LHC COMPUTING GRID

LCG-2 USER GUIDE

MANUALS SERIES

Document identifier:	**CERN-LCG-GDEIS-454439**
EDMS id:	**454439**
Version:	**2.1**
Date:	**September 7, 2004**
Section:	**LCG Experiment Integration and Support**
Document status:	**PUBLIC**
Author(s):	Antonio Delgado Peris, Patricia Méndez Lorenzo, Flavia Donno, Andrea Sciabà, Simone Campana, Roberto Santinelli
File:	**LCG-2-UserGuide**

Abstract: *This guide is an introduction to the LCG-2 Grid from a user's point of view*

Document Change Record

Issue	Item	Reason for Change
09/03/04	v1.0	First Draft
17/03/04	v1.1	Corrections from EIS group comments
13/04/04	v1.2	Some minor corrections
02/06/04	v2.0	Public. Enhancement of the IS part. New appendix: Experiments SW Installation.
07/09/04	v2.1	New LCG Data Management tools. GFAL. Corrections.

Files

Software Products	User files
PDF	https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf
PS	https://edms.cern.ch/file/454439//LCG-2-UserGuide.ps
HTML	https://edms.cern.ch/file/454439//LCG-2-UserGuide.html

CONTENTS

1. INTRODUCTION

1.1. ACKNOWLEDGEMENTS

This work received support from the following institutions:

- Istituto Nazionale di Fisica Nucleare, Roma, Italy.
- Ministerio de Educación y Ciencia, Madrid, Spain.

1.2. OBJECTIVES OF THIS DOCUMENT

This document gives an overview of the main services of the LCG-2 facility. It allows users to understand the building blocks and the available interfaces to the GRID tools 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 users and site administrators of the LCG-2 facility who would like to work on the LCG-2 service.

1.4. DOCUMENT EVOLUTION PROCEDURE

This document updates the previous LCG-1 User Guide ([R1]).

The guide reflects the current status of the LCG-2 service, and will be modified accordingly with the new LCG-2 releases. In some points of the document, references to the foreseeable future of the LCG-2 service are made.

1.5. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

APPLICABLE DOCUMENTS

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

[A2] LDAP Services User Guide
http://hepunx.rl.ac.uk/edg/wp3/documentation/wp3-ldap_user_guide.html

[A3] LCG-1 User Scenario
https://edms.cern.ch/document/414211/

[A4] LCG-2 Frequently Asked Questions
https://edms.cern.ch/document/495216/

REFERENCES

[R1] LCG-1 User Guide
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/LCG-1-UserGuide.htm

[R2] LHC Computing Grid Project
http://lcg.web.cern.ch/LCG/

[R3] Enabling Grids for E-science in Europe
http://eu-egee.org

[R4] Regional Centres for LHC computing
The MONARC Architecture Group
http://barone.home.cern.ch/barone/monarc/RCArchitecture.html
http://monarc.web.cern.ch/MONARC/

[R5] LCG User Developer Guide
http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=eis/docs

[R6] LCG Middleware Developers Guide
http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=documentation

[R7] The Anatomy of the Grid.
Enabling Scalable Virtual Organizations
Ian Foster, Carl Kesselman, Steven Tuecke
http://www.globus.org/research/papers/anatomy.pdf

[R8] Overview of the Grid Security Infrastructure
http://www-unix.globus.org/security/overview.html

[R9] Resource Management
http://www-unix.globus.org/developer/resource-management.html

[R10] GSIFTP Tools for the Data Grid http://www.globus.org/datagrid/deliverables/gsiftp-tools.html

[R11] RFIO: Remote File Input/Output
http://doc.in2p3.fr/doc/public/products/rfio/rfio.html

[R12] The GLUE schema
http://www.cnaf.infn.it/~sergio/datatag/glue/

[R13] MDS 2.2 Features in the Globus Toolkit 2.2 Release
http://www.globus.org/mds/

[R14] European DataGrid Project
http://eu-datagrid.web.cern.ch/eu-datagrid/

[R15] WP1 Workload Management Software – Administrator and User Guide. Nov 24th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf

[R16] LCG-2 Manual Installation Guide
https://edms.cern.ch/file/434070//LCG2Install.pdf

[R17] Classified Advertisements. Condor.
http://www.cs.wisc.edu/condor/classad

[R18] The Condor Project.
http://www.cs.wisc.edu/condor/

[R19] Job Description language HowTo. December 17th, 2001
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf

[R20] JDL Attributes – Release 2.x. Oct 28th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

[R21] The EDG-Brokerinfo User Guide - Release 2.x. 6th August 2003
http://server11.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2_2.pdf

[R22] Workload Management Software – GUI User Guide. Nov 24th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0143-0_0.pdf

[R23] User Guide for the EDG Local Replica Catalog 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-lrc-userguide.pdf

[R24] User Guide for the EDG Replica Metadata Catalog 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-rmc-userguide.pdf

[R25] User Guide for the EDG Replica Optimization Service 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-ros-userguide.pdf

[R26] User Guide for the Replica Location Index 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-rli-userguide.pdf

[R27] 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

[R28] GridICE: a monitoring service for the Grid
http://server11.infn.it/gridice/

[R29] Experiment software installation on LCG-1
http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=eis/docs

[R30] User Guide for the EDG Replica Manager 1.5.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-replica-manager-userguide.pdf

[R31] EDG Tutorial – Handout for Participants for EDG Release 2.x
http://edms.cern.ch/document/393671

1.6. TERMINOLOGY

1.6.1. Glossary

API: Application Programming Interface
BDII: Berkeley Database Information Index
CATOR CERN Advanced STORage manager
CE: Computing Element
CERN: European Laboratory for Particle Physics
ClassAd: Classified advertisement
CLI: Command Line Interface
CNAF: INFN's National Center for Telematics and Informatics
DIT: Directory Information Tree
DN: Distinguished Name (LDAP's)
EDG: European DataGrid
EDT: European DataTag
EGEE: Enabling Grids for E-science in Europe
ESM: Experiment Software Manager
FNAL: Fermi National Accelerator Laboratory
GIIS: Grid Index Information Server
GLUE: Grid Laboratory for a Uniform Environment
GOC: Grid Operations Centre
GRAM: Globus Resource Allocation Manager
GRIS: Grid Resource Information Service
GSI: Grid Security Infrastructure
GUI: Graphical User Interface
GUID: Grid Unique ID
ID: Identifier
INFN: Instituto Nazionale di Fisica Nucleare
IS: Information Service
JCS: Job Control Service
JDL: Job Description Language
LB: Logging and Bookkeeping Service
LDAP: Lightweight Directory Access Protocol
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
NS: Network Server
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
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
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 Organisation
WMS: Workload Management System
WN: Worker Node
WPn: Work Package #n

2. EXECUTIVE SUMMARY

This user guide is intended for users of the LCG-2 service. Within these pages, the user will hopefully find an adequate introduction to the services provided by the Grid and a description of how to use them. Examples are given for the management of jobs and data, the monitoring of resources status, etc., in order to be effective easily.

A first introduction on the organization of the service itself is presented in Chapter 3. The reader can skip this chapter if he/she is familiar with the basic architecture of the LCG-2 service already. In Chapter 4, the procedures to register with LCG, get a certificate and manage proxies are described.

An overview of the Workload Management service is given in Chapter 5. The chapter explains the basic commands for job submission and management, as well as those for retrieving information related to the Workload Management match-making mechanism from inside a Grid job.

Data Management services are described in Chapter 6. Not only the high-level interface is described but also commands that can be useful in case of problems or for debugging purposes.

Details on how to get information about the status of LCG-2 resources are given in Chapter 7, where the Information System is discussed. Many examples are provided to interrogate GRISes, the LCG-2 top GIISes, and the BDII.

Finally, the appendices give information about the installation of software in Grid resources (Appendix A, the old EDG Replica Management commands (Appendix B), the GLUE Schema used to describe LCG-2 resources (Appendix C), the version of the middleware used in LCG-2 (Appendix D) and a description of the possible states of a job during submission and execution (Appendix E).

3. OVERVIEW

The Large Hadron Collider (***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 accelerator will start operation in 2007, and the experiments that will use it will generate large amounts of data. The processing of this data will require enormous computational and storage resources.

The job of the LHC Computing Grid Project [R2] –***LCG***– is to prepare the computing infrastructure for the simulation, processing and analysis of LHC data for all four of the LHC collaborations: ***ALICE, ATLAS, CMS*** and ***LHCB***. This includes both the common infrastructure of libraries, tools and frameworks required to support the physics application software, and the development and deployment of the computing services needed to store and process data, providing batch and interactive facilities for the worldwide community of physics involved in LHC.

The requirements for LHC data handling are very large, in terms of computational power, data storage capacity, data access performance and the associated human resources for operation and support. It is not considered feasible to fund all of the resources at one site, and so it has been agreed that the LCG computing service will 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. Special software, referred to generically as ***Grid Middleware***, will hide 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 LCG project is also one of the two pilot applications selected to guide the implementation and certify the performance and functionality of the evolving European Grid infraestructure. The other one is Biomedical Grids. This is done as a part of the ***EGEE project (Enabling Grids for E-science in Europe)*** [R3], which aims to integrate current national, regional and thematic Grid efforts, in order to create a seamless European Grid infraestructure for the support of the European Research Area. The EGEE project is a two-year project conceived as part of a four-year programme.

In LCG-2, the source of experiments data, CERN, is called the Tier 0 centre. The rest of sites will store and process part of that data. The sites are divided in Tier 1 and Tier 2 centres. Tier 1 centres are sites that have a significant amount of storage resources, while Tier 2 centres may have them, but not necessarily.

In the first phase of the project, from 2002 through 2005, LCG will develop and prototype the computing services and deploy a series of computing data challenges of increasing size and complexity to demonstrate the effectiveness of the software and computing models selected by the experiments.

LCG-2 is the new release of LCG (after LCG-1, see [R1]). This new version will be running in 2004 and its main goal is to provide a stable service. LCG-2 expands the services of LCG-1, with enough resources and functionality for the *2004 Computing Data Challenge*. In addition, more Tier 1 and Tier 2 centres will join the project, following the Monarc model [R4], as in the previous LCG-1 release.

In the first phase of LCG-2, the core sites implementing the new release are CERN, Karlsruhe, Barcelona, FNAL, CNAF, Nikhef, Taipei and RAL.

3.1. PRELIMINARY MATTERS

3.1.1. Code Development

Although this is a user guide and not a developers guide, it is worth noting that many of the services offered by LCG-2 can be accesed both by directly using the user interfaces provided (Command Line Interface (CLI) or Graphical User Interface (GUI)), or from applications by making use of the different Application Programming Interfaces (API). This is the case of an application that wants to submit a job to the Grid, or of a job itself that needs to move files using a Data Management API.

General information regarding the different APIs that can be used to access the LCG resources is given in [R5]. In addition, other references of APIs used for particular services will be given later in the sections describing such services.

A complete different matter is the development of software that forms part of the LCG-2 Grid middleware itself. This falls completely out of the scope of this guide, as that is not a topic for LCG-2 users, but for LCG-2 developers. If, however, a reader is interested in this subject, he/she can refer to [R6].

3.1.2. Troubleshooting

This document will give advice on some common usage errors and the messages these produce; it will also give advice on how to avoid them. The guide cannot, however, thoroughly include all the possible unexpected failures a user may find while using LCG-2. These errors may be produced due to his/her own mistakes, to misconfigurations of the Grid components, or even to bugs in the Grid middleware.

The user may find more information in the references that are provided in the different sections, which, in general, deal with the commands and services of LCG-2 in greater detail than this user guide does.

There is also the possibility to get help from the ***Global Grid User Support***, which centralizes the user support for LCG-2, by answering questions, tracking known problems, maintiaining lists of frequently asked questions, etc. The entrance point to this service is a web site, with the following URL:

http://www.ggus.org

Finally, if the user believes that there is a bug in the code of the Grid middleware, or that there is a functionality that it is missing in the Grid and that it should be added, he/she may (and should, for the benefit of other users) submit a bug in the ***Savannah Portal***, whose URL follows:

https://savannah.cern.ch

Not only gives this web site the possibility to open bugs, but it is the central portal for all the software developement for LCG-2 and the LHC experiments, as well as for the related documentation. The specific web for the LCG-2 deployment projects (which will probably be the most interesting one for the readers of this guide) includes a bug tracking system, a patch manager, a task list, and links to the project homepage, and the download and documentation areas. Its URL is the following:

https://savannah.cern.ch/projects/lcgoperation

Both for the Global Grid User Support and for the Savannah Portal, a user must get registered in order to use the services provided.

3.1.3. Experiments Software Installation

Authorized users can install software in the computing resources of LCG-2. The installed software is also published in the Information System, so that users' jobs can run on nodes where the software they need is installed.

Appendix A describes the procedure to install software on LCG-2 nodes, as well as some scripts that are used to achieve that goal.

3.2. THE LCG-2 ARCHITECTURE

This section provides an overview of the LCG-2 architecture.

3.2.1. Getting Started

LCG-2 is organized into Virtual Organizations ([R7]): dynamic collections of individuals and institutions sharing resources in a flexible, secure and coordinated manner. In such settings, we encounter unique authentication, authorization, resource access, resource discovery, and other challenges.

Before LCG resources can be used, a user is required to register some personal data and information about the Virtual Organization he/she belongs to with the ***LCG Registration Server***. CERN will run such a service, collecting information about all LCG users.

The ***Grid Security Infrastructure (GSI)*** in LCG-2 enables secure authentication and communication over an open network ([R8]). GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. Extensions to these standards have been added for single sign-on and delegation.

In order to access Grid resources, a user needs to have a digital X509 certificate from a ***Certification Authority (CA)*** trusted by LCG. The CAs recognized by LCG are listed later on.

In LCG there are five possible Virtual Organizations (VOs) a user can be affiliated to: one for the DTeam (LCG Grid Deployment Group) and one more for each one of the four LHC experiments. A ***Virtual Organization Server*** maps user certificates to user data and lists the certificates as belonging to users that are part of a VO. The VO Server for the DTeam is run at CERN, while the VO Servers for the four experiments are run at NIKHEF. But, although LCG officially supports only those five VOs, many more VOs are possible, since LCG will provide the first Grid infraestructure for EGEE.

Each site installing the new release is free to support any of the existing VOs. Users can be aware of the VOs supported at a given site by asking directly to that site. The commands used for that purpose are shown later.

A user is authorized to use LCG-2 Grid resources by means of the ***grid-mapfile*** mechanism. Each host part of the LCG-2 Grid has a local grid-mapfile which maps user certificates to local accounts. When a user request-for-service reaches a host, the certificate subject of the user is checked against what there is in the local grid-mapfile. If the user certificate is found there, then the local account to which the user certificate is mapped is used to serve the request. The same is true for services. Details are explained in [R8].

The following sections describe several types of services run in LCG-2 to provide the Grid functionality.

3.2.2. The User Interface

The initial point of access to the LCG-2 Grid is the ***User Interface (UI)***. This is a machine where LCG users have a personal account and where the user's certificate is installed. This is the gateway to Grid services. From the UI, a user can be authenticated and authorized to use the LCG-2 Grid resources. This is the component that allows users to access the functionalities offered by the Information, Workload and Data management services. It provides a CLI to perform some basic Grid operations:

- submit a job for execution on a Computing Element;
- list all the resources suitable to execute a given job;
- replicate and copy files;
- cancel one or more jobs;
- retrieve the output of one or more finished jobs;
- show the status of one or more submitted jobs.

Besides this, the LCG-2 API libraries are also available in the UI for users to link Grid applicacions.

One or more UIs are available at each site part of the LCG-2 Grid.

3.2.3. Computing Element and Storage Element

A ***Computing Element (CE)*** is defined as a Grid batch queue and it is identified by a pair `<hostname>:<port>/<batch_queue_name>`. It is important to notice that according to this definition, several queues defined for the same hostname are considered different CEs. This is currently used to define different queues for jobs of different lengths. Examples of CE names are:

```
adc0015.cern.ch:2119/jobmanager-lcgpbs-long
adc0015.cern.ch:2119/jobmanager-lcgpbs-short
```

A Computing Element is built on a homogeneous farm of computing nodes called ***Worker Nodes (WN)*** and a node acting as a ***Grid Gate (GG)*** or front-end to the rest of the Grid. The GG runs a Globus gatekeeper, the Globus GRAM (Globus Resource Allocation Manager) [R9], the master server of a Local Resource Managemente System (LRMS), together with the a local Logging and Bookkeeping server (see later). In LCG-2 the types of LRMS supported are the Portable batch System (PBS), the Load Sharing Facility (LSF) and Condor. While all WNs need only outbound connectivity, the Gate node must be accessible from outside the site. The GG is responsible for accepting jobs and dispatching them for execution to the WNs. The GG provides a uniform interface to the computational resources it manages. On the WNs, all commands and Application Programming Interface (API) for performing actions on Grid resources and Grid data are available.

Each LCG-2 site runs at least one CE and a farm of WNs behind it.

A ***Storage Element (SE)*** provides uniform access and services to large storage spaces. The Storage Element may control large disk arrays, mass storage systems (MSS) and the like. Each LCG-2 site provides one or more SEs.

In the current LCG-2 release, each site includes at least a classic SE, which has a ***GSIFTP*** server [R10][1]. The GSIFTP protocol offers basically the functionality of FTP, i.e. used for the transfers of files, but enhanced to use GSI security. It is responsible for secure, fast and efficient file transfer to/from the Storage Element.

Thus, the main file access protocol in the Grid is GSIFTP, which is used for the transfering of files, and which will be always supported by any Grid storage resource. However, for the remote access (and not only copy) of files stored in the SEs, the protocol to be used is the ***Remote File Input/Output protocol (RFIO)*** [R11].

In the final LCG-2 release, though, the storage resources will be managed by a ***Storage Resource Manager (SRM)***. This middleware module, will make it possible to dinamically manage the contents of

[1] In the literature, the terms *GridFTP* and *GSIFTP* are sometimes used interchangeably. Actually, GSIFTP is a subset of GridFTP. Please refer to [R10] for more information.

the storage resource at any time. The SRM will interact with the operating system, with the mass storage system (to perform file archiving), and with the protocols (to perform file transfer operations). As MSS, LCG-2 will support d-Cache disk pool and tape archiving systems, such as CASTOR (CATOR CERN Advanced STORage manager)[2].

When the SRMs are working, they will still support GSIFTP for file transfer, but their file access protocol will vary depending on the MSS that is behind them. For CASTOR, the protocol will also be RFIO. For dCache, the protocol will be ***gsidcap***, which is the GSI secure version of the ordinary dCache access protocol, *dcap*; and it is possible that *kdcap*, its Kerberos secure version, will be also supported.

The *file* protocol is no longer supported in LCG-2. It may only be used to specify files that are located in a local filesystem (i.e.: in a UI or a WN), but not stored in a Grid SE or SRM.

The resources described up to now constitute the compute and storage power of the LCG-2 Grid. Together with that infrastructure, additional services are provided to locate and report on the status of Grid resources, to access data on the Grid, and to find the most appropriate resources to submit user jobs. These are the ***Information System***, the ***Data Management*** and ***Job Management*** services.

3.2.4. Information System

The Information System (IS) provides information about the LCG-2 Grid resources and their status. The offered data conforms to the ***GLUE (Grid Laboratory for a Uniform Environment) Schema***.The GLUE Schema activity aims to define a common conceptual data model to be used for Grid resources monitoring and discovery. There are three main components of the GLUE Schema. They describe the attributes and value of Computing Elements, Storage Elements and binding information for Computing and Storage Elements. Details can be found in [R12].

In LCG-2, the ***Monitoring and Discovery Service (MDS)*** from Globus [R13] has been adopted as the provider of the Information Service. It implements the GLUE Schema using OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol (LDAP).

Figure 1 shows how the information is stored and propagated. Computing and storage resources at a site report (via the ***Grid Resource Information Servers, or GRISes***) their static and dynamic status to the ***Site Grid Index Information Server (GIIS)***.

Due to dynamic nature of the GRID, the GIISes might not contain information about resources that are actually available on the Grid but that, for some reasons, are unable to publish updated information to the GIISes. Because of this, the ***Berkeley DB Information Index (BDII)*** was introduced. The BDII queries the GIISes and acts as a cache storing information about the Grid status in its database. Each BDII contains information from the site GIISes defined by a configuration file, which it accesses through a web interface. In this way, each site can easily decide which information they desire to publish.

[2] A CERN's hierarchical Storage Manager

Users and other Grid services (such as the RB) can interrogate BDIIs to get information about the Grid status. Very up-to-date information can be found by directly interrogating the site GIISes or the local GRISes that run on the specific resources.

Examples on how to query the Information System in LCG-2 are given later on.

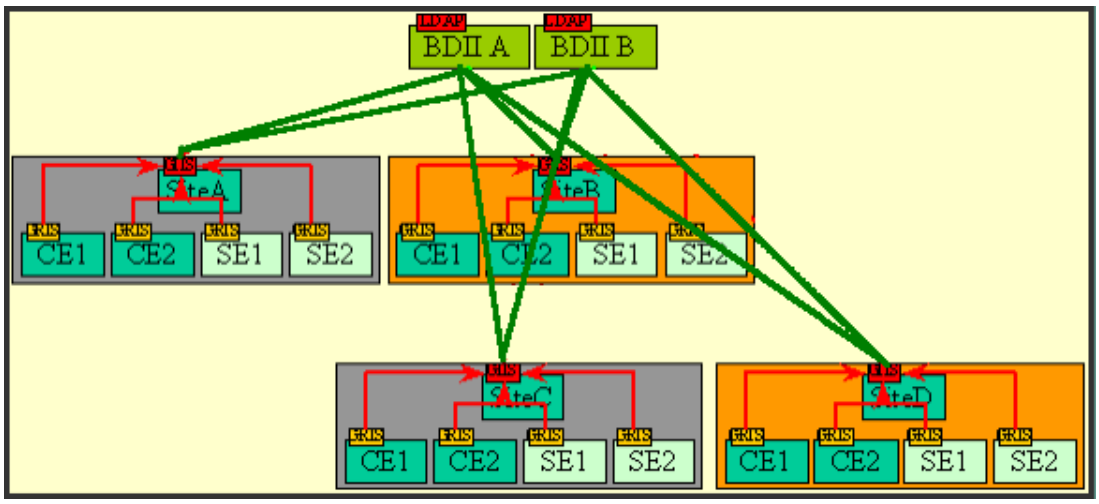

Figure 1: The Information System in LCG-2

3.2.5. Data Management

The Data Management services are provided by the ***Replica Management System (RMS)*** of the ***European DataGrid (EDG)*** [R14] and the LCG Data Management client tools. In a Grid environment, the data files are replicated, possibly on a temporary basis, to many different sites depending on where the data is needed. The users or applications do not need to know where the data is located. They use logical names for the files and the Data Management services are responsible for locating and accessing the data.

The files in the Grid are referenced by different names: ***Grid Unique IDentifier (GUID)***, ***Logical File Name (LFN)***, ***Storage URL (SURL)*** and ***Transport URL (TURL)***. While the GUID or LFN refer to files and not replicas, and say nothing about locations, the SURLs and TURLs give information about where a physical replica is located.

A file can always be identified by its GUID; this is assigned at data registration time and is based on the UUID standard to guarantee unique IDs. A GUID is of the form: `guid:<unique_string>`. All the replicas of a file will share the same GUID. In order to locate a Grid accessible file, the human user will normally use a LFN. LFNs are usually more intuitive, human-readable strings, since they are allocated by the user as GUID aliases. Their form is: `lfn:<any_alias>`.

The SURL is used by the RMS to find where a replica is physically stored, and by the SE to locate

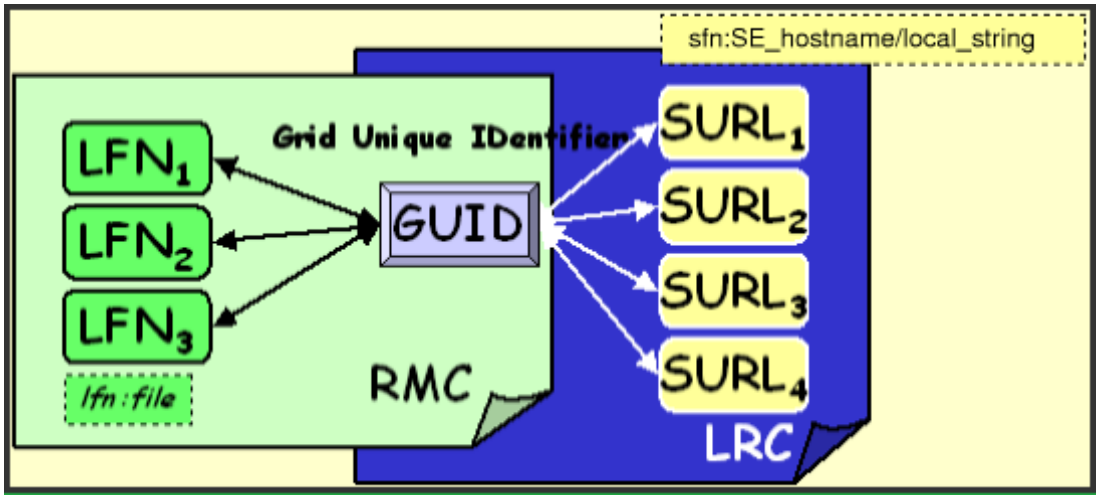

Figure 2: Different filenames in LCG-2

it. Currently, the SURLs are of the form: `sfn:<SE_hostname>/<local_string>`[3], where `<local_string>` is used internally by the SE to locate the file.

Finally, the TURL gives the necessary information to retrieve a physical replica, including hostname, path, protocol and port (as any conventional URL); so that the application can open and retrieve it. Figure 2 shows the relation between the different file names.

The main services offered by the RMS are: the ***Replica Location Service (RLS)*** and the ***Replica Metadata Catalog (RMC)***.

The RLS maintains information about the physical location of the replicas (mapping with the GUIDs). It is composed of several Local Replica Catalogs (LRCs) which hold the information of replicas for a single VO.

The RMC stores the mapping between GUIDs and the respective aliases (LFNs) associated with them, and maintains other metada information (sizes, dates, ownerships. . .)

The last component of the Data Management framework is the ***Replica Manager***. The Replica Manager presents a single interface for the RMS to the user or other services (such as the Resource Broker), and interacts with the other services of the RMS. This is illustrated in Figure 3.

In the LCG-2, the Replica Manager is integrated with the User Interface described earlier. Nevertheless, some of its functionlity has been replaced by a new, faster interface: the LCG Data Management client tools. This new tools will be used by users when managing files and replicas in LCG-2, although the older EDG interface is still used by the Resource Broker. These interfaces are explained in detail in

[3]When SRMs are already working, files stored there will use `srm` as the prefix for their SURLs, instead of `sfn`. This will allow the RMS to distinguish which kind of storage the file is in.

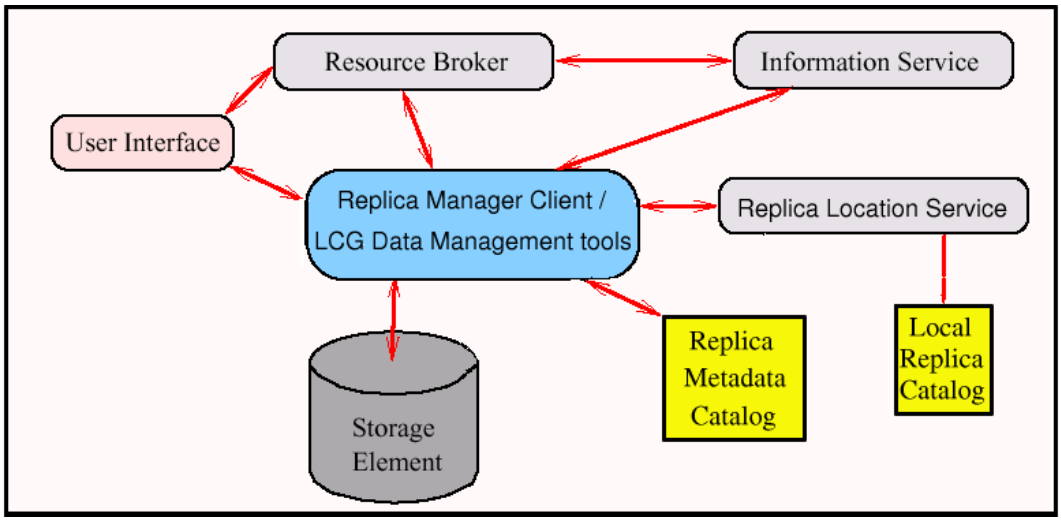

Figure 3: Interactions of the Replica Manager with other Grid components

For the moment these catalogues are centralized and there is one RLS per VO. In the first phase, all RLSs are run at CERN.

3.2.6. Job Management

The services of the ***Workload Management System (WMS)*** are responsible for the acceptance of job submits and the dispatching of those jobs to the appropiate CE, depending on the job requirements and the available resources. For that purpose, the WMS must retrieve information from the BDII, and the RLS. The ***Resource Broker (RB)*** is the machine where the WMS services run. These services are:

- ***Network Server (NS)***, which accepts the incoming job requests from the UI, and provides support for the job control functionality.
- ***Workload Manager***, which is the core component of the system.
- ***Match-Maker*** (also called Resource Broker), whose duty is finding the best resource matching the requirements of a job (match-making process).
- ***Job Adapter***, which prepares the environment for the job and its final description, before passing it to the Job Control Service.
- ***Job Control Service (JCS)***, which finally performs the actual job management operations (job submission, removal...)

In addition, the ***Logging and Bookkeeping service (LB)*** [R15]. is usually also run on a RB machine. The LB logs all job management Grid events, which can then be retrieved by users or system administrators for monitoring or troubleshooting.

Multiple RBs are available in LCG-2 Grid. Participating sites are free to install their own RBs.

The last component of the LCG-2 Grid described here is the ***Proxy Server (PS)***. When a user accesses the Grid, he/she is provided with a temporary certificate, called proxy, that has an expiration time. If the user proxy expires before the user job has finished, all subsequent requests for service will fail due to unauthorized access. In order to avoid this, the Workload Management Service provided by EDG allows for proxy renewal before the expiration time has been reached if the job requires it. The PS is the component that allows for such functionality.

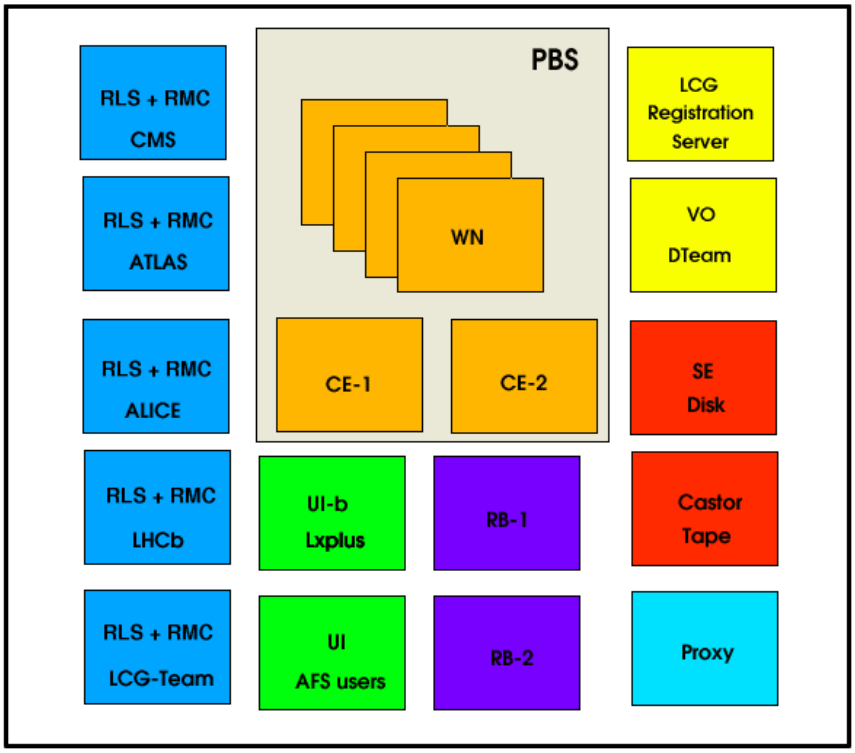

Figure 4: LCG-2 available services at CERN

In LCG-2, a site is free to install a PS. The list of sites that have installed a PS can be consulted at the LCG Grid Operations Centre, described in section 4.4.

Figure4 shows a summary of all LCG-2 service components available at CERN.

3.3. SERVICE INTERACTIONS AND JOB FLOW

This section describes briefly what happens when a user submits a job to the LCG-2 Grid to process some data and how the different components interact. A description of the components of the Data Management system is also given. User applications and further functionality can be built/developed on top of what is offered by LCG-2 Grid.

3.3.1. Job Submission

Figure 5 illustrates the process that is followed by a job submitted to the Grid. The individual steps are described after it.

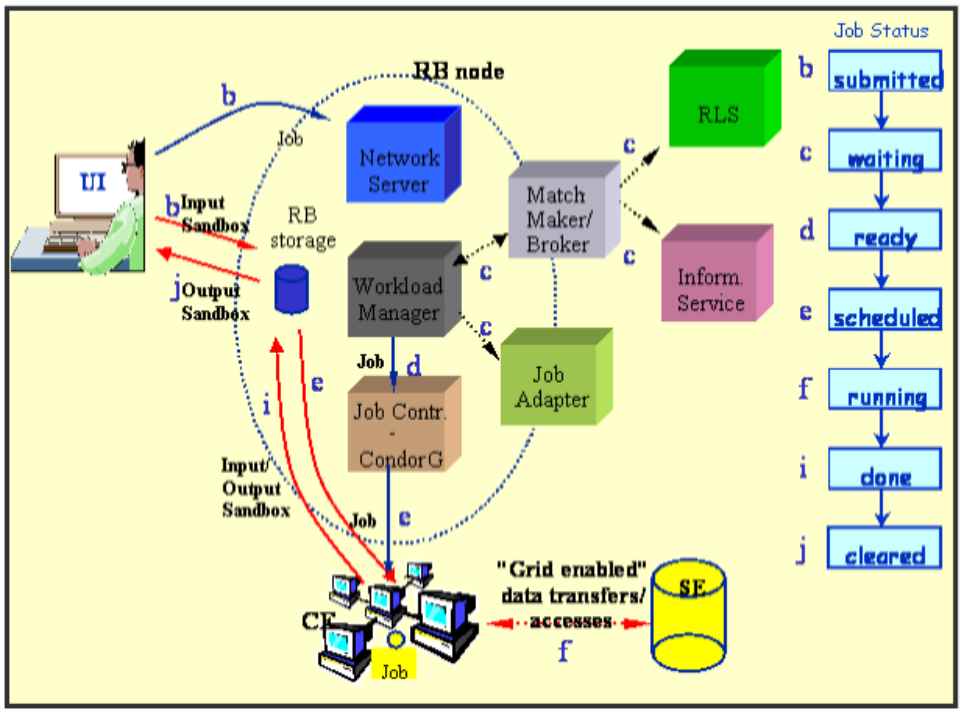

Figure 5: Job flow in the LCG-2

a. After obtaining a digital certificate from one of the LCG-2 trusted Certification Authorities, registering with LCG-2, registering with a Virtual Organization and obtaining an account on an LCG-2 User Interface (once only actions), the user is ready to use LCG-2 Grid. He/she logs to the UI machine and creates a proxy certificate that authenticates him/her in every secure interaction, and has a limited lifetime.

b. The user submits the job from the UI to the WMS, where the job will be executed on a computing

node. The user can specify in the job description file one or more files to be copied from the UI to the RB node; this set of files is called ***Input Sandbox***. The event is logged in the LB and the status of the job is SUBMITTED.

c. The WMS, and in particular the Match-Maker component, looks for the best available CE to execute the job. To do so, the Match-Maker interrogates the BDII to query the status of computational and storage resources and the RLS to find location of data. The event is logged in the LB and the status of the job is WAIT.

d. The WMS Job Adapter prepares the job for submission creating a wrapper script that is passed, together with other parameters, to the JCS for submission to the selected CE. The event is logged in the LB and the status of the job is READY.

e. The Globus Gatekeeper on the CE receives the request and sends the Job for execution to the LRMS (e.g. PBS, LSF or Condor). The 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's files are copied from the RB to the WN where the job is executed. The event is logged in the LB and the status of the job is RUNNING.

g. While the job runs, Grid files can be accessed on a (close) SE[4] using either the RFIO protocol or local access if the files are copied to the WN local filesystem. In order for the job to find out which is the close SE, or what is the result of the Match-Maker process, a file with this information is produced by the WMS and shipped together with the job to the WN. This is known as the `.BrokerInfo` file. Information can be retrieved from this file using the BrokerInfo CLI or the API library.

h. The job can produce new output data that 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 on a Storage Element and registering its location, metadata and attribute to the RMS. At the same time, during job execution or from the User Interface, data files can be replicated between two SEs using again the Data Management tools.

i. If the job reaches the end 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. The 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/her job from the UI using the WMS CLI or API. The event is logged in the LB and the status of the job is CLEARED.

k. Queries of the job status are addressed to the LB database from the UI machine. Also, from the UI it is possible to query the BDII for a status of the resources.

[4]Currently, that an SE is considered "close" to a CE is just a question of the CE declaring it that way. A CE declares some SEs (probably, those in its own network or reachable through a high bandwith connection) as "close". The other SEs are considered "not close". This will behave in a more dynamical way in the future.

l. If the site where the job is being run falls down, the job will be automatically resent to another CE that is analogue to the previous one, and following the same requirements the user asked for. In the case that this new submission is disabled, the job will be marked as aborted. Users can get information about what happened by simply questioning the LB service.

3.3.2. Data Management

The Input/Output Sandbox is a mechanism for transferring small data files needed to start the job or to check the final status over the Grid. Large data files are available on the Grid and known to other users only if they are stored on SEs and registered in the RMS catalogues. In order to optimise data access and to introduce fault-tolerance and redundancy, data files can be replicated on the Grid. The EDG Replica Manager interface and the LCG Data Management client tools are available for performing these tasks. Only anonymous access to the data catalogues is supported: the user proxy is not used to control the access to them.

In the LCG-2, as explained earlier, a file is identified uniquely by the GUID, but the user may refer to ir using different aliases. Also, there will probably be several physical replicas of each file. The user should never interact with the RMC or the RLS catalog directly. Instead, he/she should always use the LCG tools, the EDG RM, or the POOL interface (see section 6.6).

m. When a new file is produced, the file should be uploaded to the Grid to be known and usable by Grid services or other Grid users. This can be done using the LCG Data Management commands for copying and registering a file.

n. Before running a job on the Grid, the user can ask the WMS to run the job on a CE close to an SE containing the data of interest, or, at run time, the job can ask the RMS to replicate a file on a SE close or even on the WN where the job is running.

o. If a file is no longer needed, it can be deleted from the Grid and all its references removed from the data catalogues.

3.3.3. Information System

The architecture of the Information System in the LCG-2 Grid has been already described. Users can interrogate the IS to retrieve static or dynamic information about the status of the LCG-2. In order to have an optimal answer, users are encouraged to query the BDIIs or the site GIISes. Also, the specific GRISes can be queried. Details and examples on how to interrogate GRIS, GIIS and BDII are given in Chapter 7.

As explained, the current IS is based on LDAP: a directory service infraestructure. A directory service is a specialized database optimized for reading, browsing and searching information. No transaction or roll-back features are normally offered. In particular in LCG-2 Grid, only anonymous access to the

catalogue is offered. **This means that all users can browse the catalogues and all services are allowed to enter information into it.**

The LDAP information model is based on entries. An *entry* usually describes an object such as a person, a computer, a server, and so on. Each entry contains one or more *attributes* that describe the entry. Each attribute has a type and one or more *values*. Each entry has a name called a ***Distinguished Name (DN)*** that uniquely identifies it. A DN is formed by a sequence of attributes and values. Based on their DNs, the entries can be arranged into a hierarchical tree-like structure. This tree of directory entries is called the ***Directory Information Tree (DIT)***.

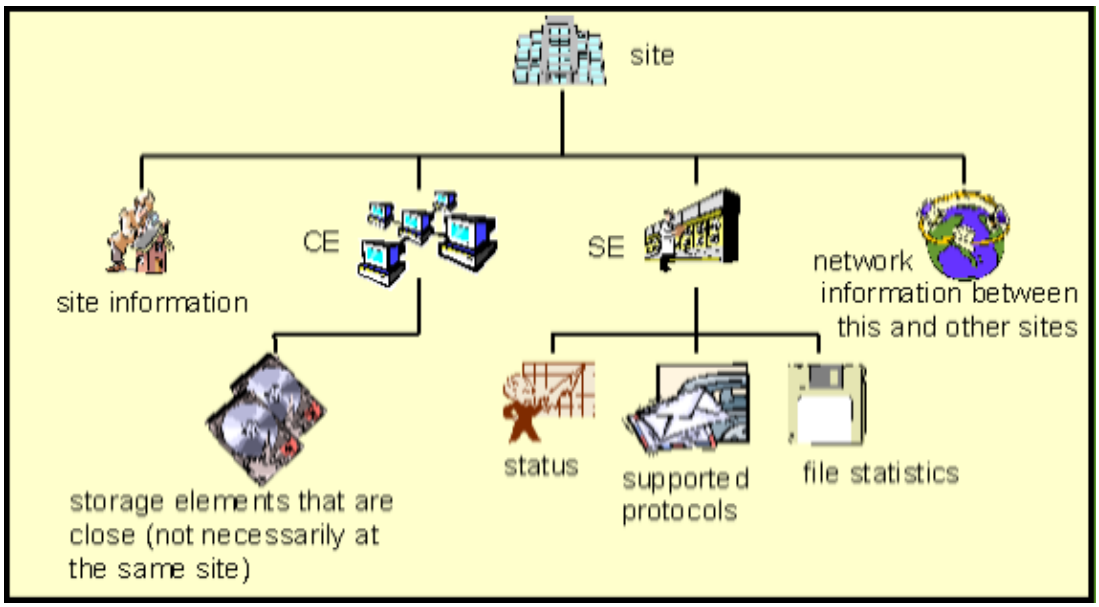

Figure 6: The Directory Information Tree (DIT)

Figure 6 shows an example Directory Information Tree (DIT) of a Grid element publishing information. In that example, the root entry identifies the site, and entries for the CEs and SEs, information on the site and the network are defined in the second level. The Distinguished Name of a particular CE entry would be here formed by an attribute identifying the site (like `site_ID=cern`) and an attribute identifying the CE (something like `CE_ID=lxn1102.cern.ch`), so the complete DN would be similar to `CE_ID=lxn1102.cern.ch,site_ID=cern`. Actual DITs published by LCG-2 elements are shown in Appendix C.

What kind of information can be stored in each entry of the DIT is specified in an ***LDAP schema***. The schema 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 one.

As stated earlier, the LDAP schema used for the Information Service implements the GLUE Schema.

In the future, other implementations for the GLUE Schema could be used in LCG.

4. GETTING STARTED

This section describes the preliminary steps to gain access to the LCG-2 Grid. Before using the LCG-2 Grid, the user must do the following:

1. Obtain a Cryptographic X.509 certificate from an LCG-2 approved ***Certification Authority (CA)***.
2. Get registered with LCG-2.
3. Join one of the LCG-2 Virtual Organizations (consequence of the registration process).
4. Obtain an account on a machine which has the LCG-2 User Interface software installed.
5. Create a proxy certificate.

Steps 1 to 4 need to be executed only once to have access to the Grid. Step 5 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. 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

The first requirement the user must fulfil 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/her certificate. It is up to the user to discover which CA he/she 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 and install certificates of a particular CA:

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

Usually, obtaining a certificate involves creating a request with the `grid-cert-request` command, which will generate the following 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.
`usercert.pem`	should be replaced by the actual certificate when sent by the CA. (This should be readable by everyone) (i.e. `chmod 444 usercert.pem`).

Then the `userreq.pem` file is sent (usually by e-mail using a particular format) to the desired CA, which will, after approval, return the new certificate also by mail.

An important property of a certificate is the ***subject***, a string containing information about the user. A typical example is:

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

To be used in the LCG-2 Grid, the certificate must be in `PEM` format. If the certificate is in PKCS12 format (extension `.p12`), then on a machine with the `openssl` package installed it can be converted to `PEM` (extension `.pem`) using the `pkcs12` command, in this way:

```
$ 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 path for the input PKCS12 format file.
`userkey.pem`	is the path to the output private key file.
`usercert.pem`	is the path to the output PEM certificate file.

The first command creates only the private key (due to the `-nocerts` option), and the second one creates the certificate (`-nokeys` option). The `-clcerts` option instructs that only client certificates, and not CA certificates, must be created.

The `grid-change-pass-phrase -file <private_key_file>` command changes the passphrase that protects the private key. This command will work even if the original key is not password protected. If the `-file` argument is not given, the default location of the file containing the private key is assumed.

4.2. REGISTERING WITH LCG-2

Before a user can use the LCG-2 service, registration of some personal data with the LCG registration server (hosted at CERN) plus some additional steps are required. For detailed information please visit the following URL:

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

To actually register oneself to the LCG-2 service, it is necessary to use a WWW browser with the user certificate installed for the request to be properly authenticated.

Browsers (including Internet Explorer and Mozilla) use a certificate format different than the one used by the LCG-2 Grid software. Browsers require a format called `PKCS12` whereas Grid software uses `PEM` 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 are available at:

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

4.3. VIRTUAL ORGANIZATIONS

A second requirement for the user is to belong to a ***Virtual Organization (VO)***. A VO is an entity, which corresponds typically to a particular organization 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.

Entering the VO of an experiment usually requires being a member of the collaboration; the user must comply with the rules of the VO relevant to him/her to gain membership. Of course, it is also possible to be expelled from a VO when the user fails to comply with these rules.

It is not possible to access the LCG-2 Grid without being member of any VO. Every user is required to select his/her VO when registering with LCG-2 and the supplied information is forwarded to the VO administration and resource prividers for validation before the registration process is completed. This forwarding is accomplished by the registration interface back-end automaticaly. It generates an email to the VO manager of the selected VO requesting addition of the user to the VO.

Currently, it is only possible to belong to one VO at a time. This is fine for most users. In the rare case that you need to belong to more than VO, then you should contact the LCG registration service (whose URL was given before).

A complete list of the VOs accepted by LCG-2 is available at the URL:

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

4.4. THE LCG GRID OPERATIONS CENTRE

Although still starting, the ***LCG Grid Operations Centre (GOC)*** is the central point of operational information for the LCG-2 Grid, such as configuration information and contact details. It is a very important source information for users of LCG2. The URL of the GOC website is the following:

https://goc.grid-support.ac.uk/gridsite/gocmain/

Among other informations, the GOC web page contains information of the status and nodes configuration of every one of the LCG2 sites in the GOC database. Its URL is the following:

https://goc.grid-support.ac.uk/gridsite/db/

To be able to access this database, the user must get registered first. This can be easily done completing a request form in:

https://goc.grid-support.ac.uk/gridsite/db-auth-request/

Note: It is necessary that the user has his/her digital certificate loaded in the web browser to be able to register with the GOC database and to access it.

The GOC also provides monitoring information for LCG-2, as described in Section 7.4, information on security in LCG-2, news...

4.5. SETTING UP THE USER ACCOUNT

To access the LCG-2 Grid, a user must also have an account on a LCG-2 User Interface. To obtain such an account, a local system administrator must be contacted. The official list of LCG sites is available at the GOC website.

As an alternative, the user can install the UI software on his/her machine (see the Installation and Administration Guide [R16]).

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.

4.6. CHECKING A CERTIFICATE

To verify that a certificate is not corrupted and print some information about it, the Globus command `grid-cert-info` can be used from the user's UI account. 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.

Example 4.6.1 (Printing 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
        Vali dity
            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)
                Modul us (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
```

Example 4.6.2 (Verifying a user certificate)

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

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

and if the certificate is valid, 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
```

4.7. 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.

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

Example 4.7.1 (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 (e.g. X509_USER_PROXY=$HOME/.globus/proxy), in which case a proxy with that file name will be created, if possible.

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 passphrase 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 4.7.2 (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 4.7.3 (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.
```

Known limitations: A person with administrator privileges on a machine can steal proxies and run jobs on the Grid.

4.7.1. Virtual Organization Membership Service

The ***Virtual Organization Membership Service (VOMS)*** is a new service that will be used to manage authorization information in VO scope. This service is not used in LCG-2 yet, but the reader may find references to it in some of the commands manpages, or in the literature, and therefore it is considered necessary to make a brief description of it in this manual.

The VOMS system will be used to include VO membership and any related authorization information in a user's proxy certificate. These proxies will be said to have *VOMS extensions*. The user will utilize the `edg-voms-proxy-init` command instead of the previously described `grid-proxy-init`, and a VOMS server will be contacted to check the user's certificate and create a proxy certificate with VOMS information included. By using that certificate, the VO of a user will be present in every action that he/she performs. Therefore, the user will not have to specify it using a `--vo` option.

NOTE: In the current release, and while VOMS is not used, a user can specify any VO using the `--vo` option when submitting a job (see Chapter 5), even if he/she does not belong to that VO, and the submission may be accepted. This does not mean, however, that the user credentials are not checked before the job is allowed to be run. The specified VO is used in this case for information and configuration purposes only, but the personal certificate of the user (through his/her proxy) is checked for the authorization, and the job is aborted if the user's real VO is not supported in the destination CE.

4.8. ADVANCED PROXY MANAGEMENT

The proxy certificates created as described in the previous section have an inconvenient: if the job does not finish before the proxy expires, it is aborted. This is clearly a problem if, for example, the user must submit a number of jobs that take a lot of time to finish: he should create a proxy certificate with a very long lifetime, fact that would increase the security risks.

To overcome this limit, a proxy credential repository system is used, which allows the user to create and store a long-term proxy certificate on a dedicated server (Proxy 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 an LCG-2 site has a Proxy Server, and what its hostname is, please check for nodes of type `PROX`, in the GOC database.

As the renewal process starts some time before the initial proxy expires, **it is necessary to generate an initial proxy long enough**, or the renewal may be triggered a bit 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 the `edg-job-*` commands will not even be accepted if the lifetime of the proxy credentials in the User Interface is lower than 10 minutes. An error message like the following will be produced:

```
**** Error: UI_PROXY_DURATION ****
Proxy certificate will expire within less then 00:10 hours.
```

The advanced proxy management offered by the UI of LCG-2 through the renewal feature is available via the `myproxy` command suite. The user must know the host name of a Proxy Server (often referred to as ***MyProxy server***).

For the WMS to know what Proxy 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 Chaper 5). If the user does not add it manually, then the name of the default Proxy Server is added automatically when the job is submitted. This default Proxy Server node is site and VO dependent and is usually defined in the UI VO's configuration file, stored at `$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf`.

Example 4.8.1 (Creating a long-term proxy and storing it in a Proxy Server)

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

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

where `-s <host_name>` specifies the hostname of the machine where a Proxy Server runs, the `-d` option instructs the server to use the subject of the certificate as the default username, and the `-n` option avoids the use of a passphrase to access to the long-term proxy, so that the WMS can perform the renewals 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 lxshare0207.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 <host_name>` option is missing, the command will try to use the `$MYPROXY_SERVER` environment variable to determine the Proxy Server.

ATTENTION! If the hostname of the Proxy 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 4.8.2 (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 <host_name> -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 4.8.3 (Deleting a long-term proxy)

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

```
$ myproxy-destroy -s <host_name> -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. WORKLOAD MANAGEMENT

In the LCG-2 Grid, a user can submit and cancel jobs, query their status, and retrieve their output. These tasks go under the name of ***Workload Management***. The LCG-2 offers two different User Interfaces to accomplish these tasks. One is the Command Line Interface and the other is the Graphical User Interface.

But, no matter what interface is used to submit a job, a description of its characteristics and requirements must be sent along with it, so that an appropriate destination for its execution can be found. The language used to describe a job is called ***Job Description Language (JDL)***, and it is discussed in the next section, before the interfaces are described.

5.1. JOB DESCRIPTION LANGUAGE

In LCG-2, job description files (`.jdl` files) are used to describe jobs for execution on Grid. These files are written using a Job Description Language (JDL). The JDL adopted within the LCG-2 Grid is the ***Classified Advertisement (ClassAd) language***[R17] defined by the ***Condor Project*** [R18], which deals with the management of distributed computing environments, and whose central construct is the ***ClassAd***, a record-like structure composed of a finite number of distinct attribute names mapped to expressions. A ClassAd is a highly flexible and extensible data model that can be used to represent arbitrary services and constraints on their allocation. The JDL is used in LCG-2 to specify the desired job characteristics and constraints, which are used by the match-making process to select the resources that the job will use.

The fundamentals of the JDL are given in this section. A detailed description of the JDL syntax is out of the scope of this guide, and can be found in [R19] and [R20].

The JDL syntax consists on staments ended by a semicolon, like:

```
attribute = value;
```

Literal strings (for values) are enclosed in double quotes. If a string itself contains double quotes, they must be escaped with a backslash (e.g.: `Arguments = " \"hello\" 10"`). For special characters, such as &, the shell on the WN will itself expect the escaped form: \&, and therefore both the slash and the ampersand will have to be escaped inside the JDL file, resulting in: \\\&. In general, special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \. The character " ‘ " cannot be specified in the JDL.

Comments must be preceded by a sharp character (#) or have to follow the C++ syntax, i.e a double slash (//) at the beginning of each line or statements begun/ended respectively with /* 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.

In a job description file, some attributes are mandatory, while some others are optional. Essentially,

one must at least specify the name of the executable, the files where to write the standard output and the standard error of the job (they can even be the same file). For example:

```
Executable = "test.sh";
StdOutput = "std.out";
StdError = "std.err";
```

If needed, arguments to the executable can be passed:

```
Arguments = "hello 10";
```

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

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

Then, the files to be transferred between the UI and the WN before (Input Sandbox) and after (Output Sandbox) the job execution can be specified:

```
InputSandbox = {"test.sh","std.in"};
OutputSandbox = {"std.out","std.err"};
```

In this example, the executable `test.sh` is also transferred. This would not be necessary if that file was already in the Worker Node (or, for example, it was a common Unix command, such as `/bin/hostname`, which was used in a previous example).

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. Neither the `InputSandbox` nor the `OutputSandbox` lists can contain two files with the same name (even if in different paths) as when transferred they would overwrite each other.

Note: The executable flag is not preserved for the files included in the Input Sandbox when transferred to the WN. Therefore, for any file needing execution permissions a `chmod +x` operation should be performed by the initial script specified as the `Executable` in the JDL file (the `chmod +x` operation is done automatically for this script).

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

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

If the job requires some files stored in an LCG Storage Element, the `InputData` attribute can be used to make the resource broker select a CE as close as possible to the files. The `OutputSE` attribute, similarly, specifies the SE where the user wants to store the generated output data. This is used by the RB to find a CE that is *close* to the given SE. Finally, the `OutputData` attribute can be used to automatically have any output data files copied and registered in the Grid.

Example 5.1.1 (Specifying input data in a job)

If the user job needs to read two files (identified by a logical file name or by their GUID), the job description file may contain a line like the following:

```
InputData = {"lfn:doe/prod/kin_1", "guid:136b48a64-4a3d-87ud-3bk5-8gnn46m49f3"};
```
[5]

In addition, if the `InputData` attribute is used, the protocols the application is able to use to read the files must be declared.

```
DataAccessProtocol = {"rfio", "gsiftp"};
```

The only supported protocols are `gsiftp` and `rfio`. The meaning of these two protocols is the following:

- **gsiftp**: for GSIFTP, the GSI version of FTP. The application will access the file via GSIFTP, for instance copying the file first on a locally accessible storage.
- **rfio**: for RFIO, a CERN's remote file access protocol [R11], which allows the user to read and write files remotely, but only within a local area network (not between sites), as it is not included in the GSI yet. It is just a read-write remote protocol.

The inclusion of these attributes will cause the Resource Broker to look for replicas of the specified files, in order to find a CE which can access them in a *close* SE. The Resource Broker will match the CE that is closer to the SE holding the greater number of the requested files. If, for example two files located in *SE A* and one located in *SE B* are requested, then a CE that has *SE A* defined as close will be chosen. If the number of files is the same in *SE A* and *SE B*, then both the CEs close to *SE A* and the ones close to *SE B* will match.

However, if the user knows that there are replicas of the required files in a distant SE, he/she can copy them manually to a close SE beforehand, so the submission works. Moreover, the user can also leave these attributes out of the JDL file, and still access the files from the job, using GSIFTP (RFIO will not work if files are located in different local area networks). This is, though, against the philosophy of the Grid, since a CE should not access distant files, increasing the network traffic, but rather use closer copies.

Detailed information of how the job can access the Grid files is given in Chapter 6.

The job will be sent to the CE with the best *rank* (which is a user-definable measurement of the CE *goodness*), between all the CEs satisfying all the job requirements and having the maximum number of file replicas on a SE close to them.

[5] For details on file names conventions refer to 3.2.5.

Example 5.1.2 (Specifying a Storage Element)

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 = "lxshare0291.cern.ch";
```

The Resource Broker will not abort the job if there is no CE close to the `OutputSE` specified by the user. The RB will try to find resources close to such SE, but if the CE cannot be found the job will run somewhere else.

Example 5.1.3 (Automatic upload and registration of output files)

The `OutputData` attribute list allows the user for the automatic upload and registration in LCG-2 of files produced by the job on the WN. Several output files can be specified. For each of these files, three attributes can be set.

The `OutputFile` attribute of `OutputData` is mandatory and specifies the name of the generated file to be uploaded to the Grid. The `StorageElement` is an optional string indicating the SE where the file should be stored, if possible. If unspecified, the WMS automatically choses a SE close to the CE. Finally, the `LogicalFileName` attribute (also optional) represents a LFN the user wants to be associated to the output file in LCG-2.

The following code shows an example `OutputData` attribute:

```
OutputData = {
[
 OutputFile="my_file_1.out";
 LogicalFileName="lfn:my_test_result"
 StorageElement="lxshare0291.cern.ch"
],
[
 OutputFile="my_file_2.out"
 LogicalFileName="lfn:my_debugging"
]
};
```

To express any kind of requirement on the resources where the job can run, there is the `Requirements` attribute. 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. For a list of GLUE attributes, see Appendix C.

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.

Example 5.1.4 (Specifying requirements on the CE)

Let us suppose that the user wants to run on a CE using PBS as the LRMS, and whose WNs have at least two CPUs. He/she will write then in the job description file:

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

where the `other.` prefix is used to indicate that the `GlueCEInfoLRMSType` attribute refers to the CE characteristics and not to those of the job. If `other.` is not specified, then the default `self.` is assumed, indicating that the attribute refers to the job characteristics description.

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";
```

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.

As a general rule, requirements on attributes of a CE are written prefixing `"other."` to the attribute name in the Information System schema.

Example 5.1.5 (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 this jobs to run on CEs 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 5.1.6 (Specifying requirements on a close SE)

The previous requirements affected always two entities: the job and the CE. In order to specify requirements involving three entities (i.e., the job, the CE and a SE), the RB uses a special match-making mechanism, called ***gangmatching***. This is supported by some JDL functions: `anyMatch`, `whichMatch`, `allMatch`. A typical example of this functionality follows. For more information on the gangmatching, please refer to [R20].

To ensure that the job runs on a CE with, for example, at least 200 MB of free disk space on a close SE, the following JDL expression can be used[6]:

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

Example 5.1.7 (A complex requirement used in LCG-2)

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 befores it has time to finish).

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

The `VirtualOrganisation` attribute represents another way to specify the VO of the user, as for example in:

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

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

This value is anyway superseded by the `--vo` option of `edg-job-submit`.

The JDL attribute called `RetryCount` can be used to specify how many times the WMS must try to resubmit a job if it fails due to some LCG component; that is, not the job itself (though it is sometimes difficult to tell where the failure of the job was originated). The default value (if any) is defined in the file `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`.

The `MyProxyServer` attribute indicates the Proxy Server containing the user's long-term proxy that the WMS must use to renew the proxy certificate when it is about to expire. If this attribute is not included manually by the user, then it is automatically added when the job is submitted. Its value, in this case, is the name of the UI's default Proxy Server for the user's VO.

[6]The function used to calculate the available space in a SE can be inaccurate if the SE uses NFS mounted filesystems. Also, the measurement is not useful for SE using MSS (such as tape systems), as the available space returned is infinite (or 1000000000000), since new tapes can always be added.

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.

The user can define the rank with the `Rank` attribute as a function of the CE attributes. The default definition takes into account the number of CPUs in the CPU that are free:

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

But other definitions are possible. The next one is a more complex expression:

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

In this case, the number of waiting jobs in a CE is used if this number is not null. The minus sign is used so that the rank decreases as the number of waiting jobs gets higher. If there are not waiting jobs, then the numberof free CPUs is used.

5.2. THE COMMAND LINE INTERFACE

In this section, all commands available for the user to manage jobs are described. For a more detailed information on all these topics, and on the different commands, please refer to [R15].

5.2.1. Job Submission

To submit a job to the LCG-2 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>
```

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

Example 5.2.1.1 (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
```

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

```
======================== edg-job-submit Success ========================
 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://lxshare0234.cern.ch:9000/rIBubkFFKhnSQ6CjiLUY8Q
========================================================================
```

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 cancelling 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 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[7], he/she can specify his virtual organization with the `--vo <vo_name>` option; otherwise the default VO specified in the standard configuration file (`$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`) 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`, the command will return the following error:

[7] and currently this must be the case.

```
**** Error: UI_NO_VO_CONF_INFO ****
Unable to find configuration information for VO "unspecified"

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

where the absence of VOMS extensions in the user's proxy is also shown.

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 BrokerInfo functionality (see Section 5.2.7) 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>
```

where `<full_hostname>` and `<port>` are the hostname of the machine and the port where the Globus Gatekeeper 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
```

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.

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 `edg-job-submit` command inside scripts as an alternative to the `-o` option.

Example 5.2.1.2 (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 file using the command `edg-job-list-match`:

```
$ edg-job-list-match test.jdl
```

```
Connecting to host  lxshare0380.cern.ch, port 7772
Selected Virtual Organisation name (from UI conf file): dteam

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

                  *CEId*
adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite
adc0015.cern.ch:2119/jobmanager-lcgpbs-long
adc0015.cern.ch:2119/jobmanager-lcgpbs-short
**********************************************************************
```

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 `edg-job-submit`.

5.2.2. 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 5.2.2.1 (Retrieving the status of a job)

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

```
$ edg-job-status <jobId>
```

And an example of a possible output is

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

Printing status info for the Job:
https://lxshare0234.cern.ch:9000/X-ehTxfdlXxSoIdVLS0L0w

Current Status:    Ready
Status Reason:     unavailable
Destination:       lxshare0277.cern.ch:2119/jobmanager-pbs-infinite
reached on:        Fri Aug  1 12:21:35 2003
*************************************************************
```

where the current status of the job is showed, 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 summarised in Appendix E. Finally, the `destination` field contains the ID of the CE where the job has been submited.

Much more information is provided if the verbosity level is increased by using `-v1` or `-v2` with the command. See [R15] for detailed information on each of the fields that are returned then.

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

```
edg-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 `edg-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).

```
$ edg-job-status -i jobs.list
------------------------------------------------------------
1 : https://lxshare0234.cern.ch:9000/UPBqN2s2ycxt1TnuU3kzEw
2 : https://lxshare0234.cern.ch:9000/8S6IwPW33AhyxhkSv8Nt9A
3 : https://lxshare0234.cern.ch:9000/E9R0Yl4J7qgsq7FYTnhmsA
4 : https://lxshare0234.cern.ch:9000/Tt80pBn17AFPJyUSN9Qb7Q
a : all
q : quit
------------------------------------------------------------

Choose one or more edg_jobId(s) in the list - [1-4]all:
```

If the `--all` option is used insted, 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.

```
$ edg-job-status --all -s DONE -s RUNNING
$ edg-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 [R15].

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

Example 5.2.2.2 (Cancelling a job)

A job can be cancelled before it ends using the command `edg-job-cancel`. This command requires as arguments one or more job identifiers. For example:

```
$ edg-job-cancel https://lxshare0234.cern.ch:9000/dAE162is6EStca0VqhVkog \
https://lxshare0234.cern.ch:9000/C6n5Hq1ex9-wF2t05qe8mA

Are you sure you want to remove specified job(s)? [y/n]n :y
==========================  edg-job-cancel Success============================
The cancellation request has been successfully submitted for the following job(s)
 - https://lxshare0234.cern.ch:9000/dAE162is6EStca0VqhVkog
 - https://lxshare0234.cern.ch:9000/C6n5Hq1ex9-wF2t05qe8mA
================================================================================
```

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

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

Example 5.2.2.3 (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 `edg-job-get-output`, which takes a list of jobs as argument. For example:

```
$ edg-job-get-output https://lxshare0234.cern.ch:9000/snPegp1YMJcnS22yF5pFlg

Retrieving files from host lxshare0234.cern.ch

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

Output sandbox files for the job:
- https://lxshare0234.cern.ch:9000/snPegp1YMJcnS22yF5pFlg
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/snPegp1YMJcnS22yF5pFlg

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

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 `edg-job-status`.

Example 5.2.2.4 (Retrieving logging information about submitted jobs)

The `edg-job-get-logging-info` command queries the LB persistent database for logging information about jobs previously submitted using `edg-job-submit`. 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 consider:

```
$ edg-job-get-logging-info -v 0 -o logfile.txt \
https://lxshare0310.cern.ch:9000/C_CBUJKqc6Zqd4clQaCUTQ

==============  edg-job-get-logging-info Success =================
 Logging Information has been found and stored in the file:
 /afs/cern.ch/user/d/delgadop/pruebas/logfile.txt
===================================================================
```

where the `-v` option sets the detail level of information about the job displayed to the user (possible values are 0,1 and 2).

The output (stored in the file `logfile.txt`) will be:

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

Printing info for the Job: https://lxshare0310.cern.ch:9000/C_CBUJKqc6Zqd4clQaCUTQ

        - - -
 Event: RegJob
- source               =    UserInterface
- timestamp            =    Fri Feb 20 10:30:16 2004
        - - -
 Event: Transfer
- destination          =    NetworkServer
- result               =    START
- source               =    UserInterface
- timestamp            =    Fri Feb 20 10:30:16 2004
        - - -
 Event: Transfer
- destination          =    NetworkServer
- result               =    OK
- source               =    UserInterface
- timestamp            =    Fri Feb 20 10:30:19 2004
        - - -
 Event: Accepted
- source               =    NetworkServer
- timestamp            =    Fri Feb 20 10:29:17 2004
        - - -
```

```
 Event: EnQueued
- result               =    OK
- source               =    NetworkServer
- timestamp            =    Fri Feb 20 10:29:18 2004
[...]
```

5.2.3. Interactive Jobs

NOTE: Interactive jobs are not yet supported in LCG, and that functionality is not part of the official distribution of the current LCG-2 relase, although it may be in the future. Any site installing or using it will do it only under its own responsability.

This section gives an overview of how interactive jobs should work in LCG-2.

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 that 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 she/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 5.2.3.1 (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 7 shows the result of the program (after the user has entered his name) in the generated X window.

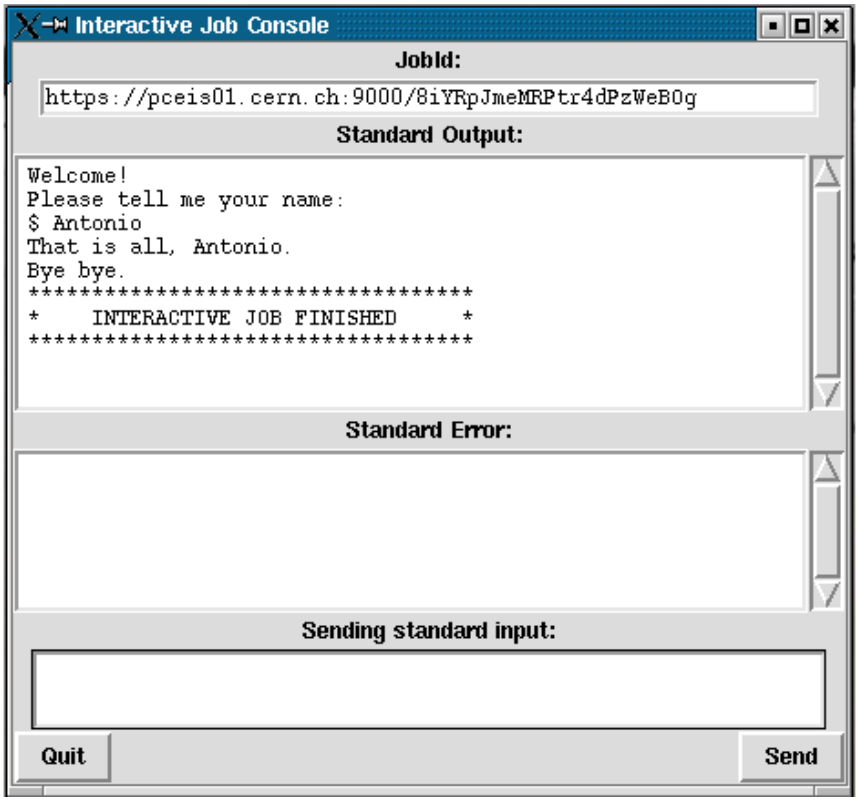

Figure 7: X window for an interactive job

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 her/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 5.2.3.2 (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 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, 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 ubmission of the ame `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.

5.2.4. Checkpointable Jobs

NOTE: Checkpointable jobs are not yet supported in LCG, and that functionality is not part of the official distribution of the current LCG-2 relase. Any site installing or using it will do it only under its own responsability.

This section gives a brief overview of how checkpointable jobs should work in LCG-2.

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.

5.2.5. MPI Jobs

NOTE: MPI software has not been tested yet, and it is not part of the official distribution of the current LCG-2 relase. Any site installing or using it will do it only under its own responsability.

This section gives a brief overview of how MPI jobs should work in LCG-2.

Message Passing Interface (MPI) applications are run in parallel on several processors. Jobs that must be run as MPI are specified setting the JDL `JobType` attribute to `MPICH`. When a MPI job is submitted, the presence of the `NodeNumber` attribute (it specifies the required number of CPUs) in the JDL is mandatory and 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 adding the following expression:

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

to the the JDL requirements expression.

5.2.6. Advanced Command Options

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

The main configuration file is located by default at `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`, and sets, among other things, the default VO, the default location for job outputs and command log files and the default values of mandatory JDL attributes. It is possible to point to a different configuration file by setting the value of the environment variable `$EDG_WL_UI_CONFIG_VAR` to the file path, or by specifying the file in the `--config <file>` option of the `edg-job-*` commands (which takes precedence).

In addition, VO-specific configurations are defined by default in the file `$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf`, consisting essentially in the list of Network Servers, Proxy Servers and LB servers accessible to that VO. A different file can be specified using the variable `$EDG_WL_UI_CONFIG_VO` or the `--config-vo <file>` option of the `edg-job-*` commands.

Example 5.2.6.1 (Changing the default VO)

A user can change his/her default VO by performing the following steps:

a. Make a copy of the file `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`, for example to `$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 EDG_WL_UI_CONFIG_VAR $HOME/my_ui.conf      ((t)csh)
export EDG_WL_UI_CONFIG_VAR=$HOME/my_ui.conf      (bash)
```

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.

5.2.7. 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.

Detailed information about the BrokerInfo file, the `edg-brokerinfo` CLI, and its respective API can be found in [R21].

The `edg-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 `VirtualOrganization` JDL attribute;
- `getAccessCost`: not supported at present.

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 linebreak `$EDG_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 `$EDG_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:

```
/opt/edg/bin/edg-brokerinfo getCE
```

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

If, on the contrary, `edg-brokerinfo` is invoked from the UI, the `$EDG_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 ouput of the submitted job, and retrieve it with the rest of generated output, when the job finishes. This can be done by specifying the file in the Output Sandbox of the job or by including the following lines:

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

in a submitted shell script.

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

5.3. THE GRAPHICAL USER INTERFACE

The EDG WMS GUI is a Java Graphical User Interface composed of three different applications: the JDL Editor, the Job Monitor and the Job Submitter. The 3 GUI components are integrated although they can be used as standalone applications so that the JDL Editor and the Job Monitor can be invoked from the Job Submitter, thus providing a comprehensive tool covering all main aspects of workload Management in a Grid environment: from creation of job descriptions to job submission, monitoring and control up to output retrieval.

Details on the EDG WMS GUI are not given in this guide. Please refer to [R22] for a complete description of the functionalities provided by the GUI, together with some example screenshots.

6. DATA MANAGEMENT

6.1. INTRODUCTION

6.1.1. Data Management Client Tools

In this chapter, the Data Management tools are described. These are high level tools used to upload files to the Grid, replicate data and locate the best replica available. Some use cases and example usage for these tools are listed. Besides, some lower level tools (like `edg-gridftp-*` commands) are introduced. These low level tools should only be used in case of problems and anyway by system administrators only and not by LCG-2 Grid users. As a reference only, a brief summary on their functions will be given.

The Data Management client tools are:

`lcg-* commands`	General file and replica management
`edg-replica-manager` (`edg-rm`)	Older file and replica management
`edg-local-replica-catalog` (`edg-lrc`)	LRC catalog manipulation
`edg-replica-metadata-catalog` (`edg-rmc`)	RMC catalog manipulation

The more important tools are the LCG Data Management tools (or `lcg-*` commands), which will be described in detail later. They provide the basic functionality that a LCG-2 user needs.

But in a UI, the `edg-*` tools can also be found. As shown in the table above, this command accepts both the long and the abbreviated form. These commands were created for the European DataGrid. The `edg-replica-manager` tool offers the same functionality as the `lcg-*` commands, but its use is advised against, since it is much slower. Besides, it is an older tool and new functionalities and improvements will be only added to the `lcg-*` commands. The description of the `edg-rm` tool can be found in appendix B. The EDG catalog manipulation tools are still in use, although they are low level tools and should only be used with great care.

For more information on how to use the client tools mentioned above, please refer to the command manpages, and to [R23] and [R24].

Apart from those presented above, there are two more commands in the UI: the `edg-replica-location-index` (`edg-rli`) and the `edg-replica-optmization` (`edg-ros`). These two commands will allow the user to interact with the Replica Location Index and the Replica Optimization services, when they are in operation. These services are planned by the LCG architecture, but they are not in use yet (and their commands are therefore not useful at the moment). Information about these two commands can be found in [R25] and [R26].

6.1.2. File Names within LCG-2

As a reminder of what was explained in Chapter 3, the different types of names that can be used within the LCG-2 files catalogues are summarized as follows:

A GUID, which identifies a file uniquely, is of the form:

```
guid:<40_bytes_unique_string>
```

like:

```
guid:38ed3f60-c402-11d7-a6b0-f53ee5a37e1d
```

An LFN or User Alias, which can be used to refer to a file in the place of the GUID, has this format:

```
lfn:<anything_you_want>
```

like:

```
lfn:importantResults/Test1240.dat
```

A SURL, which identifies a replica in a SE, is of the form:

```
sfn://<SE_hostname><SE_Accesspoint><VO_path><filename>
```

like:

```
sfn://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/
file3596e86f-c402-11d7-a6b0-f53ee5a37e1d
```

Finally, a TURL, which is a valid URI with the necessary information to access a file in a SE, has the following form:

```
<protocol>://<SE_hostname><SE_Accesspoint><VO_path><filename>
```

like:

```
gsiftp://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/
file3596e86f-c402-11d7-a6b0-f53ee5a37e1d
```

Also, remember that while a SURL is stored in the LRC, the TURL is either obtained by the information provided in the Information Service (in the case of a classical SE) or by the SRM (when these are used).

Failure to comply with these syntax rules results in corrupted catalogues and malfunctioning replica management.

6.2. File and Replica Management Client Tools

The LCG Data Management tools allow allow users to copy files between UI, CE, WN and a SE, to register entries in the RLS and replicate files between SEs. To perform those actions, several commands can be used. The name and functionality overview of them is shown in the following table.

`lcg-aa`	Adds an alias in RMC for a given GUID.
`lcg-cp`	Copies a Grid file to a local destination.
`lcg-cr`	Copies a file to a SE and registers the file in the LRC.
`lcg-del`	Deletes one file (either one replica or all replicas).
`lcg-gt`	Gets the TURL for a given SURL and transfer protocol.
`lcg-infosites`	Gives information about resources on the Grid.
`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.
`lcg-ra`	Removes an alias in RMC for a given GUID.
`lcg-rep`	Copies a file from one SE to another SE and registers it in the LRC.
`lcg-rf`	Registers in the LRC (and optionally in the RMC) a file residing on an SE.
`lcg-uf`	Unregisters in the LRC a file residing on an SE.

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 in all the commands, except for `lcg-gt`. Furthermore, in the commands where the option is present, it is mandatory —without it, the command will not work.

The `--config <file>` option is also present in most of the commands, to allow the specification of a configuration file, but is currently ignored. The situation is the same fo the `-i` option, which would indicate the use of an insecure connection to the replica catalog. This option is currently ignored, since the access to the catalog is done according to the endpoints published in Information Service.

It is important to note that for all these commands to work, the environment variable `LCG_GFAL_INFOSYS` must point to the IS provider (the BDII), so that the commands can retrieve the necessary information for their operation.

In what refers to access control, physical replicas of a file are protected by the use of the grid-mapfile and the local permissions that a user has on a SE. For this reason, `lcg-cr`, `lcg-cp`, `lcg-rep` and `lcg-del` need that the user has a valid proxy certificate to operate. Otherwise, an error is shown.

However, **the information in the LRC and RMC catalogs is not protected** in the same way, and no proxy certificate is required for the operation of the rest of `lcg-*` commands, which do not deal with physical replicas. Although this situation will be improved in the future, currently the information stored in the file catalogs **can be altered by anyone**, and this could lead to the loss of files, not only for the user but for other users as well (if their files catalog references are deleted). Be careful when dealing with the information in the catalogs.

In what follows, some usage examples are given. For details on the options of each command, please use the manpages of the commands.

For clarity reasons, in the pieces of code that follow (throghout the whole chapter), the commands introduced by the user are leaded by a '$' symbol, and the answers of the shell are usually preceded by '>' (unless the difference is obvious).

6.2.1. Basic Replica Management Commands

Example 6.2.1.1 (Uploading a file from the UI to the Grid)

In order to upload a file to the Grid; i.e.: to transfer it from the local machine to a Storage Element where it must reside permanently, the `lcg-cr` command (which stands for copy and register) can be used (in a machine with a valid proxy). An example of usage follows:

```
$ 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 required `-d <destination>` option indicates the SE (which must be known in advance) that will be used as the destination for the file. The command returns the unique GUID.

If, for the `-d` option, the user specifies only the SE hostname, the file will be stored with a generated filename under the user's VO directory. The path to that directory is expressed as the ***accesspoint***, within which all the SE's Grid files are stored, plus path to the particular VO directory. These both informations are published by the IS. If the `-P <relative_path>` option is used, then the specified path, relative to the user's VO directory, and filename will be used.

There is also the possibility to specify the destination as a complete SURL, incluing 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
```

where the SE's accesspoint is `/flatfiles/SE00` and the directory for the VO dteam is called `dteam`. As seen, if the last option (with `-P`) is used, there is no necessity to know the accesspoint or VO directoy name of the SE.

In the previous examples, no LFN is associated with the file, and in that case, the returned GUID will be the only way to access the file in the Grid. In order to associate an alias to the file, the `-l <lfn>` option must be included. This is illustrated by the following command:

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch -l lfn:my_alias1 file:/home/antonio/file1
> guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
```

As seen in the previous examples, when a file is registered in LCG-2, a GUID for it is automatically generated. This should be the usual case for files uploaded to the Grid. Nevertheless, for some special cases, the user may want to specify the GUID he/she wants for the file. This can be accomplished by using the `-g <guid>` option. The specified GUID must be a well-formed GUID. An example of this follows:

```
$ 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. Because the specification of an existing GUID is also allowed, a misuse of the tool may end up in a corrupted file in which replicas sharing GUID 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 8).

Example 6.2.1.2 (Retrieving information about the Grid)

As seen, 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. Either the Information Service is queried directly (as explained in Chapter 7), or the `lcg-infosites` command is used:

```
$ lcg-infosites --vo dteam all nousedspace
```

The previous command returns all CEs and SEs that are published in the IS for the specified VO; as well as the RMC and LRC endpoints for that VO. For each CE, information about the number of CPUs and the running and queued jobs is given, and for each SE, the total and available space figures are provided.

A typical output is as follows:

```
LRC endpoint for dteam:  http://rlsdteam.cern.ch:7777/dteam/v2.2/edg-local-replica-catalog
/services/edg-local-replica-catalog
RMC endpoint for dteam:  http://rlsdteam.cern.ch:7777/dteam/v2.2/edg-replica-metadata-cata
log/services/edg-replica-metadata-catalog

************************************************************
These are the related data for dteam: (in terms of CPUs)
************************************************************

#CPU    Free    Total Jobs      Running Waiting ComputingElement
```

```
-------------------------------------------------------------
   6       6       0              0         0    ce01.lip.pt:2119/jobmanager-lcgpbs-dteam
   9       9       0              0         0    lcg-ce.ecm.ub.es:2119/jobmanager-pbs-long
   2       0       0              0         0    lcg03.gsi.de:2119/jobmanager-lcgpbs-long
   5       1       0              0         0    ce00.inta.es:2119/jobmanager-lcgpbs-long
   9       9       0              0         0    lcg-ce.ecm.ub.es:2119/jobmanager-pbs-short
   2       0       2              2         0    lcg03.gsi.de:2119/jobmanager-lcgpbs-short
   7       6       0              0         0    lxt03.jinr.ru:2119/jobmanager-pbs-long
  44      44       0              0         0    ce.prd.hp.com:2119/jobmanager-lcgpbs-long
   5       1       4              4         0    ce00.inta.es:2119/jobmanager-lcgpbs-short
   7       6       0              0         0    lxt03.jinr.ru:2119/jobmanager-pbs-short
  57      57       0              0         0    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-A
  57      57       0              0         0    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-G
  93      83      13             13         0    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-T

[...]

The total values are:
-------------------------------------------------------------
15780   6177    4826    1415    3411

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

Avail Space(Kb)         SEs
-------------------------------------------------------------
72982236        se01.lip.pt
7549980         se00.inta.es
31310044        se.prd.hp.com
560491852       teras.sara.nl
10176836        lcg-se.ecm.ub.es
1000000000000   lcgse02.ifae.es
1000000000000   lcgse03.ifae.es
1899648224      dgse0.icepp.jp
1588648960      gridkap02.fzk.de
68361184        lxn1183.cern.ch
34200680        epcf37.ph.bham.ac.uk
1134532256      grid004.ft.uam.es

[...]
```

In the above example information for the `dteam` VO was printed. Since the `all` argument was given, the information relative to the CEs and SEs for that VO was provided. For the CEs, the printed data is the number of CPUs and how many of them are free, as well as the number of running and waiting jobs. For the SEs, the available space in each one of them is given. In addition, the LRC and RMC endpoints are printed at the beginning.

The `nousedspace` option was added so that the command did not return the space used in each SE. Currently, this functionality takes very long time, so its use should be avoided. Thus, the `nousedspace` option should be usually included when the `all` or `se` arguments are used.

If, instead `all`, other arguments are used, the returned information is a subset of that seen before. The `ce` argument makes the command show only the information relative to CEs, `se` makes it show only the information of SEs, and `lrc` and `rmc` show the LRC and RMC endpoints respectively.

There is one more possible argument, which gives information not provided by `all`. When using the `closeSE` option, the `lcg-infosites` shows a list of close SEs for each of the CEs. This information can be useful for jobs that must access a file stored in an SE. The output produced by this command is the following:

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

ce01.lip.pt:2119/jobmanager-lcgpbs-dteam
        se01.lip.pt

lcg-ce.ecm.ub.es:2119/jobmanager-pbs-long
        lcg-se.ecm.ub.es

lcg03.gsi.de:2119/jobmanager-lcgpbs-long

cclcgceli01.in2p3.fr:2119/jobmanager-bqs-A
        cclcgseli01.in2p3.fr
        cclcgseli02.in2p3.fr

lcg-ce.ecm.ub.es:2119/jobmanager-pbs-short
        lcg-se.ecm.ub.es

[...]
```

where for every CE listed, a list of close SEs are introduced in different lines and preceded by a tabstop.

Finally, there is one more option, `--is`, which allows the user to specify the IS provider (the BDII) to be used when retrieving the information. If not used, the default is taken from the UI environmental variable `$LCG_GFAL_INFOSYS`. An usage example of this option would be:

```
lcg-infosites --vo dteam closeSE  --is lxn1178.cern.ch
```

which produces the same output seen previously.

IMPORTANT: The order of options and arguments for `lcg-infosites` must be followed. Otherwise, the command will not work properly. The proper order is the following:

```
lcg-infosites --vo <vo_name> <option> [nousedspace] [--is <BDII_host>]
```

Example 6.2.1.3 (Replicating a file)

Once a file is stored on an SE and registered within the Replica Location Service, the file can be replicated using the `lcg-rep` command, as in:

```
$ lcg-rep --vo dteam -d lxb0707.cern.ch guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
```

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`).

If the `--verbose` (or `-v`) option is used, some information on what is happening is presented:

```
$ 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
```

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 succesfully, but no new replica will be created.

Example 6.2.1.4 (Listing replicas, GUIDs and aliases)

The lcg-lr command allows users to list all the replicas of a file that have been successfully registered within the Replica Location Service.

```
$ 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.

Reciprocally, 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
```

And `lcg-la` (list aliases) can be used to list the LFNs associated with a particular file, which can be, as usual, identified by its GUID, any of its LFNs, or the SURL of one of its replicas. An example on this follows:

```
$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
> lfn:my_alias1
```

The tools `edg-local-replica-catalog` and `edg-replica-metadata-catalog`, described later, offer more functions for catalog interaction, although the ones provided by the `lcg-*` commands should be enough for a normal user.

Example 6.2.1.5 (Copying files out of the Grid)

The `lcg-cp` command can be used to copy a Grid file to a non-grid storage resource. This is useful to have a local copy of the file. The command accepts the LFN, GUID or one SURL of the LCG-2 file as its first argument and a local filename or valid TURL as the second, as it is shown in the following example:

```
$ lcg-cp --vo dteam lfn:my_alias1 file:/home/antonio/copy_of_file1
```

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 6.2.1.6 (Obtaining a TURL for a replica)

For any given replica (identified by its SURL) the TURL for accessing it using a particular protocol can be obtained with the `lcg-gt` command. The arguments are the SURL of the file and the protocol desired for the TURL.

```
$ lcg-gt sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-221
4-4db8-9ee8-2930de1a6bef gsiftp
> gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4d
b8-9ee8-2930de1a6bef
> 0
> 0
```

As the reader can see, the command output is composed of three lines. The first is the file's TURL, whilst the last two give information regarding SRM states, which can be, currently, safely ignored.

If the specified protocol is not supported by that SE for the given replica, an error message is returned:

```
$ lcg-gt sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-221
4-4db8-9ee8-2930de1a6bef ftp
>lcg_gt: Protocol not supported
```

Example 6.2.1.7 (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 the deleted replica was the last or only valid replica, the entries corresponding to its GUID are also removed from the RMC (including aliases).

The following series of commands show how to delete one particular replica of a file, and also all the available replicas. The `lcg-lr` command is used to show that the replicas are deleted:

```
$ 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 (when deleting all the replicas of the file).

6.2.2. Other Commands

Example 6.2.2.1 (Registering and unregistering Grid files)

Usually, new files are introduced in LCG-2 copying them from a non-grid resource using `lcg-cr`; they are replicated to different SEs using `lcg-rep`; and can be copied out of the Grid with `lcg-cp`. But it is also possible that a file is copied between SEs using `lcg-cp` (i.e., without registering) or by physically carrying a great amount of data in tapes, or it is possible that a new storage resource that already holds files is added to the Grid (becoming an SE). These files will be in an SE (there will be a valid SURL associated to them), but will not be registered in the LCG2 catalogs (i.e., they will not have an associated GUID).

For this situation, the `lcg-rf` command may be useful. The command associates a GUID for a given SURL. If the `-g <GUID>` option is used, then the file is associated with the specified GUID. In this case, it is assumed that there exist some other replicas of the files, which are already registered. Otherwise, a new GUID is generated for the file.

An example of the commands usage follows:

```
$ lcg-rf --vo dteam  sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file
0e85c3bf-d2f6-4b04-8db9-c848b4c2d7df
> guid:c06a92ee-6911-11d8-a453-d9c1af867039

$ 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, instead of using the `lcg-del`, which both unregisters and physically deletes a replica, a user can unregister a replica from the LRC catalogue, without actually deleting it (it can still be accessed on the SE with `lcg-cp`, for instance). This can be achieved with the `lcg-uf` command, specifying both the GUID and the SURL to be unregistered, as in:

```
$ 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, then the GUID is also removed from the catalogue along with all the associated information in the RMC.

Example 6.2.2.2 (Managing aliases)

The `lcg-aa` (add alias) command allows the user to add a new LFN to an existing GUID, as it is illustrated in the following example:

```
$ 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
```

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.

6.3. EDG-LRC AND EDG-RMC COMMANDS

The `edg-local-replica-catalog` and `edg-replica-metadata-catalog` commands are low level tools that allow users to browse and directly manipulate the LRC and the RMC catalogues.

Attention! With these tools, a user can change the content of the catalogues making them inconsistent. For instance, a GUID can be removed from the RMC but not from the LRC making a file not addressable by its alias. In normal operation, a user should preferably use the LCG Data Management tools, which provide most of the functionality, and only use the `edg-lrc` and `edg-rmc` with extreme care.

The general form of a `edg-lrc` and `edg-rmc` invocation is the following:

```
$ <edg-lrc | edg-rmc> <general_options> <cmd_name> <cmd_arguments> <cmd_options>
```

where the `<general_options>` refer to `edg-lrc` or `edg-rmc`, `<cmd_name>` is the particular command or action that must be performed, and `<cmd_arguments>` and `<cmd_options>` refer to that command. Most commands have both an extended and an abbreviated name form.

NOTE: If the above described order is not followed (general options before the command name, and particular options after it) the general and command-specific options may be mixed, resulting in a fail of the command.

When dealing with the catalogs using the `edg-lrc` or `edg-rmc` commands, the `guid:` and `lfn:` prefixes must be used if an entry is being added, but they can be omitted when consulting. This is so because with these commands is always clear if a GUID, a LFN or a SURL is being used. In this guide, though, we will always use the prefixes.

Only some usage examples of the most important commands will be given here. For detailed information please refer to [R23] and [R24].

6.3.1. Local Replica Catalog Commands

The `edg-lrc` commands operate with GUID-SURLs mappings. The `-i` option is used to connect to the LRC using `http` instead of `https` (sometimes it may be the only available way to connect to the server).

Note: For historical reasons, the commands name and in the manpages, the SURL is often called *PFN* (for *Physical File Name*).

All the commands require the LRC ***endpoint***, which can be obtained using the `lcg-infosites` command. This usually takes the form:

```
http(s)://<host>:<port>/<VO>/edg-local-replica-catalog/services/edg-local-replica-catalog
```

It can be specified either using the `--endpoint` option followed by the full endpoint, or setting the values for the hostname, the port and the VO to be used, with the `-h`, `-p` and `--vo` options respectively. It is safer to use the `--endpoint` option, since it does not make any assumption regarding the path.

The following tables summarize the most useful commands;

Mapping management commands:

`addMapping guid pfn`	Adds the given mapping to the catalog.
`pfnExists pfn`	Checks whether the given PFN exists in the catalog.
`guidExists guid`	Checks whether the given GUID exists in the catalog.
`guidForPfn pfn`	Returns the GUID for a given PFN.
`pfnsForGuid guid`	Returns the PFNs for a given GUID.
`removePfn guid pfn`	Removes a PFN from a given GUID.

Wildcard query commands (to retrieve GUIDs or SURLs that match a pattern):

`mappingsByPfn pfnPattern`	Gets a set of mappings by a wildcard search on PFN name.
`mappingsByGuid guidPattern`	Gets a set of mappings by a wildcard search on guid.
`getResultLength`	Returns the result length by default (i.e. how many mappings will be returned when using `mappingsByGuid` or `mappingsByPfn`).
`setResultLength length`	Sets the result length by default (i.e. how many mappings will be returned when using `mappingsByGuid` or `mappingsByPfn`).

Attribute management commands (metadata associated with GUID-SURL mapings):

`listAttrDefns`	Lists all attributes (name and type) that are defined in the LRC.
`attrDefnExists name`	Checks whether the given attribute exists.
`getPfnAttr pfn attrName`	Gets the given attribute value.
`setPfnAttr pfn attrName value`	Sets the given attribute's value.
`removePfnAttr pfn attrName`	Removes the given atribute.
`mappingsByAttr attributeConditions`	Returns the mappings whose pfn attributes match the given attribute conditions.

Examples.

For clarity reasons, environmental variables -rather than long file names- are used in the following examples. Thus, it will be assumed that a file is registered in the Grid with its GUID, SURL and LFN assigned to:

```
$ setenv GUID  guid:c06a92ee-6911-11d8-a453-d9c1af867039
$ setenv SURL  sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_test1
$ setenv ALIAS lfn:lasts_results
```

In addition, some false values (not assigned to any real file) are defined:

```
$ setenv GUID2  guid:c06a92ee-6911-11d8-a453-000000000000
$ setenv SURL2  sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_fake
$ setenv ALIAS2 lfn:fake_alias
```

Finally, we will use another variable for the `--endpoint` option:

```
$ setenv LRC_ENDPOINT http://rlscert01.cern.ch:7777/dteam/v2.2/edg-local-replica-catalog/
services/edg-local-replica-catalog
```

Example 6.3.1.1 (Checking existence of SURLs and GUIDs)

Confirming that $SURL and $GUID exist, but $SURL2 does not:

```
$ edg-lrc pfnExists $SURL --endpoint $LRC_ENDPOINT
> Pfn exists : 'sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_test1'

$ edg-lrc guidExists $GUID --endpoint $LRC_ENDPOINT
> GUID exists : 'guid:c06a92ee-6911-11d8-a453-d9c1af867039'

$ edg-lrc pfnExists $SURL2 --endpoint $LRC_ENDPOINT
> Pfn does not exist : 'sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_fake'
```

Example 6.3.1.2 (Retrieving SURLs and GUIDs)

Retrieving the GUID for a SURL.

```
$ edg-lrc guidForPfn $SURL --endpoint $LRC_ENDPOINT
> guid:c06a92ee-6911-11d8-a453-d9c1af867039
```

Retrieving the SURLs for a GUID (if it exists):

```
$ edg-lrc pfnsForGuid $GUID --endpoint $LRC_ENDPOINT
> sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_test1

$ edg-lrc pfnsForGuid $GUID2 --endpoint $LRC_ENDPOINT
> No such guid : 'guid:c06a92ee-6911-11d8-a453-00000000000'
```

Example 6.3.1.3 (Retrieving with wildcards)

Retrieving GUIDs for a SURL pattern:

```
$ edg-lrc mappingsByPfn '*my_test*' --endpoint $LRC_ENDPOINT
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30,
sfn://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/my_test3
> guid:c06a92ee-6911-11d8-a453-d9c1af867039,
sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_test1
```

Retrieving SURLs for a GUID pattern:

```
$ edg-lrc mappingsByGuid '*b3fa*' --endpoint $LRC_ENDPOINT
> guid:0abdd087-5a43-11d8-b57f-a48b3faf9ccd,
sfn://lxshare0291.cern.ch/flatfiles/LCG-CERT-SE03/dteam/generated/2004/02/08/file05e657d6
-5a43-11d8-b57f-a48b3faf9ccd
> guid:0abdd087-5a43-11d8-b57f-a48b3faf9ccd,
sfn://lxshare0236.cern.ch/flatfiles/LCG-CERT-SE01/dteam/generated/2004/02/08/file1010ba9e
-5a43-11d8-9971-fa6a704d33db
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30, sfn://wacdr002d.cern.ch/castor/cern.ch/grid/
dteam/my_test3
[...]
```

Example 6.3.1.4 (Adding a mapping)

Adding a mapping with a false SURL:

```
$ edg-lrc addMapping $GUID $SURL2 --endpoint $LRC_ENDPOINT

$ edg-lrc pfnExists $SURL2 --endpoint $LRC_ENDPOINT
> Pfn exists : 'sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_fake
```

Example 6.3.1.5 (Removing a mapping)

Removing the previously added SURL:

```
$ edg-lrc removePfn $GUID $SURL2 --endpoint $LRC_ENDPOINT
```

Example 6.3.1.6 (Checking the attributes defined for GUID-SURL mappings)

Checking the existence of the `size` attribute.

```
$ edg-lrc attrDefnExists size --endpoint $LRC_ENDPOINT
> Attribute definition exists : 'size'

$ edg-lrc attrDefnExists fakeAttr --endpoint $LRC_ENDPOINT
> Attribute definition does not exist : 'fakeAttr'
```

In fact, `size` is the only attribute that is currently set and used in LCG-2. The user may find some other attributes defined, but their value will then be always null.

Example 6.3.1.7 (Retrieving and setting an attribute)

Retrieving it:

```
$ edg-lrc getPfnAttr $SURL size --endpoint $LRC_ENDPOINT
> 30
```

Setting it (although this practice is not recommended or sensible at all):

```
$ edg-lrc setPfnAttr $SURL size 150 --endpoint $LRC_ENDPOINT

$ edg-lrc getPfnAttr $SURL size --endpoint $LRC_ENDPOINT
> 150
```

Example 6.3.1.8 (Unsetting an attribute)

Unsetting the `size` attribute (again not a very sensible practice).

```
$ edg-lrc removePfnAttr $SURL size --endpoint $LRC_ENDPOINT

$ edg-lrc getPfnAttr $SURL size --endpoint $LRC_ENDPOINT
> null
```

Example 6.3.1.9 (Retrieving SURLs based on attributes condition)

The condition on the attributes has the format of an SQL query, where comparison and boolean operators are allowed. It is important to notice that the prefix `pfn.` must precede the attribute names (as must the `guid.` for RMC attributes). In the following example, we retrieve all the stored SURLs whose size is less than 150 bytes:

```
$ edg-lrc -v mappingsByAttr "pfn.size < '150'" --endpoint $LRC_ENDPOINT
> guid:5cd3ab36-5c7e-11d8-beac-ef183bb6fbad, sfn://lxshare0236.cern.ch/flatfiles/LCG-CERT
-SE01/dteam/generated/2004-02-11/file58ff71b5-5c7e-11d8-beac-ef183bb6fbad
> guid:dbe8eefc-5c7e-11d8-9889-957166856d29, sfn://lxshare0236.cern.ch/flatfiles/LCG-CERT
-SE01/dteam/generated/2004-02-11/filed7da1d7b-5c7e-11d8-9889-957166856d29
> guid:20922401-5c7f-11d8-912c-f27e165efd20, sfn://lxshare0236.cern.ch/flatfiles/LCG-CERT
-SE01/dteam/generated/2004-02-11/file1c525770-5c7f-11d8-912c-f27e165efd20
> guid:185612cd-6305-11d8-831e-9e96df859b8a, sfn://castorgrid.cern.ch/castor/cern.ch/grid
/dteam/output.txt
> guid:3b43991d-637c-11d8-b4a4-adc4d7141a27, sfn://tbed0101.cern.ch/flatfile/SE00/dteam/o
utput.txt
> guid:f31de2ee-6889-11d8-848c-83bed4b833ae, sfn://lxshare0291.cern.ch/flatfiles/LCG-CERT
-SE03/dteam/generated/2004-02-26/filee2062a2b-688a-11d8-86a0-cf5744832cb8
[...]
```

6.3.2. Replica Metadata Catalog Commands

The `edg-rmc` commands operate with GUID-LFNs mappings. The `-i` option is used in the same way as with `edg-lrc`, and so are the options used to specify the endpoint for the RMC server (which, again, can be obtained with the `lcg-infosites` command).

The following tables summarize the most useful commands;

Mapping management commands:

`addAlias guid alias`	Adds a new alias to the catalog.
`aliasExists alias`	Checks whether the given alias exists in the catalog.
`guidExists guid`	Checks whether the given GUID exists in the catalog.
`guidForAlias alias`	Returns the GUID for the given alias.
`aliasesForGuid guid`	Returns the aliases for the given GUID.
`removeAlias guid alias`	Removes an alias from the given GUID.

Wildcard query commands (to retrieve GUIDs or SURLs that match a pattern):

`mappingsByAlias aliasPattern`	Gets a set of mappings by a wildcard search on alias name.
`mappingsByGuid guidPattern`	Gets a set of mappings by a wildcard search on guid.
`getResultLength`	Returns the result length by default (i.e. how many mappings will be returned when using `mappingsByGuid` or `mappingsByAlias`).
`setResultLength length`	Sets the result length by default (i.e. how many mappings will be returned when using `mappingsByGuid` or `mappingsByAlias`).

As in the case of `edg-lrc`, there are some other commands that set/get attributes for the GUIDs or the aliases, and one that retrieve mappings whose attributes satisfy certain conditions. Those commands behave much like their LRC counterparts and no examples for them will be provided in this guide. If the reader is interested in them, he/she can refer to the manpages or to [R24].

Examples.

The same environmental variables of the previous section are used in the following examples. In addition, we define a new one for the RMC endpoint option:

```
$ setenv RMC_ENDPOINT http://rlscert01.cern.ch:7777/dteam/v2.2/edg-replica-metadata-catal
og/services/edg-replica-metadata-catalog
```

Example 6.3.2.1 (Checking the existence of GUIDs and LFNs)

Confirming that $ALIAS exists but $ALIAS2 does not.

```
$ edg-rmc aliasExists $ALIAS --endpoint $RMC_ENDPOINT
> Alias exists : 'lfn:last_results'

$ edg-rmc guidForAlias $ALIAS2 --endpoint $RMC_ENDPOINT
> No such alias : 'lfn:fake_alias'
```

The same for $GUID and $GUID2.

```
$ edg-rmc guidExists $GUID --endpoint $RMC_ENDPOINT
> GUID exists : 'guid:c06a92ee-6911-11d8-a453-d9c1af867039'

$ edg-rmc guidExists $GUID2 --endpoint $RMC_ENDPOINT
> GUID does not exist : 'guid:c06a92ee-6911-11d8-a453-00000000000'
```

Example 6.3.2.2 (Retrieving LFNs and GUIDs)

Retrieving the GUID for a known alias.

```
$ edg-rmc guidForAlias $ALIAS --endpoint $RMC_ENDPOINT
> guid:c06a92ee-6911-11d8-a453-d9c1af867039
```

Retrieving the existing aliases for a GUID.

```
$ edg-rmc aliasesForGuid $GUID --endpoint $RMC_ENDPOINT
> lfn:last_results
```

Example 6.3.2.3 (Adding new LFNs)

In order to add a new alias, the `guid:` and `lfn:` prefixes must be used. Consider the following example, where only the last command is accepted:

```
$ edg-rmc addAlias c06a92ee-6911-11d8-a453-d9c1af867039 \
lfn:new_results --endpoint $RMC_ENDPOINT
> Error: addAlias: Invalid file type for URI : 'c06a92ee-6911-11d8-a453-d9c1af867039',
reason: Scheme is not 'guid'

$ edg-rmc addAlias $GUID new_results --endpoint $RMC_ENDPOINT
> Error: addAlias: Invalid file type for URI : 'new_results', reason: Scheme is not 'lfn'

$ edg-rmc addAlias $GUID lfn:new_results --endpoint $RMC_ENDPOINT
```

Example 6.3.2.4 (Retrieving with wildcards)

Using an alias pattern, two mappings are returned:

```
$ edg-rmc mappingsByAlias '*result*' --endpoint $RMC_ENDPOINT
> guid:c06a92ee-6911-11d8-a453-d9c1af867039, lfn:last_results
> guid:c06a92ee-6911-11d8-a453-d9c1af867039, lfn:new_results
```

A GUID pattern can also be used:

```
$ edg-rmc mappingsByGuid $GUID --endpoint $RMC_ENDPOINT
> guid:c06a92ee-6911-11d8-a453-d9c1af867039, lfn:last_results
> guid:c06a92ee-6911-11d8-a453-d9c1af867039, lfn:new_results
```

Example 6.3.2.5 (Deleting an LFN)

The previously added mapping is removed:

```
$ edg-rmc removeAlias $GUID lfn:new_results --endpoint $RMC_ENDPOINT
```

6.4. LOW LEVEL DATA MANAGEMENT TOOLS

The low level tools allow users to perform some actions on the GSIFTP server of a SE. A brief summary of their functions follows:

`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.
`globus-url-copy sourceURL destURL`	Copies files between SEs.
`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 GSIFTP server.
`edg-gridftp-rmdir URL`	Removes a directory on a SE.

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 catalogues 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.

Some examples of possible uses of some of these commands follow. They are by no means exhaustive. To obtain help on these commands use the option `--usage` or `--help`. General information on GSIFTP is available in [R10].

Example 6.4.1 (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.

The `edg-gridftp-exists` command will output an error message if the specified file does not exist and nothing if the file does exist.

```
$ 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
```

Notice that, as for the other `edg-gridftp-*` commands, files are identified by a URL with the `gsiftp` protocol (the one supported in every SE or SRM in the Grid). A URL is also used to specify the SE's

directory for the the `edg-gridftp-ls` command to list. In the following example only one file is found in the specified location:

```
$ edg-gridftp-ls gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13
> file42ff7086-8063-414d-9000-75c459b71296
```

Example 6.4.2 (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`, and as this command, `globus-url-copy` does not register or unregister files, and so it can corrupt the LCG-2 catalogs. Use it only in case of real necessity.

As in the previous commands, the arguments of `globus-url-copy` are URLs. They can have the prefix `gsiftp://` for files stored in Storage Elements, or the prefix `file://` for files in the local filesystem. In the following example the file whose existence was confirmed above, is retrieved from its SE to the local filesystem.

```
globus-url-copy gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/file42
ff7086-8063-414d-9000-75c459b71296 file://`pwd`/my_file
```

No error message is shown. The file has been copied.

6.5. ACCESSING A GRID FILE FROM A JOB

As seen in Chapter 5, a job that is submitted to the Grid can access files stored in LCG-2. For that purpose, the JDL file of the job should include[8] the name (GUID or LFN) of the files to be accessed, in the `InputData` attribute; and the protocol that will be used to access them. Currently, the only two supported protocols to access Grid files in a SE are: GSIFTP (`gsiftp`), for file copy, and RFIO (`rfio`), for file access. When dCache is deployed, dCache access protocols will be also supported for remote file access, for SEs both in local or remote LANs.

The code used to effectively access the Grid files must be included in the submitted job. This code may consist of calls to an API from a C/C++ program, which should be the more general and efficient method, or it may use command line tools from a Perl or shell script. In the first case, the user should use the C API of the ***Grid File Access Library (GFAL)***. As for the second option, the LCG Data Management or RFIO client tools may be called from a script.

These concepts are explained in more detail in the following sections.

[8] For a discussion on what this *should* means refer to Section 5.1.

6.5.1. Accessing a File Using an Application Programming Interface

The development of code for the jobs that are submitted to LCG-2 is out of the scope of this guide, and therefore the different APIs for accessing Grid files will not be covered in detail here. This section just summarizes what APIs exist and gives an example of GFAL use. For general information on APIs for accessing files and other Grid resources, refer to [R5].

The Grid File Access Library is the currently preferred API for accessing files in LCG-2, and the only one that should be used by developers. The GFAL 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. This means that GFAL can deal with GSIFTP, RFIO, or even the dCache access protocols, but that is transparent for the user (unless a TURL is used, in which case the protocol is explicitely indicated).

Example 6.5.1.1 (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;
}
```

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 can of course specify also a SURL so that we store the file in an LCG-2 Storage Element, and we read it from there afterwards. And if the program had been used to read the contents of an existing LCG-2 file, then the GUID or any LFN of the file could have been also used to access it.

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 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 RFIO does not handle Grid certificates yet, 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.

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
```

For classic SEs that use CASTOR as a backend, the SURL and TURL should not contain the hostname, but only the path. Example of these are:

```
sfn:///castor/cern.ch/grid/dteam/my_file
rfio:///castor/cern.ch/grid/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 register 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 [R5] and 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 (an example of this will be shown in the next section).

Finally, as was explained previously, 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 RFIO (as is the case for the current classic SEs), then GFAL calls will also use 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 [R11].

6.5.2. Accessing a File through Client Tools

Although GFAL is the preferred method to access Grid files in LCG-2, a user may desire to call some client tools from a scripting language, without having to write a program in C or C++. Both the RFIO client tools and the LCG Data Management tools can be used to copy files stored in an Storage Element to the local filesystem of a Worker Node. Then, the file can be accessed locally. Notice that in this way, the file is not accessed remotely. Instead, it is copied to the WN.

This section gives two complete examples of jobs that access files stored in a SE in the LCG-2 Grid. The first example uses the LCG Data Management Tools, which use GSIFTP, and the second one uses the RFIO tools, which relies on the RFIO protocol, and therefore can only be used to access files located in the same local area network of the WN.

Example 6.5.2.1 (Copying a file using the LCG Data Management Tools)

We assume that a user has registered a data file (called `values`) within LCG-2, using `lfn:example_values` as its LFN. The contents of the file are the following:

```
The contents of these lines,
which are not really important,
will be shown in the std.out file.
```

The JDL file of the job (`example.jdl`) includes the LFN of the file, and the protocol (`gsiftp`) to be used when accessing it. The contents of the JDL file follow:

```
Executable="example.pl";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"example.pl"};
OutputSandbox={"std.out","std.err"};
InputData={"lfn:example_values"};
DataAccessProtocol={"gsiftp"};
```

The executable (`example.pl`) is a perl program, that calls the `lcg-cp` command (already explained) to copy the Grid file to the local filesystem of the Worker Node where the job is running. The rest of the script is simple perl code to show the data retrieved:

```
#!/usr/bin/perl

# Copy the input data file to the WN local filesystem
system "lcg-cp --vo=dteam lfn:example_values file:`pwd`/values";

# Open it
open(file,'values');

# Read all the lines
```

```
@lines=<file>;

#Show the info
print "The values stored in the input data file are:\n";
print " @lines";
```

The job is submitted as usual:

```
$ edg-job-submit -o jobid example.jdl
```

And the results retrieved with:

```
$ edg-job-get-output -i jobid
```

The `std.out` file obtained is this:

```
The values stored in the input data file are:
 The contents of these lines,
 which are not really important,
 will be shown in the std.out file.
```

Example 6.5.2.2 (Copying a file using the RFIO protocol)

This example is very similar to the previous one, but here the RFIO protocol is used. The same data file `values` is used, and the only changes in the new `example2.jdl` file are the executable file, and the access protocol:

```
Executable="example2.pl";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"example2.pl"};
OutputSandbox={"std.out","std.err"};
InputData={"lfn:example_values"};
DataAccessProtocol={"rfio"};
```

The `example2.pl` file is a bit more complicated this time, because the RFIO protocol cannot handle LFNs, and needs the complete path to the file instead. For this reason, the TURL of the file is obtained first and then it is adapted to RFIO needs. The commands to get the TURL from a known LFN have been already seen and could be also performed manually beforehand instead of inserting them in the perl script, but are included here for completeness. The form of the TURL will be: `rfio://<hostname>/<path>`, while the RFIO command expects a `<hostname>:<path>` string, and therefore the perl code has to do a little extra work to adapt the string before invoking the `rfcp` command, which copies the file to the WN local filesystem.

```
#!/usr/bin/perl
```

```
# Obtain the SURL of the file whose LFN we know
# (if several replicas exist, the closest one should be selected)
$surl= `lcg-lr --vo dteam lfn:example_values`;
chop($surl);

# Now obtain the TURL for the RFIO protocol
@lcg-gt_output=`lcg-gt $surl rfio`;  # this returns three lines
$turl=$lcg-gt_output[0];             # the interesting one is just the first one

# Adapt the returned "rfio://hostname/path" to the "hostname:path" format that RFIO uses
$turl =~ s/rfio:\/\///;  # delete the extra "rfio://"
$turl =~ s/\//:\//;      # add the ":"
chop($turl);

# Copy the input data file to the WN local filesystem
system "rfcp $turl `pwd`/values";

# Open it
open(file,'values');

# Read all the lines
@lines=<file>;

#Show the info
print "The values stored in the input data file are:\n";
print " @lines";
```

The job is submitted and the output retrieved like in the previous example, and the retrieved `std.out` file is:

```
95 bytes in 0 seconds through eth0 (in) and local (out)
The values stored in the input data file are:
 The contents of these lines,
 which are not really important,
 will be shown in the std.out file.
```

where the first line is produced by the `rfcp` command.

6.6. POOL AND LCG-2

The ***POOL*** tool (**P**OOL **O**f persistent **O**bjects for **L**HC) is used by most of the LHC experiments as a common persistency framework for the LCG application area. Objects created by users using POOL are stored into its own File Catalog (XML Catalog). This File Catalog keeps track of all POOL databases and resolves file references into PFN which are then used by lower level components like the storage service to access file contents.

Until now, the POOL catalog (XML) and the EDG Replica location Service (RLS) were working in parallel. This could cause some conflicts between files created and registered into the XML catalog with those files registered into the RLS. The new LCG-2 release has updated its software to make entries in XML and RLS compatible and consistent.

6.6.1. LCG Catalog (RLS) vs POOL Catalog (XML)

In order to make the RLS and POOL catalogs work together, a problem of compatibility had to be solved.

- The LCG Data Management tools write entries in the RLS for LFNs, GUIDs and SURLs giving explicitly the prefix of the entry. For example, these are typical RLS entries:

```
LFN  - lfn:this-is-my-logical-file-name
GUID - guid:73e16e74-26b0-11d7-b1e0-c5c68d88236a
SURL - sfn://lxshare0384.cern.ch/flatfiles/cms/data/05/x.dat
```

- However, POOL stores the same entries as follows:

```
LFN  - this-is-my-logical-file-name
GUID - 73e16e74-26b0-11d7-b1e0-c5c68d88236a
SURL - sfn://lxshare0384.cern.ch/flatfiles/cms/data/05/x.dat
     - any format accepted by ROOT
     ex.: /home/user/mydata/file.dat
     rfio:/afs/cern.ch/project/cms/data/file.dat
     d-cache:/pool/disks/cms/file.dat
```

The problem was therefore that an entry inserted by POOL in the RLS cannot be processed by the Data Management tools and viceversa. LCG-2 solved this problem changing the LCG tools to store LFNs and GUIDs as POOL does (i.e., without the `guid:` and `lfn:` prefixes).

Example 6.6.1.1 (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.

- Basically it is just 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>   (sh shell)
$ setenv POOL_CATALOG edgcatalog_http://<host>:<port>/<path>   (csh shell)
```

  For example into LCG-2 this environment variable should 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 a 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 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 [R27]. And in a machine where the interface is installed, the user can see them just by typing `FC<tab>` (or, if that does not work, `FC<Ctrl-D>`).

7. INFORMATION SYSTEM

In the following sections examples are given on how to interrogate the Information System in LCG-2 Grid. In particular, the different servers from which the information can be obtained are discussed. These are the local GRISes, the site GIISes and the global BDIIs. As explained earlier, the data in the IS of LCG-2 conforms to the LDAP implementation of the GLUE Schema. 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 C.

NOTE: In the new release of LCG, the `GlueCEPolicyMaxWallClockTime` and `GlueCEPolicyMaxCPUTime` attributes are measured in seconds, and not in minutes as it was previously.

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 ditributions by default. Some comments on this tool are given in section 7.3.

7.1. 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 C.

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 7.1.1 (Interrogating the GRIS on a Computing Element)

The command used to interrogate the GRIS located on host `lxn1181` is:

```
$ ldapsearch -x -h lxn1181.cern.ch -p 2135 -b "mds-vo-name=local, o=grid"
```

or:

```
$ ldapsearch -x -H ldap://lxn1181.cern.ch:2135 -b "mds-vo-name=local, o=grid"
```

And the obtained reply will be:

```
version: 2

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

# lxn1181.cern.ch/siteinfo, local, grid
dn: in=lxn1181.cern.ch/siteinfo,Mds-Vo-name=local,o=grid
objectClass: SiteInfo
objectClass: DataGridTop
objectClass: DynamicObject
siteName: CERN-LCG2
sysAdminContact: hep-project-grid-cern-testbed-managers@cern.ch
userSupportContact: hep-project-grid-cern-testbed-managers@cern.ch
siteSecurityContact: hep-project-grid-cern-testbed-managers@cern.ch
dataGridVersion: LCG-2_0_0beta
installationDate: 20040106120000Z

# lxn1181.cern.ch:2119/jobmanager-lcgpbs-infinite, local, grid
dn: GlueCEUniqueID=lxn1181.cern.ch:2119/jobmanager-lcgpbs-infinite, mds-vo-name=local,
    o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
objectClass: GlueCEInfo
objectClass: GlueCEPolicy
objectClass: GlueCEState
objectClass: GlueInformationService
objectClass: GlueKey
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1
GlueCEName: infinite
GlueCEUniqueID: lxn1181.cern.ch:2119/jobmanager-lcgpbs-infinite
GlueCEInfoGatekeeperPort: 2119
GlueCEInfoHostName: lxn1181.cern.ch
```

```
GlueCEInfoLRMSType: pbs
GlueCEInfoLRMSVersion: OpenPBS_2.4
GlueCEInfoTotalCPUs: 16
GlueCEStateEstimatedResponseTime: 0
GlueCEStateFreeCPUs: 16
GlueCEStateRunningJobs: 0
GlueCEStateStatus: Production
GlueCEStateTotalJobs: 0
GlueCEStateWaitingJobs: 0
GlueCEStateWorstResponseTime: 0
GlueCEPolicyMaxCPUTime: 172800
GlueCEPolicyMaxRunningJobs: 99999
GlueCEPolicyMaxTotalJobs: 999999
GlueCEPolicyMaxWallClockTime: 259200
GlueCEPolicyPriority: 1
GlueCEAccessControlBaseRule: VO:alice
GlueCEAccessControlBaseRule: VO:atlas
GlueCEAccessControlBaseRule: VO:cms
GlueCEAccessControlBaseRule: VO:lhcb
GlueCEAccessControlBaseRule: VO:dteam
GlueForeignKey: GlueClusterUniqueID=lxn1181.cern.ch
GlueInformationServiceURL: ldap://lxn1181.cern.ch:2135/mds-vo-name=local,o=grid
[...]
```

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 C.

Example 7.1.2 (Getting information about the site name from the GRIS on a CE)

```
$ ldapsearch -x -h lxn1181.cern.ch -p 2135 -b "mds-vo-name=local, o=grid" \
'objectclass=SiteInfo' siteName

version: 2

#
# filter: objectclass=SiteInfo
# requesting: siteName
#

# lxn1181.cern.ch/siteinfo, local, grid
dn: in=lxn1181.cern.ch/siteinfo,Mds-Vo-name=local,o=grid
siteName: CERN-LCG2

# 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=SiteInfo' siteName

dn: in=lxn1181.cern.ch/siteinfo,Mds-Vo-name=local,o=grid
siteName: CERN-LCG2
```

7.2. THE SITE GIIS

At each site, a site GIIS collects information about all resources present at a site (i.e. data from all GRISes of the site).

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

Usually a site GIIS runs on a Computing Element. In order to, for example, interrogate the site GIIS for PIC (Barcelona), one needs to find out the name of that CE. This can be found in the GOC databse, in:

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

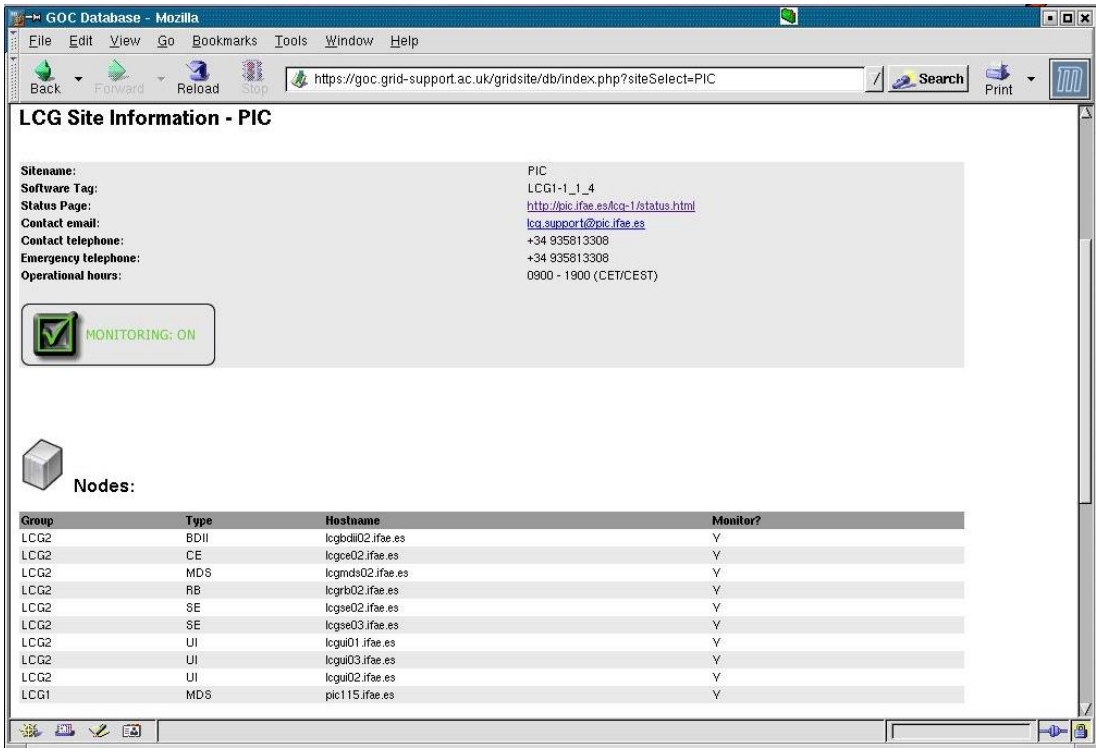

Figure 8: The status page of the PIC site

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`). In this case, the value of the `mds-vo-name` attribute of the second entry in the DIT informs us about the site from which the GIIS collects information. This value is just the site name, which is published by all sites, where all "-" characters have been removed. This base name is written in lowercase. So, for instance, for site PIC (Barcelona), the site name is `PIC-LCG2` and the DN to be used as base name is `mds-vo-name=piclcg2, o=grid`.

Note: In the GOC web page, you may find, besides the published site name, a more friendly name for a site (e.g., simply `PIC`). For the site GIIS base name, be sure to use the site name that is published in the Information Service (`PIC-LCG2`), or the LDAP query will not work.

As we can see in Figure 8, the CE name is `lcgce02.ifae.es`. So, in order to interrogate the site GIIS,

we can use the command shown in the following example:

Example 7.2.1 (Interrogating the site GIIS)

```
$ ldapsearch -x -H ldap://lcgce02.ifae.es:2135 -b "mds-vo-name=piclcg2,o=grid"

version: 2

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

# lcgse03.ifae.es, piclcg2, grid
dn: GlueSEUniqueID=lcgse03.ifae.es,Mds-Vo-name=piclcg2,o=grid
objectClass: GlueSETop
objectClass: GlueSE
objectClass: GlueInformationService
objectClass: Gluekey
objectClass: GlueSchemaVersion
GlueSEUniqueID: lcgse03.ifae.es
GlueSEName: PIC-LCG2:disk
GlueSEPort: 2811
GlueInformationServiceURL: ldap://lcgse03.ifae.es:2135/Mds-Vo-name=local,o=gri
 d
GlueForeignKey: GlueSLUniqueID=lcgse03.ifae.es
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1

[...]

# lcgse03.ifae.es, piclcg2, grid
dn: GlueSLUniqueID=lcgse03.ifae.es,Mds-Vo-name=piclcg2,o=grid
objectClass: GlueSLTop
objectClass: GlueSL
objectClass: GlueSLArchitecture
objectClass: Gluekey
objectClass: GlueSchemaVersion
GlueSLUniqueID: lcgse03.ifae.es
GlueSLName: PIC-LCG2
GlueSLArchitectureType: mss
GlueForeignKey: GlueSEUniqueID=lcgse03.ifae.es
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1

[...]

# lcgce02.ifae.es/siteinfo, piclcg2, grid
```

```
dn: in=lcgce02.ifae.es/siteinfo,Mds-Vo-name=piclcg2,o=grid
objectClass: SiteInfo
objectClass: DataGridTop
objectClass: DynamicObject
siteName: PIC-LCG2
sysAdminContact: lcg.support@pic.ifae.es
userSupportContact: lcg.support@pic.ifae.es
siteSecurityContact: lcg.support@pic.ifae.es
dataGridVersion: lcg2_20040225_1700
installationDate: 20040109180000Z

[...]
```

7.3. THE BDII

Each site running a Resource Broker can run as well a BDII, which collects all information coming from site GIISes and stores them in a permanent database. The BDII can be configured to get publish information from resources in all sites, or only from some of them. In order to find out the location of the BDII you can consult the GOC web page as done for the site GIISes.

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 9, a view of the DIT of the BDII of the LCG-2 is shown. This image has been obtained using the program *gq*. 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.

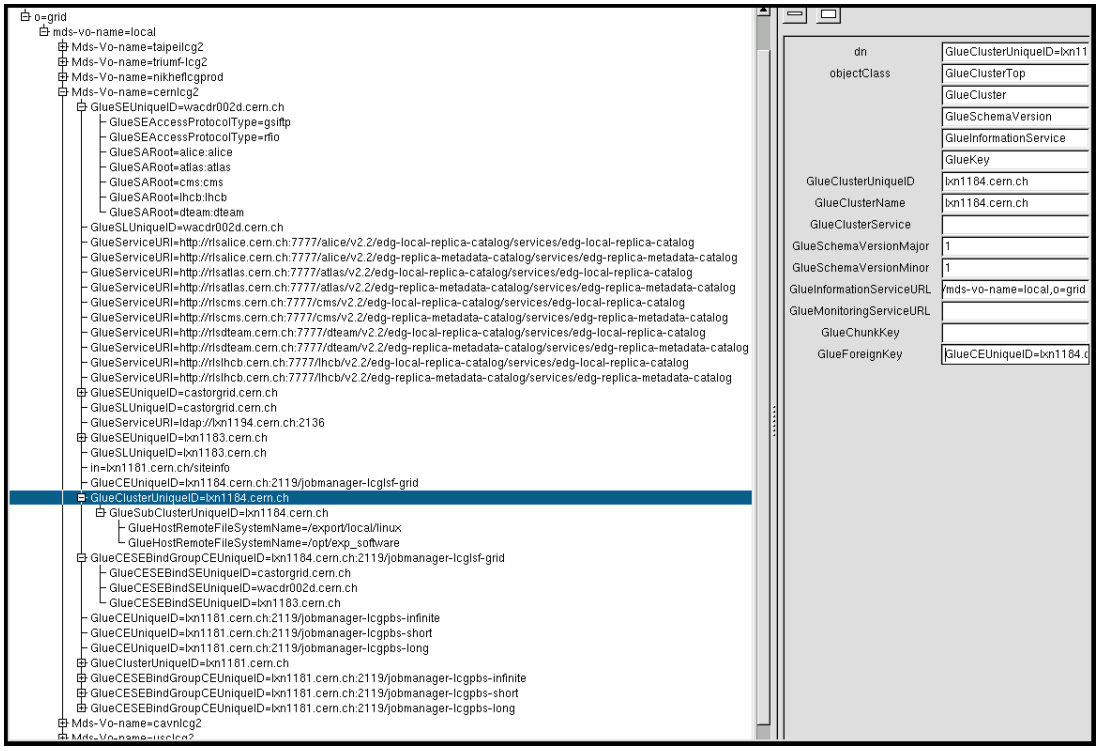

Figure 9: The LDAP directory of an LCG-2 BDII

Example 7.3.1 (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://lxshare0222.cern.ch:2170 -b "mds-vo-name=local,o=grid" \
'objectclass=GlueCESEBind' GlueCESEBindCEUniqueID GlueCESEBindSEUniqueID

dn: GlueCESEBindSEUniqueID=grid100.kfki.hu,
GlueCESEBindGroupCEUniqueID=grid109.kfki.hu:2119/jobmanager-pbs-infinite,
Mds-Vo-name=budapestlcg1, Mds-Vo-name=lcgeast, Mds-Vo-name=local,o=grid
GlueCESEBindCEUniqueID: grid109.kfki.hu:2119/jobmanager-pbs-infinite
GlueCESEBindSEUniqueID: grid100.kfki.hu

dn: GlueCESEBindSEUniqueID=grid100.kfki.hu,
GlueCESEBindGroupCEUniqueID=grid109.kfki.hu:2119/jobmanager-pbs-long,
Mds-Vo-name=budapestlcg1, Mds-Vo-name=lcgeast, Mds-Vo-name=local,o=grid
GlueCESEBindCEUniqueID: grid109.kfki.hu:2119/jobmanager-pbs-long
GlueCESEBindSEUniqueID: grid100.kfki.hu

dn: GlueCESEBindSEUniqueID=grid100.kfki.hu,
GlueCESEBindGroupCEUniqueID=grid109.kfki.hu:2119/jobmanager-pbs-short,
```

```
Mds-Vo-name=budapestlcg1, Mds-Vo-name=lcgeast, Mds-Vo-name=local,o=grid
GlueCESEBindCEUniqueID: grid109.kfki.hu:2119/jobmanager-pbs-short
GlueCESEBindSEUniqueID: grid100.kfki.hu

dn: GlueCESEBindSEUniqueID=adc0021.cern.ch,
GlueCESEBindGroupCEUniqueID=adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite,
Mds-Vo-name=cernlcg1,Mds-Vo-name=lcgeast,Mds-Vo-name=local,o=grid
GlueCESEBindCEUniqueID: adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite
[...]
```

Example 7.3.2 (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 lxshare0222.cern.ch -p 2170 -b "mds-vo-name=local,o=grid" \
-x 'objectclass=GlueSubCluster' GlueChunkKey GlueHostApplicationSoftwareRunTimeEnvironment
```

Example 7.3.3 (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 af all SEs supporting ATLAS, together with the storage space avalaible in each of them, the `GlueSAAccessControlBaseRule`, which specifies a supported VO, is used:

```
$ ldapsearch -LLL -h  lxn1178.cern.ch -p 2170 -b \
"mds-vo-name=local,o=grid" -x "GlueSAAccessControlBaseRule=atlas" \
GlueChunkKey GlueSAStateAvailableSpace GlueSAStateUsedSpace
```

And the obtained result will be something like the following:

```
dn: GlueSARoot=atlas:atlas,GlueSEUniqueID=lcg00123.grid.sinica.edu.tw,Mds-Vo-n
 ame=taipeilcg2,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 1956026048
GlueSAStateUsedSpace: 1573892
GlueChunkKey: GlueSEUniqueID=lcg00123.grid.sinica.edu.tw

dn: GlueSARoot=atlas:atlas,GlueSEUniqueID=lcgse01.triumf.ca,Mds-Vo-name=triumf
 -lcg2,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 759426780
GlueSAStateUsedSpace: 7212348
GlueChunkKey: GlueSEUniqueID=lcgse01.triumf.ca
```

```
dn: GlueSARoot=atlas:atlas,GlueSEUniqueID=tbn17.nikhef.nl,Mds-Vo-name=nikheflc
 gprod,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 1464065168
GlueSAStateUsedSpace: 359189348
GlueChunkKey: GlueSEUniqueID=tbn17.nikhef.nl

dn: GlueSARoot=atlas:atlas,GlueSEUniqueID=wacdr002d.cern.ch,Mds-Vo-name=cernlc
 g2,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 1000000000000
GlueSAStateUsedSpace: 1000000000000
GlueChunkKey: GlueSEUniqueID=wacdr002d.cern.ch
[...]
```

7.4. MONITORING SYSTEMS

The ability to monitor resource related parameters is currently considered a necessary functionality in any network. In such an hetereogeneus 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 LCG-2, 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 LCG-2. Several different monitoring tools can be used, including general purpose monitoring tools and Grid specific systems like GridIce [R28].

In the following section, the GridIce service is reviewed, since it is currently the officially supported monitoring system in LCG-2.

7.4.1. GridIce

The GridIce monitoring service is structured in a five layer architecture. The resource information is obtained from the LCG-2 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 [R28] 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 configurate event detection and notification actions; though these two functionalities are still at an early development stage.

The GridIce web page that shows the monitoring information for LCG-2 is accessible at the following URL (also linked in the GOC web site):

http://grid-ice.esc.rl.ac.uk/gridice

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.

APPENDIX A EXPERIMENTS SOFTWARE INSTALLATION

This section gives an overview of a possible approach to software installation in LCG-2's CEs. Since the LCG-2 Grid includes many sites, each with its own administrators and possibly with different security and software installation policies, there must be some cooperation between the experiments[9] (which want to install their software in the CEs of the sites) and the site administrators. It is up to the experiments and the sites themselves to decide to what degree they will cooperate, and how exactly they desire to perform the software installation.

In [R29], a possible way for software installation is given. This is the approach that will be described here. For more information, please refer to [R29], and to the man pages of the commands cited in the section.

A.1. GENERAL PROCEDURE

The Experiment Software Manager (ESM) is the person from a experiment responsible for the installation of new software in the different sites. The ESM must have privileges to install software in a particular area of the CEs. There is dedicated space for each one of the supported VOs. There is an environmental variable that holds the path to that space. Its format is the following:

`$VO_<name_of_VO>_SW_DIR`

If a site uses a shared file system for the WNs of its CEs, then the software is installed once in the filesystem, and the jobs that want to use it just need to use the `$VO_<name_of_VO>_SW_DIR` variable to access the software.

If, on the contrary, the site does not use a shared file system, the software is stored in a SE in the site, and its only installed in a WN when a job requires it. When the job gets to the WN, it must check the `$VO_<name_of_VO>_SW_DIR` variable, and if it only contains a ".", instead of a valid path, then it must get the software from a SE and install it in the WN. Then, the job is executed. Finally, the software is erased from the WN. This is done to avoid the exhaustion of disk space in the node.

The `lcg-ManageSoftware` script can be used to perform the installation of the software.

Finally, the software must be published in the Information System, so that the jobs requesting for a particular piece of software can be directed to the appropriate CE. For that purpose, a tag is added to the `GlueHostApplicationSoftwareRunTimeEnvironment` atribute of the IS, using the GRIS running in the CE.

The `lcg-ManageVO` can be used to manage this tag.

[9] What this section says for the *experiments* can be extended to the more general concept of *VOs*

A.2. LCG-MANAGESOFTWARE

The ESM must prepare a tarball containing the software to be installed and some scripts to install, validate, run and unistall the software. This tarball has to be uploaded (`edg-rm cr` command) to a SE of the site where the software will be installed.

Then, a JDL file that will install the software must be written. It will have the `lcg-ManageSoftware` script as executable, the tarball and any other needed files as input data, and the actions to be performed by the script as arguments. The first time, this actions will be installation, validation and publication of the software. Depending on the presence of a shared file system (advertised by the `$VO_<name_of_VO>_SW_DIR` variable), the software is installed in a shared directory, or in a particular WN. Only when the software has been installed and validated, it is published in the IS (this is done automatically by the script).

Once the software is published for a CE, new jobs requiring the software can be scheduled to it. If a shared file system is present, no other action than running the software is needed. Otherwise, every job must install the software before running it, and must uninstall it afterwards. No validation or new publication is needed, since that has been already done (as the publishing tag is present). For this tasks, the `lcg-ManageSoftware` script is used again.

For information on the different options available for the `lcg-ManageSoftware` command, plese check its manpage.

A.3. LCG-MANAGEVO

The `lcg-ManageVOTag` command can be used to manage the contents of the `GlueHostApplicationSoftwareRunTimeEnvironment` attribute of the IS, for a particular CE. Its different options can be used to add a new tag for installed software, show the list of published tags for a given VO, remove a defined tag, or clean everything that had been previously published.

This tool can be handy in several situations, such as when an error in the information published has to be corrected, or when the tag for some software needs to be manually added. Notice that this will not be the case if the `lcg-ManageSoftware` command was used, since that tool automatically publishes the information tag for an installed and validated software. However, if the ESM chooses to use another method for the installation (e.g.: asking the site administrator to manually install the software), then he/she must make sure that the software is correctly published in the IS; and for that he can use the `lcg-ManageVOTag` command.

Please refer to the manpage of the command for details on the syntax of the command.

APPENDIX B EDG-REPLICA-MANAGER COMMAND

It has been already explained that the use of the LCG Data Management tools is preferred to that of the `edg-rm` command. Nevertheless, this sections gives information on its functionalities (more or less in the same fashion that was done for the `lcg-*` commands in Chapter 6).

The `edg-rm` command allows users to copy files between UI, CE, WN and a SE, to register entries in the RLS and replicate files between SEs. More information on its use can be found in [R30]. In addition, more detailed examples on Replica Manager usage can be found in [R31].

The general form of a `edg-rm` invocation is the following:

```
$ edg-rm <general_options> <cmd_name> <cmd_arguments> <cmd_options>
```

where the `<general_options>` refer to `edg-rm`, `<cmd_name>` is the particular command or action that the RM must perform, and `<cmd_arguments>` and `<cmd_options>` refer to that command. Most commands have both an extended and an abbreviated name form.

NOTE: If the above described order is not followed (general options before the command name, and particular options after it) the general and command-specific options may be mixed, resulting in a fail of the command.

The `--vo <vo_name>` option specifies the virtual organization of the user. This option is mandatory — without it the command will not work. Other general `edg-rm` options are: `--log-debug`, `--log-info` and `--log-off`, which are used for enabling or disabling bug-level or info-level logging; and the `--config <file>` option, which is used to read the specified configuration file, instead of the default `$EDG_LOCATION/etc/edg-replica-manager/edg-replica-manager.conf`.

In what follows some usage examples are given. For details on the options of each command, please use the `--help` option with `edg-rm`. If the name of a command is also given, then specific information about that command is presented. The user can also consult the manpages and [R30].

For clarity reasons, in the pieces of code that follow (throghout the whole chapter), the commands introduced by the user are leaded by a '$' symbol, and the answers of the shell are usually preceded by '>' (unless the difference is obvious).

B.0.1. Basic Replica Manager Commands

Example B.0.1.1 (Uploading a file from the UI to the Grid)

In order to upload a file to the Grid, i.e., to transfer it from the local machine to a Storage Element where it must reside permanently, the `CopyAndRegister` (`cr`) command can be used (in a machine with a valid proxy):

```
$ edg-rm --vo dteam cr file:///home/antonio/file1 -l lfn:my_alias1
> guid:6ac491ea-684c-11d8-8f12-9c97cebf582a
```

where the only argument is the local file to be uploaded (a fully qualified URI) and the `-l` option indicates an LFN for it. The command returns the unique GUID for the file. If no LFN is provided, then the returned GUID will be the only way to access the file in the Grid.

If the `-d <destination>` option is included, then the specified SE (which must be known in advance) is used as the destination for the file. Without the `-d` option, a default SE is chosen automatically. A complete SURL, including the SE hostname, the path (accesspoint plus VO-specific directory) and a chosen filename, or only the SE hostname can be used as the destination. This is illustrated by the following commands:

```
$ edg-rm --vo dteam cr file:/home/antonio/file1 -l lfn:my_alias1 -d tbed0115.cern.ch
```

or

```
$ edg-rm --vo dteam cr file:/home/antonio/file1 -l lfn:my_alias1 \
 -d sfn://tbed0115.cern.ch/dteam/my_file1
```

In this and other commands, the `-p <protocol>` and `-n <#streams>` options can be used to specify the protocol (`gsiftp` being the default) and the number of parallel streams to be used in the transfer (default is 8).

Example B.0.1.2 (Retrieving information about the Grid)

If the above described `-d` option is to be used, then the information about the available SEs must be retrieved in advance. The EDG Replica Manager `printInfo` (`pi`) command can be used to that goal:

```
$ edg-rm --vo dteam printInfo
```

The previous command returns all CEs and SEs that the Replica Manager retrieves from the IS, as well as the RMC and LRC used in the specified VO. For every CE, the close SEs are also given. Regarding the SEs, the VOs and protocols supported, and their accesspoint are provided.

A typical output is as follows:

```
VO used            : cms
default SE         : tbed0101.cern.ch
default CE         : pceis01.cern.ch
Info Service       : MDS

RMC endpoint : http://rlscert01.cern.ch:7777/dteam/v2.2/edg-replica-metadata-catalog/
               services/edg-replica-metadata-catalog
LRC endpoint : http://rlscert01.cern.ch:7777/edg-replica-location/services/
```

```
               edg-local-replica-catalog
ROS endpoint : no information found: No Service found edg-replica-optimization

List of CE ID's: pceis01.cern.ch:2119/jobmanager-pbs-infinite
                 pceis01.cern.ch:2119/jobmanager-pbs-long
                 pceis01.cern.ch:2119/jobmanager-pbs-medium
                 pceis01.cern.ch:2119/jobmanager-pbs-short
[...]

CE at infinite :
               name : infinite
               ID: pceis01.cern.ch:2119/jobmanager-pbs-infinite
               closeSEs : cmslcgse02.cern.ch,lcgse02.ifae.es,tbed0101.cern.ch,
                          tbed0115.cern.ch, wacdr002d.cern.ch
               VOs : alice,atlas,cms,lhcb,dteam
[...]

List of SE ID's : tbed0101.cern.ch
                  tbed0115.cern.ch
                  wacdr002d.cern.ch
                  cmslcgse02.cern.ch
                  lcgse02.ifae.es

SE at eis :
      name : eis
      host : tbed0101.cern.ch
      type : disk
      accesspoint : /flatfile/SE00
      VOs : alice,atlas,cms,dteam,lhcb
      VO directories : alice:/alice,atlas:/atlas,cms:/cms,dteam:/dteam,lhcb:/lhcb
      protocols : gsiftp,rfio
[...]
```

In order to find all SEs, their access point and the VO directories the user can filter the previous response with `grep`, as in the following example, where the desired information is specified with the `-e` option and, just to get a nicer output, unwanted lines are eliminated by using the `-v` option.

```
$ edg-rm --vo=dteam pi | grep -e SE -e host -e accesspoint \
        -e 'VO directories' | grep -v closeSEs | grep -v "List of SE"

default SE : tbed0101.cern.ch

SE at eis :
            host : tbed0101.cern.ch
            accesspoint : /flatfile/SE00
            VO directories : alice:/alice,atlas:/atlas,cms:/cms,dteam:/dteam,lhcb:/lhcb
SE at eis :
            host : tbed0115.cern.ch
            accesspoint : /
            VO directories : alice:alice,atlas:atlas,cms:cms,dteam:dteam,lhcb:lhcb
```

```
SE at CERN-LCG2 :
            host : wacdr002d.cern.ch
            accesspoint : /castor/cern.ch/grid
            VO directories : alice:alice,atlas:atlas,cms:cms,dteam:dteam,lhcb:lhcb
SE at eis :
            host : cmslcgse02.cern.ch
            accesspoint : /data1/lcg
            VO directories : cms:cms
SE at PIC-LCG2 :
            host : lcgse02.ifae.es
            accesspoint : /castor/ifae.es/lcg
            VO directories : atlas:atlas,cms:cms,dteam:dteam,lhcb:lhcb
```

The `printInfo` command does not return the free space on the SE.

Example B.0.1.3 (Replicating a file)

Once a file is stored on an SE and registered with the Replica Location Service, the file can be replicated using the `replicateFile` (`rep`) command, as in:

```
$ edg-rm --vo=dteam replicateFile guid:6ac491ea-684c-11d8-8f12-9c97cebf582a \
           -d wacdr002d.cern.ch
> sfn://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/generated/2004-02-26/
filea778c4f6-687d-11d8-a111-c2fed1a6363a
```

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 (and, as with `CopyAndRegisterFile`, using either the SE hostname or a complete SURL). If this option is not set, then an SE is chosen automatically.

For one GUID, there can be only one replica per SE. If the user tries to use the `replicateFile` command with a destination SE that already holds a replica, the existing SURL will be returned, and no new replica will be created.

Example B.0.1.4 (Listing replicas and GUIDs)

The Replica Manager allows users to list all the replicas of a file that have been successfully registered with the Replica Location Service. For that purpose the `listReplicas` (`lr`) command is used:

```
$ edg-rm --vo=dteam lr lfn:my_alias1
> sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/
filea72eaedc-684b-11d8-8efc-fc10ad029740
> sfn://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/generated/2004-02-26/
filea778c4f6-687d-11d8-a111-c2fed1a6363a
```

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.

Reciprocally, the `listGUID` (`lg`) return the GUID associated with a specified LFN or SURL:

```
$ edg-rm --vo=dteam lg sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_file1
> guid:c06a92ee-6911-11d8-a453-d9c1af867039
```

Example B.0.1.5 (Copying files out of the Grid)

The `copyFile` (`cp`) command can be used to copy a Grid file to a non-grid storage resource. This is useful to have a local copy of the file. The command accepts the LFN, GUID or SURL of the LCG-2 file as its first argument and a local filename or valid TURL as the second, as is shown in the following example:

```
$ edg-rm --vo dteam cp lfn:my_alias2 file:/home/antonio/file2
```

Note that although this command is designed to copy files from a SE to a non-grid resources, if the proper TURL is used, 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 `replicateFile` but **skipping the file registration**, making in this way this replica invisible to Grid users.

Example B.0.1.6 (Obtaining a TURL for a replica)

For any given replica (identified by its SURL) the TURL for accessing it using a particular protocol can be obtained with the `getTurl` (`gt`) command. The arguments are the SURL of the file and the protocol to be used. The command returns the valid TURL or an error message if the specified protocol is not supported by that SE for the given replica.

```
$ edg-rm --vo dteam getTurl \
sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1 gsiftp
> gsiftp://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1

$ edg-rm --vo dteam getTurl \
sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1 ftp
> The file sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1
is not accessible via the protocol: ftp
```

Example B.0.1.7 (Deleting replicas)

Once a file is stored on a Storage Element and registered with a catalog, it can be deleted using the `deleteFile` (`del`) command. If a SURL is provided as argument, then that particular replica will be

deleted. If a LFN is given instead, then the `-s <SE>` option must be used to indicate which one of the replicas must be erased. The same is true if a GUID is specified, unless the `--all-available` option is used, in which case all replicas of the file will be deleted and unregistered (on a best-effort basis).

The following commands:

```
$ edg-rm --vo=dteam del guid:adb8e950-bf7e-11d7-a29c-fbbda1b7a6d1 -s wacdr002d.cern.ch
```

and

```
$ edg-rm --vo=dteam del guid:adb8e950-bf7e-11d7-a29c-fbbda1b7a6d1 --all-available
```

remove, from the file system and the catalog, one particular replica and all available replicas of the file, respectively.

B.0.2. Other Commands

Example B.0.2.1 (Registering and unregistering Grid files)

The `registerFile` command creates a new GUID for a given SURL, whereas `registerGUID` associates the replica identified by a SURL with an existent GUID (also specified as an argument). In the second case, it is assumed that there exist some other replicas of the files, which are already registered.

An example of the commands usage follows:

```
$ edg-rm --vo dteam rf sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_file1
> guid:c06a92ee-6911-11d8-a453-d9c1af867039

$ edg-rm --vo dteam rg sfn://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/my_file3 \
guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30
```

The `unregisterFile` (`uf`) command unregisters a replica from the LRC catalogue, without actually deleting it physically from the disk. The user must specify both the GUID and the SURL to be unregistered, as in:

```
$ edg-rm -i --vo=dteam unregisterFile guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30 \
sfn://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/my_test3
```

If the last replica of a file is unregistered, then the GUID is also removed from the catalogue.

Example B.0.2.2 (Managing aliases)

The `addAlias` (`aa`) command allows the user to add a new LFN to an existing GUID:

```
$ edg-rm --vo=dteam addAlias guid:c06a92ee-6911-11d8-a453-d9c1af867039 lfn:last_results
```

The `removeAlias` (`ra`) command allows the user to remove an LFN from an existing GUID:

```
$ edg-rm --vo=dteam ra guid:c06a92ee-6911-11d8-a453-d9c1af867039 lfn:last_results
```

The `edg-rm` command cannot be used to list the aliases of a file.

Example B.0.2.3 (Listing an SE directory)

The `list` (`ls`) command can be used to list the contents of an SE directory.

```
$ edg-rm --vo dteam ls sfn://tbed0101.cern.ch/flatfile/SE00/dteam
> my_test1
> generated
> output.txt
> POOL-RM.txt
```

The argument of the command is a URI where the schema can be `sfn`, `srm`, or `gsiftp`.

APPENDIX C THE GLUE SCHEMA

As explained earlier, the GLUE Schema describes the Grid resources information that is stored for its use 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 [R12].

First of all, the tree of object classes definitions is shown. Then, the attributes for each one of the objectclasses (where the dynamique data is actually stored) are presented. Please, notice that 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.

C.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
        |     |
        |     ----- .2. Attributes
        |           |
        |            ----- .1. Attributes for GlueSchemaVersion
        |                              . . .
        |           |
        |            ----- .5. Attributes for GlueInformationService
        |
        ----- .2. GlueCETop
        |     |
        |     ----- .1. ObjectClass
        |     |     |
        |     |     ----- .1  GlueCE
        |     |     |
```

```
|     |     ----- .2  GlueCEInfo
|     |     |
|     |     ----- .3  GlueCEState
|     |     |
|     |     ----- .4  GlueCEPolicy
|     |     |
|     |     ----- .5  GlueCEAccessControlBase
|     |     |
|     |     ----- .6  GlueCEJob
|     |
|     ----- .2. Attributes
|     |     |
|     |     ----- .1.  Attributes for GlueCE
|     |                      . . .
|     |     |
|     |     ----- .6.  Attributes for GlueCEJob
|     |
|     ----- .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
|     |
|     ----- .2. Attributes
|     |     |
|     |     ----- .1. Attributes for GlueCluster
|     |                     . . .
|     |     |
|     |     ----- .16  Attributes for GlueHostFile
|     |
|     ----- .3. MyObjectClass
|     |
|     ----- .4. MyAttributes
|
----- .4. GlueSETop
|     |
|     ----- .1. ObjectClass
|     |     |
|     |     ----- .1  GlueSE
|     |     |
|     |     ----- .2  GlueSEState
|     |     |
|     |     ----- .3  GlueSEAccessProtocol
|     |
|     ----- .2. Attributes
|     |     |
|     |     ----- .1.  Attributes for GlueSE
|     |                     . . .
|     |     |
|     |     ----- .3.  Attributes for GlueSEAccessProtocol
|     |
|     ----- .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
```

C.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... (objectclass `GlueGeneralTop`)**
  - No attributes
- **Schema Version Number (objectclass `GlueSchemaVersion`)**
  - `GlueSchemaVersionMajor`: Major Schema version number
  - `GlueSchemaVersionMinor`: Minor Schema version number
- **Access to the Information Service (objectclass `GlueInformationService`)**
  - `GlueInformationServiceURL`: The Information Service URL publishing the information of this entry
- **Internal atributes 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

C.3. ATTRIBUTES FOR THE COMPUTING ELEMENT

These are atributes 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
- **CE (objectclass `GlueCE`)**
  - `GlueCEUniqueID`: unique identifier for the CE

  - `GlueCEName`: human-readable name of the service

- **Info (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
- **State (objectclass `GlueCEState`)**
  - `GlueCEStateRunningJobs`: number of running jobs
  - `GlueCEStateWaitingJobs`: number of jobs not running
  - `GlueCEStateTotalJobs`: total number of jobs (running + waiting)
  - `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)
  - `GlueCEStateWorstResponseTime`: worst possible time between the submission of a job and the start of its execution
  - `GlueCEStateEstimatedResponseTime`: estimated time between the submission of a job and the start of its execution
  - `GlueCEStateFreeCPUs`: number of CPUs available to the scheduler
- **Policy (objectclass `GlueCEPolicy`)**
  - `GlueCEPolicyMaxWallClockTime`: maximum wall clock time available to jobs submitted to the CE, in seconds (previously it was in minutes)
  - `GlueCEPolicyMaxCPUTime`: maximum CPU time available to jobs submitted to the CE, in seconds (previously it was 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
- **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
- **Subcluster (objectclass `GlueSubCluster`)**
  - GlueSubClusterUniqueID: unique identifier for the subcluster
  - GlueSubClusterName: human-readable name of the subcluster
- **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
- **Processor (objectclass `GlueHostProcessor`)**
  - GlueHostProcessorVendor: name of the CPU vendor
  - GlueHostProcessorModel: name of the CPU model
  - GlueHostProcessorVersion: version of the CPU
  - GlueHostProcessorOtherProcessorDescription: other description for the CPU
  - GlueHostProcessorClockSpeed: clock speed of the CPU
  - GlueHostProcessorInstructionSet: name of the instruction set architecture of the CPU
  - GlueHostProcessorGlueHostProcessorFeatures: list of optional features of 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 id 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

C.4. ATTRIBUTES FOR THE STORAGE ELEMENT

These are atributes that give information about a SE and the storage space that it holds. In the GLUE Schema, they are defined in the UML diagram for the Storage Element.

It is probably 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
- **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
  - `GlueSEAccessProtocolAccessTime`: time to access a file using this protocol
  - `GlueSEAccessProtocolSupportedSecurity`: security features supported by the 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
  - `GlueSLLocalFileSystemServer`: unique identifier of the server exporting this file system (only for remote file systems)
- **Remote File system (objectclass `GlueSLRemoteFileSystem`)**
  - `GlueSLRemoteFileSystemRoot`: path name (or other information) defining the root of the file system
  - `GlueSLRemoteFileSystemName`: name of the file system
  - `GlueSLRemoteFileSystemType`: file system type (e.g. NFS, AFS, etc.)
  - `GlueSLRemoteFileSystemReadOnly`: true is the file system is read-only
  - `GlueSLRemoteFileSystemSize`: total space assigned to this file system
  - `GlueSLRemoteFileSystemAvailableSpace`: total free space in this 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
- **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

C.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

C.6. THE DIT USED BY THE MDS

The DITs used in the GRISes of a CE and a SE are shown in the Figures 10, 11 and 12. 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 site GIIS, grouping all the information of the Grid resources in that site. The information of the different GIISes is compiled in a BDII, as described in section 7.3.

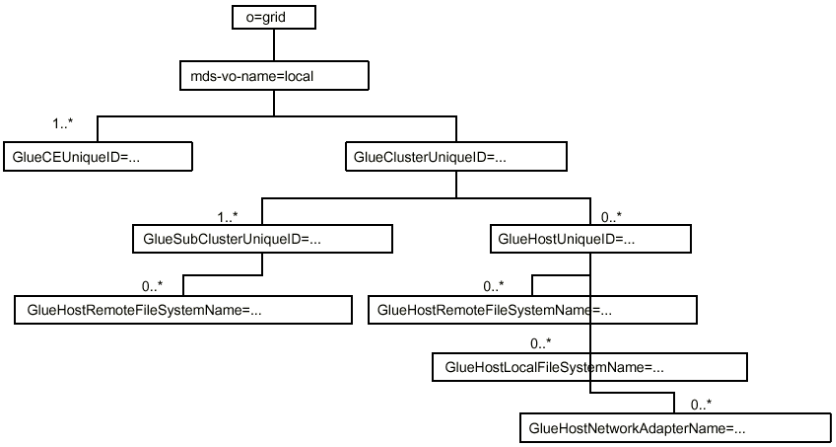

Figure 10: DIT for the computing resources

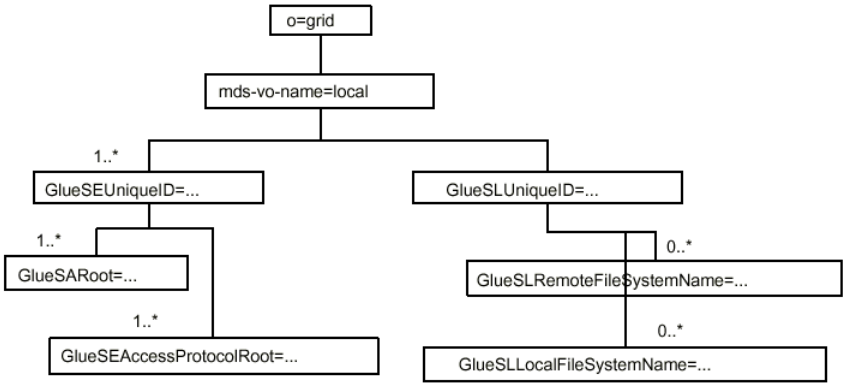

Figure 11: DIT for the storage resources

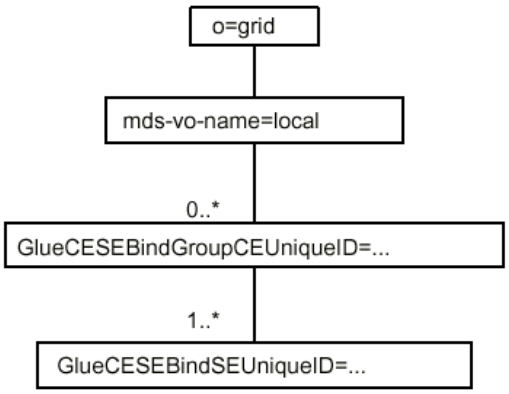

Figure 12: DIT for the computing-storage service relationship

APPENDIX D THE GRID MIDDLEWARE

The Grid middleware deployed in the LCG-2 service is reported below.

The operating system for the Computing Elements is Linux Red Hat 7.3, mainly running on IA32 computers.

The LCG-2 middleware layer uses components from ***EDT (European DataTag)*** 1.1, ***EDG (European DataGrid)*** 2.1 and VDT ***(Virtual Data Toolkit)*** 1.1.14.

Because only some components and not packages as a whole are selected, different components of one package may be selected from different versions of that package. Also, notice that, in some cases, LCG patches are applied to the components, so the final software used is not exactly the same as the initially distributed in the packages.

In the following we list the components from these packages/suites, which are currently used in LCG-2. Please, notice that only major and minor version numbers are provided. Patches numbers are not included, as they change often.

- EDG 2.1
  - EDG-WMS Workload Management System
  - Data Management System
    - EDG-RM (Replica Manager)
    - EDG-RLS (Replica Location Service)
      - Including the EDG-LRC (Local Replica Catalog)
    - EDG-RMC (Replica Metadata Catalog)
  - Fabric Management
    - EDG WP4 tools/procedures (LCFG, LCFG-Lite or manual procedures)
    - LCAS/LCMAPS (Local Centre Auth. System and Local Mapping)
  - Virtual Organization Management
    - EDG infrastructure and procedures
  - Information service
    - Information index BDII
- EDT 1.1
  - Monitoring system
    - Grid-ICE
  - Glue Schema LCG-EDT 1.1

- VDT 1.1.14
  - Core components:
    - Globus 2.4.3
    - Condor 6.6.0
      - Condor-G
      - ClassAds
- Information Service
  - Globus MDS
    - GRIS (Grid Resource Information Service)
    - GIIS (Grid Index Information Service)

APPENDIX E JOB STATUS DEFINITION

As already mentioned in chapter 5, a job can find itself in one of several possible states, the definition of which 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.)
CANCELLED	The job has been cancelled by the user
CLEARED	The Output Sandbox has been transferred to the User Interface

Only some transitions between states are allowed. These transitions are depicted in Figure 13.

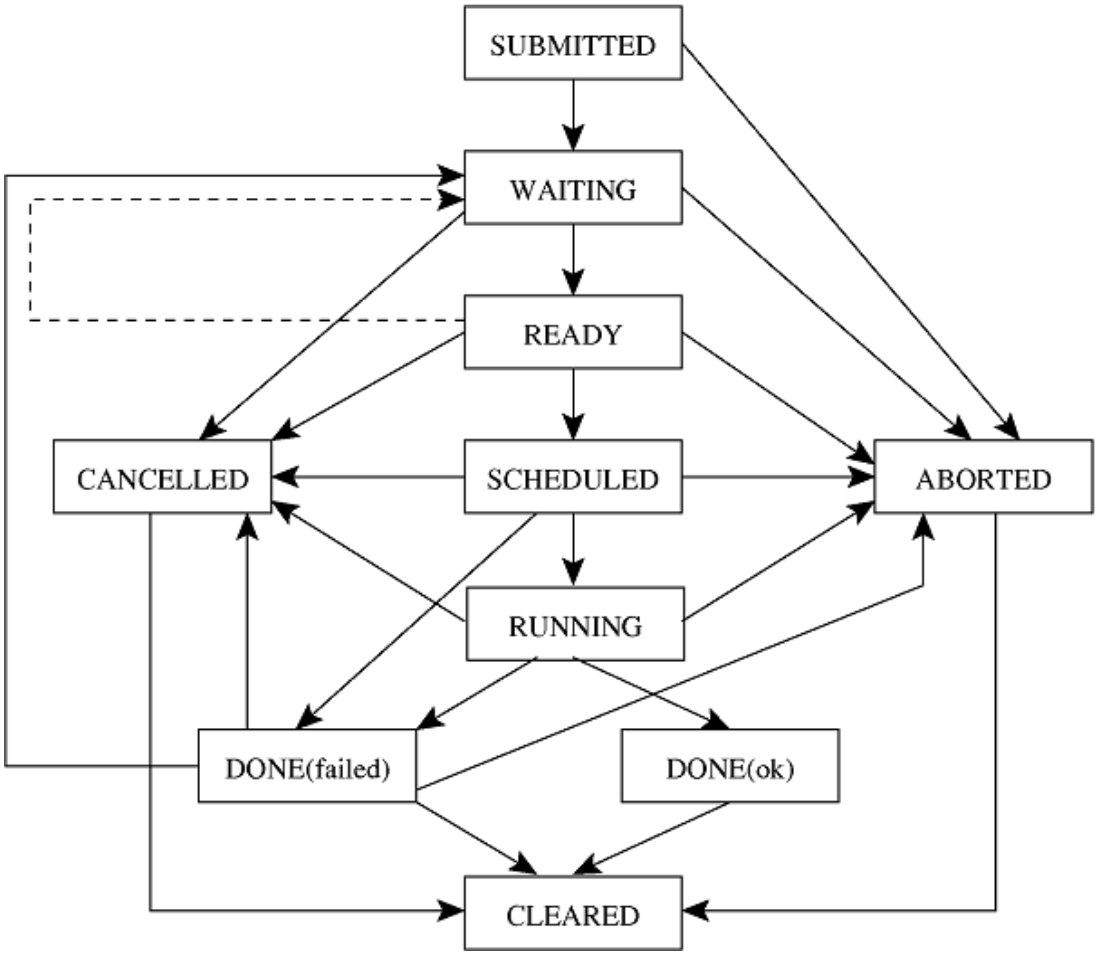

Figure 13: Possible job states in the LCG-2