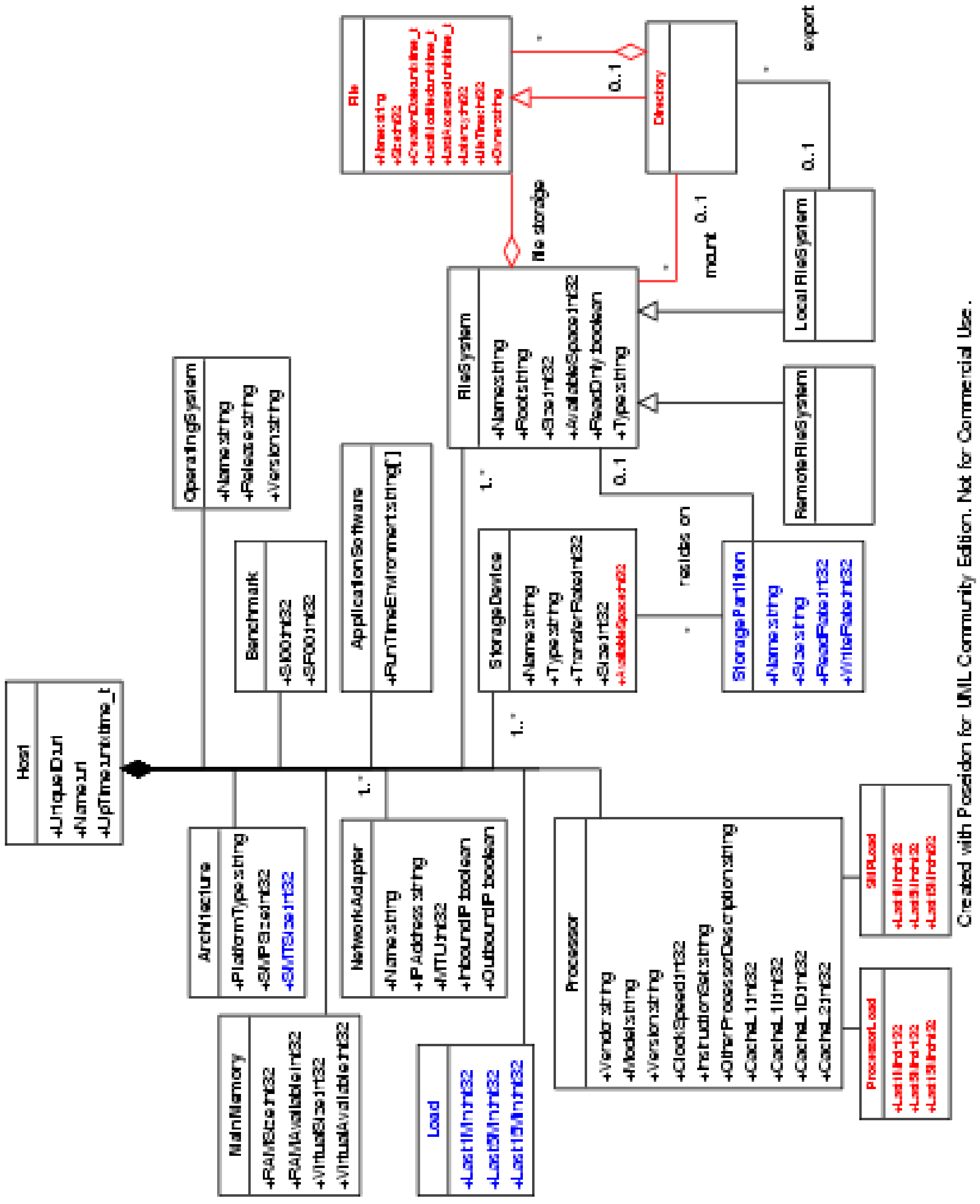


Figure 17: DIT for the worker nodes

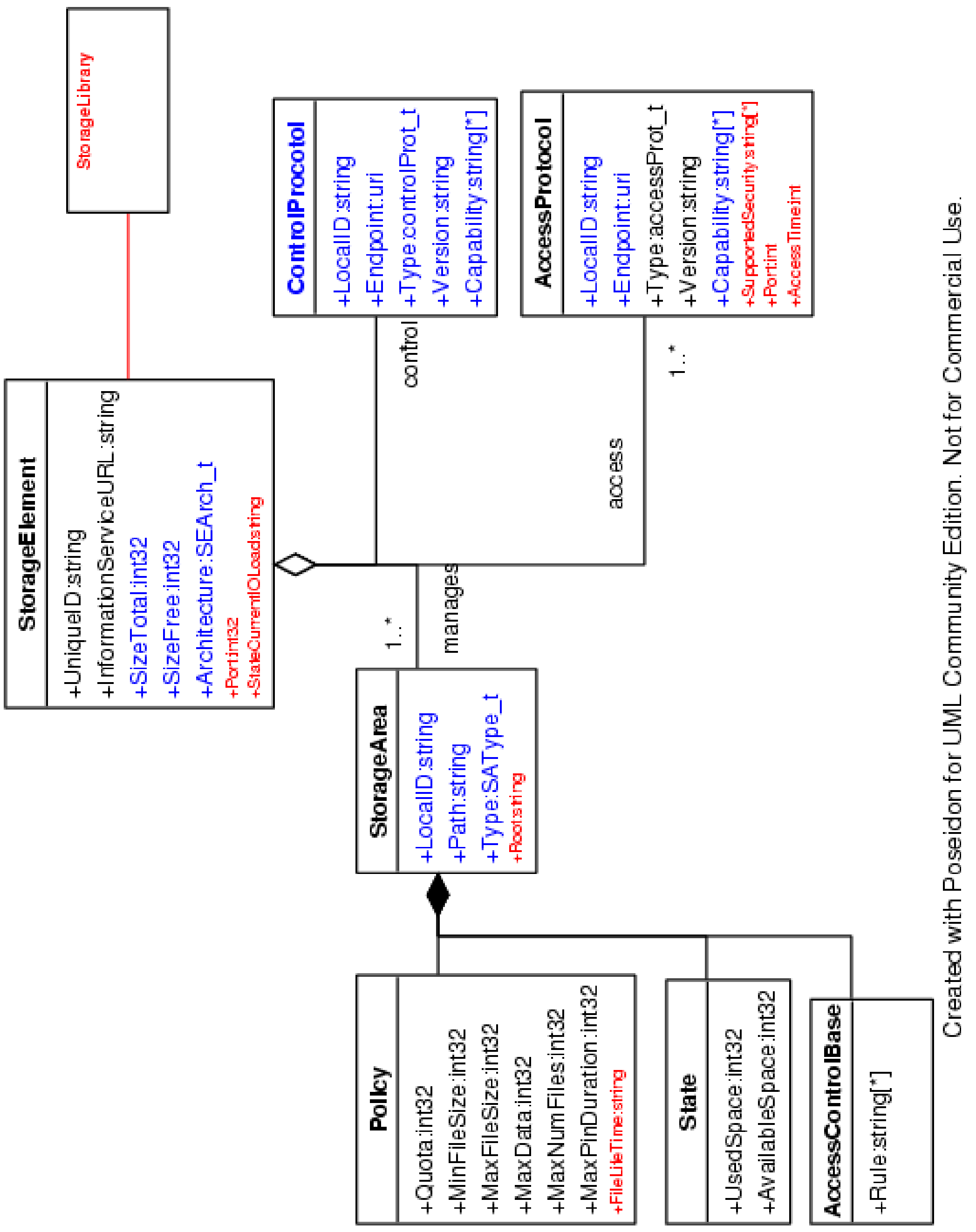


Figure 18: DIT for the storage resources

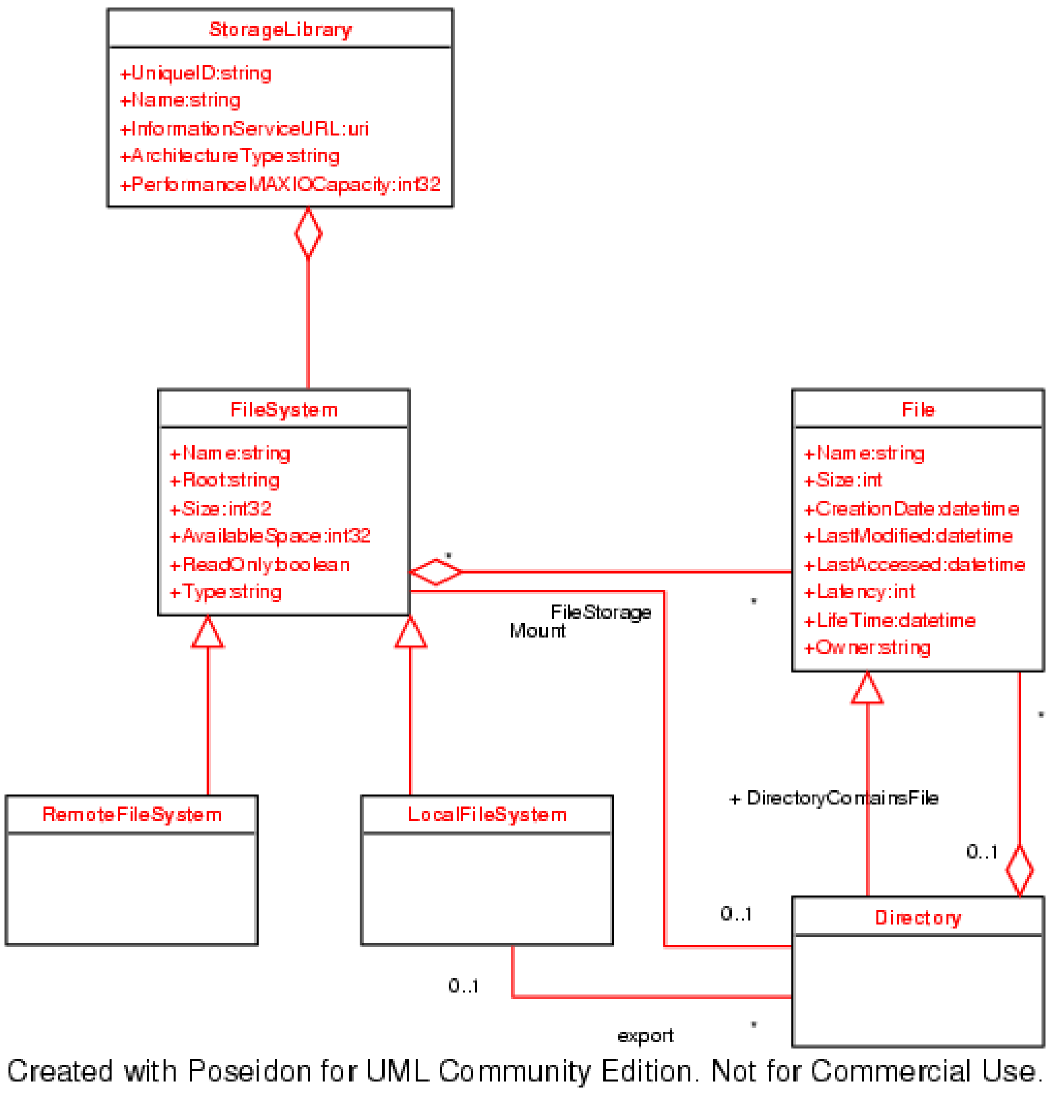


Figure 19: DIT for the storage libraries