



# LHC COMPUTING GRID

## LCG-2 USER GUIDE

### MANUALS SERIES

---

**Document identifier:** CERN-LCG-GDEIS-454439

**EDMS id:** 454439

**Version:** 2.3

**Date:** August 4, 2005

**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.  |
| 14/04/05 | v2.2   | SRM, dCache, DPM. LFC. R-GMA. lcg_util API. MPI. Exp SW installation. Submission framework... Corrections.   |
| 10/05/05 | v2.2.2 | Minor corrections. Experiments SW installation enhancements. Reordering.   |
| 04/08/05 | v2.3   | New: User and VO utilities, DLI, Updated: lcg_util (time outs), GLUE schema. Enhanced: MPI, R-GMA. Removed (appendices): exp SW, job mgm., edg-rm. |

## Files

| Software Products | User files  |
|-------------------|---|
| PDF               | <a href="https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf">https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf</a>   |
| PS                | <a href="https://edms.cern.ch/file/454439//LCG-2-UserGuide.ps">https://edms.cern.ch/file/454439//LCG-2-UserGuide.ps</a>     |
| HTML              | <a href="https://edms.cern.ch/file/454439//LCG-2-UserGuide.html">https://edms.cern.ch/file/454439//LCG-2-UserGuide.html</a> |



---

# CONTENTS

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>INTRODUCTION.....</b>                    | <b>8</b>  |
| 1.1      | ACKNOWLEDGMENTS .....                       | 8         |
| 1.2      | OBJECTIVES OF THIS DOCUMENT .....           | 8         |
| 1.3      | APPLICATION AREA.....                       | 8         |
| 1.4      | DOCUMENT EVOLUTION PROCEDURE .....          | 8         |
| 1.5      | REFERENCE AND APPLICABLE DOCUMENTS .....    | 8         |
| 1.6      | TERMINOLOGY .....                           | 11        |
| 1.6.1    | Glossary .....                              | 11        |
| <b>2</b> | <b>EXECUTIVE SUMMARY.....</b>               | <b>14</b> |
| <b>3</b> | <b>OVERVIEW .....</b>                       | <b>15</b> |
| 3.1      | PRELIMINARY MATTERS.....                    | 16        |
| 3.1.1    | Code Development .....                      | 16        |
| 3.1.2    | Troubleshooting .....                       | 16        |
| 3.1.3    | User and VO utilities .....                 | 17        |
| 3.2      | THE LCG-2 ARCHITECTURE.....                 | 17        |
| 3.2.1    | Security .....                              | 17        |
| 3.2.2    | The User Interface .....                    | 18        |
| 3.2.3    | Computing Element and Storage Element ..... | 18        |
| 3.2.4    | Information Service .....                   | 20        |
| 3.2.5    | Data Management .....                       | 25        |
| 3.2.6    | Job Management .....                        | 27        |



---

|          |   |           |
|----------|---|-----------|
| 3.3      | JOB FLOW .....                                | 27        |
| 3.3.1    | Job Submission .....                          | 28        |
| 3.3.2    | Other operations .....                        | 29        |
| <b>4</b> | <b>GETTING STARTED .....</b>                  | <b>31</b> |
| 4.1      | OBTAINING A CERTIFICATE .....                 | 31        |
| 4.1.1    | X.509 Certificates .....                      | 31        |
| 4.1.2    | Generating a request .....                    | 32        |
| 4.1.3    | Getting the Certificate .....                 | 32        |
| 4.1.4    | Renewing the Certificate .....                | 33        |
| 4.2      | REGISTERING WITH LCG-2.....                   | 33        |
| 4.2.1    | The Registration Service .....                | 33        |
| 4.2.2    | Virtual Organisations .....                   | 34        |
| 4.3      | SETTING UP THE USER ACCOUNT.....              | 35        |
| 4.3.1    | The User Interface .....                      | 35        |
| 4.3.2    | Checking a Certificate .....                  | 35        |
| 4.4      | PROXY CERTIFICATES .....                      | 37        |
| 4.4.1    | Proxy Certificates .....                      | 37        |
| 4.4.2    | Virtual Organisation Membership Service ..... | 39        |
| 4.4.3    | Advanced Proxy Management .....               | 40        |
| 4.5      | THE LCG GRID OPERATIONS CENTRE.....           | 42        |
| <b>5</b> | <b>INFORMATION SERVICE.....</b>               | <b>44</b> |
| 5.1      | THE MDS .....                                 | 44        |
| 5.1.1    | lcg-infosites .....                           | 44        |
| 5.1.2    | lcg-info .....                                | 48        |



---

|          |                                       |           |
|----------|---------------------------------------|-----------|
| 5.1.3    | The Local GRIS . . . . .              | 50        |
| 5.1.4    | The Site GIIS/BDII . . . . .          | 54        |
| 5.1.5    | The top BDII . . . . .                | 56        |
| 5.2      | R-GMA . . . . .                       | 59        |
| 5.2.1    | R-GMA Browser . . . . .               | 59        |
| 5.2.2    | R-GMA CLI . . . . .                   | 59        |
| 5.2.3    | R-GMA API . . . . .                   | 63        |
| 5.3      | MONITORING . . . . .                  | 63        |
| 5.3.1    | GridIce . . . . .                     | 64        |
| <b>6</b> | <b>WORKLOAD MANAGEMENT . . . . .</b>  | <b>65</b> |
| 6.1      | JOB DESCRIPTION LANGUAGE . . . . .    | 65        |
| 6.2      | THE COMMAND LINE INTERFACE . . . . .  | 69        |
| 6.2.1    | Job Submission . . . . .              | 70        |
| 6.2.2    | Job Operations . . . . .              | 73        |
| 6.2.3    | The BrokerInfo . . . . .              | 77        |
| 6.2.4    | Interactive Jobs . . . . .            | 78        |
| 6.2.5    | Checkpointable Jobs . . . . .         | 83        |
| 6.2.6    | MPI Jobs . . . . .                    | 84        |
| 6.2.7    | Advanced Command Options . . . . .    | 88        |
| 6.3      | THE GRAPHICAL USER INTERFACE. . . . . | 89        |
| <b>7</b> | <b>DATA MANAGEMENT. . . . .</b>       | <b>91</b> |
| 7.1      | INTRODUCTION . . . . .                | 91        |
| 7.2      | STORAGE ELEMENTS. . . . .             | 91        |
| 7.2.1    | Data Channel Protocols . . . . .      | 91        |



---

|          |  |            |
|----------|--|------------|
| 7.2.2    | The Storage Resource Manager interface . . . . .               | 92         |
| 7.2.3    | Types of Storage Elements . . . . .                            | 93         |
| 7.3      | FILES NAMING CONVENTION IN LCG-2 . . . . .                     | 94         |
| 7.4      | FILE CATALOGS IN LCG-2 . . . . .                               | 95         |
| 7.5      | LFC INTERACTION COMMANDS . . . . .                             | 96         |
| 7.6      | RLS INTERACTION COMMANDS . . . . .                             | 100        |
| 7.6.1    | Local Replica Catalog Commands . . . . .                       | 101        |
| 7.6.2    | Replica Metadata Catalog Commands . . . . .                    | 105        |
| 7.7      | FILE AND REPLICA MANAGEMENT CLIENT TOOLS . . . . .             | 108        |
| 7.7.1    | LCG Data Management Client Tools . . . . .                     | 108        |
| 7.7.2    | Low Level Data Management Tools . . . . .                      | 116        |
| 7.8      | JOB SERVICES AND DATA MANAGEMENT . . . . .                     | 118        |
| 7.9      | ACCESSING GRID FILES FROM A JOB . . . . .                      | 120        |
| 7.10     | POOL AND LCG-2 . . . . .                                       | 120        |
| <b>A</b> | <b>THE GRID MIDDLEWARE . . . . .</b>                           | <b>123</b> |
| <b>B</b> | <b>CONFIGURATION FILES AND VARIABLES . . . . .</b>             | <b>125</b> |
| <b>C</b> | <b>JOB STATUS DEFINITION . . . . .</b>                         | <b>127</b> |
| <b>D</b> | <b>USER TOOLS . . . . .</b>                                    | <b>129</b> |
| D.1      | INTRODUCTION . . . . .   | 129        |
| D.2      | JOB MANAGEMENT FRAMEWORK . . . . .                             | 129        |
| D.3      | JOB MONITORING (LCG-JOB-MONITOR) . . . . .                     | 130        |
| D.4      | JOB STATUS MONITORING (LCG-JOB-STATUS) . . . . .               | 130        |
| D.5      | TIME LEFT UTILITY (LCG-GETJOBSTATS, LCG_JOBSTATS.PY) . . . . . | 130        |



---

|          |  |            |
|----------|--|------------|
| D.6      | INFORMATION SYSTEM READER (LCG-INFO).....  | 131        |
| <b>E</b> | <b>VO-WIDE UTILITIES.....</b>  | <b>133</b> |
| E.1      | INTRODUCTION.....  | 133        |
| E.2      | FREEDOM OF CHOICE FOR RESOURCES.....   | 133        |
| E.3      | THE VO BOX.....  | 133        |
| E.4      | EXPERIMENTS SOFTWARE INSTALLATION.....   | 134        |
| <b>F</b> | <b>DATA MANAGEMENT AND FILE ACCESS THROUGH AN APPLICATION PRO-<br/>GRAMMING INTERFACE.....</b> | <b>135</b> |
| <b>G</b> | <b>THE GLUE SCHEMA.....</b>  | <b>148</b> |
| G.1      | THE GLUE SCHEMA LDAP OBJECT CLASSES TREE.....  | 148        |
| G.2      | GENERAL ATTRIBUTES.....  | 152        |
| G.3      | ATTRIBUTES FOR THE COMPUTING ELEMENT.....  | 153        |
| G.4      | ATTRIBUTES FOR THE STORAGE ELEMENT.....  | 158        |
| G.5      | ATTRIBUTES FOR THE CE-SE BINDING.....  | 162        |
| G.6      | THE DIT USED BY THE MDS.....   | 162        |



# 1. INTRODUCTION

## 1.1. ACKNOWLEDGMENTS

This work received support from the following institutions:

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

## 1.2. OBJECTIVES OF THIS DOCUMENT

This document gives an overview of the main characteristics of the LCG-2 middleware, which is being used for EGEE. 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 EGEE who would like to work with the LCG-2 Grid middleware.

## 1.4. DOCUMENT EVOLUTION PROCEDURE

The guide reflects the current status of the LCG-2 middleware, 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 software are made.

## 1.5. REFERENCE AND APPLICABLE DOCUMENTS

# REFERENCES

- [R1] Enabling Grids for E-science in Europe  
<http://eu-egee.org>





- 
- [R2] LHC Computing Grid Project  
<http://lcg.web.cern.ch/LCG/>
  
  - [R3] The Anatomy of the Grid.  
Enabling Scalable Virtual Organizations  
Ian Foster, Carl Kesselman, Steven Tuecke  
<http://www.globus.org/research/papers/anatomy.pdf>
  
  - [R4] Requirements for LCG User Registration and VO Membership Management  
[https://edms.cern.ch/file/428034//LCG\\_User\\_Registration.pdf](https://edms.cern.ch/file/428034//LCG_User_Registration.pdf)
  
  - [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] Experiment Software Installation  
<https://edms.cern.ch/file/498080/1.0/SoftwareInstallation.pdf>
  
  - [R8] Overview of the Grid Security Infrastructure  
<http://www-unix.globus.org/security/overview.html>
  
  - [R9] European DataGrid Project  
<http://eu-datagrid.web.cern.ch/eu-datagrid/>
  
  - [R10] LCG-2 Middleware overview  
<https://edms.cern.ch/file/498079//LCG-mw.pdf>
  
  - [R11] The Storage Resource Manager  
<http://sdm.lbl.gov/srm-wg/>
  
  - [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] R-GMA: Relational Grid Monitoring Architecture  
<http://www.r-gma.org/index.html>
  
  - [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](http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf)
  
  - [R16] Using lxplus as an LCG2 User Interface  
<http://grid-deployment.web.cern.ch/grid-deployment/documentation/UI-lxplus/>
  
  - [R17] LCG-2 Manual Installation Guide  
<https://edms.cern.ch/file/434070//LCG2Install.pdf>



- 
- [R18] GridICE: a monitoring service for the Grid  
<http://server11.infn.it/gridice/>
  - [R19] Classified Advertisements. Condor.  
<http://www.cs.wisc.edu/condor/classad>
  - [R20] The Condor Project.  
<http://www.cs.wisc.edu/condor/>
  - [R21] Job Description language HowTo. December 17th, 2001  
[http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0\\_2-Document.pdf](http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf)
  - [R22] JDL Attributes – Release 2.x. Oct 28th, 2003  
[http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0\\_2.pdf](http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf)
  - [R23] 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>
  - [R24] Workload Management Software – GUI User Guide. Nov 24th, 2003  
[http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0143-0\\_0.pdf](http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0143-0_0.pdf)
  - [R25] GSIFTP Tools for the Data Grid  
<http://www.globus.org/datagrid/deliverables/gsiftp-tools.html>
  - [R26] RFIO: Remote File Input/Output  
<http://doc.in2p3.fr/doc/public/products/rfio/rfio.html>
  - [R27] CASTOR  
<http://castor.web.cern.ch/castor/>
  - [R28] dCache  
<http://www.dCache.org>
  - [R29] 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>
  - [R30] 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>
  - [R31] 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>
  - [R32] The edg-replica-manager Wrapper Script  
<http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/edg-rm-wrapper.pdf>

## APPLICABLE DOCUMENTS



- [A1] EDG User's Guide  
<http://marianne.in2p3.fr/datagrid/documentation/EDG-Users-Guide-2.0.pdf>
- [A2] LCG-1 User Guide  
<http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/LCG-1-UserGuide.htm>
- [A3] LDAP Services User Guide  
[http://hepunix.rl.ac.uk/edg/wp3/documentation/wp3-ldap\\_user\\_guide.html](http://hepunix.rl.ac.uk/edg/wp3/documentation/wp3-ldap_user_guide.html)
- [A4] LCG-2 User Scenario  
<https://edms.cern.ch/file/498081//UserScenario2.pdf>
- [A5] LCG-2 Frequently Asked Questions  
<https://edms.cern.ch/document/495216/>
- [A6] Tank And Spark  
[http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/internal/chep04/SW\\_Installation.pdf](http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/internal/chep04/SW_Installation.pdf)
- [A7] How to Manually configure and test the Experiment Software Installation mechanism on LCG-2  
[http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/configuration\\_of\\_tankspark](http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/configuration_of_tankspark)

## 1.6. TERMINOLOGY

### 1.6.1. Glossary

|                 |   |
|-----------------|---|
| <b>AFS:</b>     | Andrew File System                                    |
| <b>API:</b>     | Application Programming Interface                     |
| <b>BDII:</b>    | Berkeley Database Information Index                   |
| <b>CASTOR</b>   | CERN Advanced STORage manager                         |
| <b>CE:</b>      | Computing Element                                     |
| <b>CERN:</b>    | European Laboratory for Particle Physics              |
| <b>ClassAd:</b> | Classified advertisement                              |
| <b>CLI:</b>     | Command Line Interface                                |
| <b>CNAF:</b>    | INFN's National Center for Telematics and Informatics |
| <b>dcap:</b>    | dCache Access Protocol                                |
| <b>DIT:</b>     | Directory Information Tree                            |
| <b>DLI:</b>     | Data Location Interface                               |
| <b>DN:</b>      | Distinguished Name (LDAP's)                           |
| <b>EDG:</b>     | European DataGrid                                     |
| <b>EDT:</b>     | European DataTag                                      |
| <b>EGEE:</b>    | Enabling Grids for E-science                          |



---

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



---

|                      |  |
|----------------------|--|
| <b><i>R-GMA:</i></b> | Relational Grid Monitoring Architecture      |
| <b><i>RLI:</i></b>   | Replica Location Index                       |
| <b><i>RLS:</i></b>   | Replica Location Service                     |
| <b><i>RM:</i></b>    | Replica Manager                              |
| <b><i>RMC:</i></b>   | Replica Metadata Catalog                     |
| <b><i>RMS:</i></b>   | Replica Management System                    |
| <b><i>ROS:</i></b>   | Replica Optimization Service                 |
| <b><i>SASL:</i></b>  | Simple Authorization & Security Layer (LDAP) |
| <b><i>SE:</i></b>    | Storage Element                              |
| <b><i>SFT:</i></b>   | Site Functional Tests                        |
| <b><i>SMP:</i></b>   | Symmetric Multi Processor                    |
| <b><i>SRM:</i></b>   | Storage Resource Manager                     |
| <b><i>SURL:</i></b>  | Storage URL                                  |
| <b><i>TURL:</i></b>  | Transport URL                                |
| <b><i>UI:</i></b>    | User Interface                               |
| <b><i>URI:</i></b>   | Uniform Resource Identifier                  |
| <b><i>URL:</i></b>   | Universal Resource Locator                   |
| <b><i>UUID:</i></b>  | Universal Unique ID                          |
| <b><i>VDT:</i></b>   | Virtual Data Toolkit                         |
| <b><i>VO:</i></b>    | Virtual Organization                         |
| <b><i>WMS:</i></b>   | Workload Management System                   |
| <b><i>WN:</i></b>    | Worker Node                                  |
| <b><i>WPn:</i></b>   | Work Package #n                              |



---

## 2. EXECUTIVE SUMMARY

This user guide is intended for users of the LCG-2 middleware. 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 retrieving of information of resources status, etc., in order to easily be effective.

An introduction to the the LCG-2 middleware is presented in Chapter 3. This chapter is the starting point for all the others, since it describes all the middleware components and provides most of the necessary terminology. It also presents the EGEE project, within which LCG-2 software is being used.

In Chapter 4, the procedures to get a certificate, join a Virtual Organization and manage proxies in order to start working in the Grid are described.

Details on how to get information about the status of Grid resources are given in Chapter 5, where the different Information Services, their architecture and interface, are discussed.

An overview of the Workload Management service is given in Chapter 6. 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 7. Not only the high-level interface is described but also commands that can be useful in case of problems or for debugging purposes.

Finally, the appendices give information about the middleware components of LCG-2 releases (Appendix A), the most interesting configuration files and enviromental variables for users (Appendix B), the possible states of a job during submission and execution (Appendix C), user tools for the Grid (Appendix D), VO-wide utilities (Appendix E, APIs for data management and file access (Appendix F), and the GLUE Schema used to describe Grid resources (Appendix G).



### 3. OVERVIEW

The LCG-2 Grid middleware comes from a number of Grid development projects, like DataGrid, DataTag, Globus, GriPhyN, iVDGL, and *EGEE project (Enabling Grids for E-science)* [R1]. This middleware is currently installed also in sites participating in EGEE.

The *LHC Computing Grid Project (LCG)* [R2] was born to prepare the computing infrastructure for the simulation, processing and analysis of the data of the *Large Hadron Collider (LHC)* experiments. The LHC, which is being constructed at the European Laboratory for Particle Physics (*CERN*), will be the world's largest and most powerful particle accelerator.

The case of the LHC experiments illustrates well the motivation behind Grid technology. The LHC accelerator will start operation in 2007, and the experiments that will use it (*ALICE, ATLAS, CMS* and *LHCb*) will generate enormous amounts of data. The processing of this data will require large computational and storage resources and the associated human resources for operation and support. It was not considered feasible to fund all of the resources at one site, and so it was agreed that the LCG computing service would be implemented as a geographically distributed *Computational Data Grid*. This means that the service will use computing and storage resources, installed at a large number of computing sites in many different countries, interconnected by fast networks. LCG-2 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.

**Note:** For historical reasons, throughout this text we will often speak about the LCG-2 Grid, the LCG-2 services, sites participating in LCG-2, etc. This can be interpreted as the Grid, services and sites using LCG-2 middleware, and also participating in the EGEE project.

The users of a Grid infrastructure are divided into *Virtual Organisations (VO)* [R3], abstract entities grouping users, institutions and resources in the same administrative domain [R4].

The LCG-2 VOs correspond to real organisations or projects, such as the four LHC experiments; other VOs exist in the context of EGEE, like the community of biomedical researchers, etc. One special VO is DTeam, which is formed by the EGEE site administrators and the members of the LCG Grid Deployment Group.

An updated list of all the LCG-2/EGEE VOs can be found in the first lines of the following web page:

<http://goc.grid.sinica.edu.tw/gstat/service.html>

while an updated list of the participating sites is visible at the following URL:

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



---

## 3.1. PRELIMINARY MATTERS

### 3.1.1. Code Development

Many of the services offered by LCG-2 can be accessed both by 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).

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

A totally different matter is the development of software that forms part of the 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 can refer to [R6].

### 3.1.2. Troubleshooting

This document will also explain the meaning of the most common error messages and give some advice on how to avoid some common errors. The guide cannot, however, thoroughly include all the possible failures a user may find while using LCG-2. These errors may be produced due to his own mistakes, to misconfiguration of the Grid components, or even to bugs in the Grid middleware.

The references provided in the different sections, in general, deal with the commands and services of LCG-2 in greater detail than this user guide does.

The *Global Grid User Support (GGUS)* centralizes the user support for LCG-2, by answering questions, tracking known problems, maintaining lists of frequently asked questions, etc. The entrance point to this service is a web site, with the following URL:

<http://www.ggus.org>

The GGUS portal is the only entry point for Grid users looking for help.

To report bugs in the middleware or to ask for new functionalities, a bug may (and should, for the benefit of other users) be submitted to the LCG *Savannah Portal*, whose URL follows:

<https://savannah.cern.ch/projects/lcoperation>

Finally, if the user thinks there is a security risk in the Grid, then he may contact directly the site administrator (if the situation is urgent, and this may be faster that go through GGUS). Information on the local site contacts can be obtained in the Information Service or in the GOC web site, which is described in Chapter 4.





### 3.1.3. User and VO utilities

In this guide only information useful for the average user is provided. Thus, only core LCG-2 middleware is described. Nevertheless, there exist several user tools (that use the middleware, rather than being part of it) that can be very useful to the user. Some of these tools are summarised in Appendix D.

Likewise, there are utilities that are only available to certain (authorized) users of the Grid. Example of this is the administration of the resources viewed by a VO or the installation of VO software in LCG-2 nodes. Authorized users can install software in the computing resources of LCG-2. The installed software is also published in the Information Service, so that user jobs can run on nodes where the software they need is installed. Information on such topics is given in Appendix E.

## 3.2. THE LCG-2 ARCHITECTURE

This section provides a quick overview of the LCG-2 architecture and services.

### 3.2.1. Security

As explained before, LCG-2 is organised in Virtual Organisations. Before LCG resources can be used, a user must read and agree to the LCG usage rules and register some personal data, including the Virtual Organisation he belongs to, with a Registration Service.

Once the user registration is complete, he can access LCG-2. 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 authenticate with Grid resources, a user needs to have a digital X.509 certificate issued by a *Certification Authority (CA)* trusted by LCG.

The user certificate, whose private key is protected by a password, is used to generate and sign a temporary certificate, called *proxy*, which is used for the actual authentication and does not have a password. As the possession of the proxy certificate is a proof of identity, the file containing it must be kept readable only by the user and the proxy has, by default, a short lifetime to reduce security risks.

The authorization of a user on a specific Grid resource can be done in two different ways. The first is simpler, and relies on the *grid-mapfile* mechanism. The Grid resource has a local grid-mapfile which maps user certificates to local accounts. When a user request-for-service reaches a host, the certificate subject of the user (which is present in the proxy) is checked against what is in the local grid-mapfile to find out to which local account (if any) the user certificate is mapped to, and this account is used to perform the requested operation [R8]. The second way relies on the Virtual Organisation Membership



---

Service (VOMS) and the LCAS/LCMAPS mechanism, which allow a more detailed definition of the user privileges, and will be explained in more detail later.

For long jobs, it may happen that the user proxy expires before the user job has finished, causing all subsequent requests for service to fail due to unauthorized access. In order to avoid this, the Workload Management Service provided by the *European DataGrid (EDG)* [R9] allows for proxy renewal before the expiration time has been reached if the job requires it. The *Proxy Server (PS)* is the component that allows for such functionality.

The list of sites that have installed a PS can be consulted at the LCG Grid Operations Centre, described in Section 4.5.

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

### 3.2.2. The User Interface

The 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 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 systems. It provides a CLI to perform some basic Grid operations:

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

In addition, the LCG-2 API libraries are also available in the UI for users to develop Grid applications. One or more UIs are available to all VOs.

### 3.2.3. Computing Element and Storage Element

A *Computing Element (CE)* is defined as a Grid batch queue and is identified by a string like `<hostname>:<port>/<batch_queue_name>`. It is important to notice that according to this definition, sev-



---

eral queues defined for the same hostname are considered different CEs. This is currently used to define different queues for jobs of different lengths or for different VOs.

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 farm of computing nodes called *Worker Nodes (WN)*, a Local Resource Management System (LRMS) and a node acting as a *Grid Gate (GG)*, or *Gatekeeper*, which acts as a front-end to the rest of the Grid. The details of the components and daemons running on the CE (and in the rest of LCG-2 nodes) can be found in [R10].

In LCG-2 the supported LRMS are the Portable batch System (PBS), the Load Sharing Facility (LSF), Torque 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 libraries 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 to storage resources. The Storage Element may control simple disk servers, large disk arrays or Mass Storage Systems (MSS). Each LCG-2 site provides one or more SEs.

Storage elements can support different data access protocols and interfaces. They are described in Section 7.2 in much more detail than here. Please refer there for definitions the concepts used in this section.

Regarding data movements, *GSIFTP* is the protocol for file transfers (it is basically a GSI-secure FTP), while secure and insecure *RFIO* and *conceptgsidcap* are used for file access (not only file copy). The *file* protocol is no longer supported in LCG-2 for remote file access. It may only be used to specify files that are located in a local filesystem

As for what concerns the APIs supported by the SEs, some storage resources are managed by a *Storage Resource Manager (SRM)* [R11]. This middleware module allows to manage the contents of the storage resource and provides capabilities like transparent migrations from disk to tape, file pinnings, reservations, etc.

The simplest SEs, referred to as *classic Storage Elements*, do not have a SRM interface and offer no storage management functionalities. They are basically machines with a disk and a GSIFTP server. Additionally, they usually support the insecure RFIO access protocol.

Other types of SEs can be Mass Storage Systems (with front-end disks and back-end tape storages), like *CASTOR*, and pools of disks (and a disk pool manager), like *dCache*. These usually present an SRM

interface.

A recently introduced type of SE is the *Disk Pool Manager (DPM)*, created by LCG. The DPM should replace the classic SE, offering much of the functionality of dCache but avoiding its complexity and being much easier to install, manage and maintain.

The most common types of SEs currently present in LCG-2 are summarized in the following table:

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

#### 3.2.4. Information Service

The *Information Service (IS)* provides information about the LCG-2 Grid resources and their status. This information is essential for the operation of the whole Grid. It is through the Information Service that available CEs to run jobs can be located and also that the SEs holding replicas of Grid files and the catalogs keeping the information on these files are found.

The published information is also used for monitoring and accounting purposes. Namely, the monitoring of the published data is used for analyzing usage and performance of the Grid, detecting fault situations and any other interesting events. The accounting services allow to create statistics of the applications run by users in the LCG resources. The resources consumed by the VOs and provided by the sites as a function of time can be determined using these services.

The data published in the IS 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 [R12].

In LCG-2, the *Monitoring and Discovery Service (MDS)* from Globus [R13] was adopted as main provider of the Information Service. Recently, however, a new type of Information Service has started to be deployed, and many applications are already using it. This is the *Relational Grid Monitoring Architecture (R-GMA)* [R14].

Both Information Services are described as follows.

#### Monitoring and Discovery Service (MDS)

The Monitoring and Discovery Service implements the GLUE Schema<sup>1</sup> using OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol (LDAP).

LDAP is a specialized database optimized for reading, browsing and searching information. In particular in LCG-2, only anonymous access to the catalog is offered. This means that all users can browse the catalogs 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 *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)*.

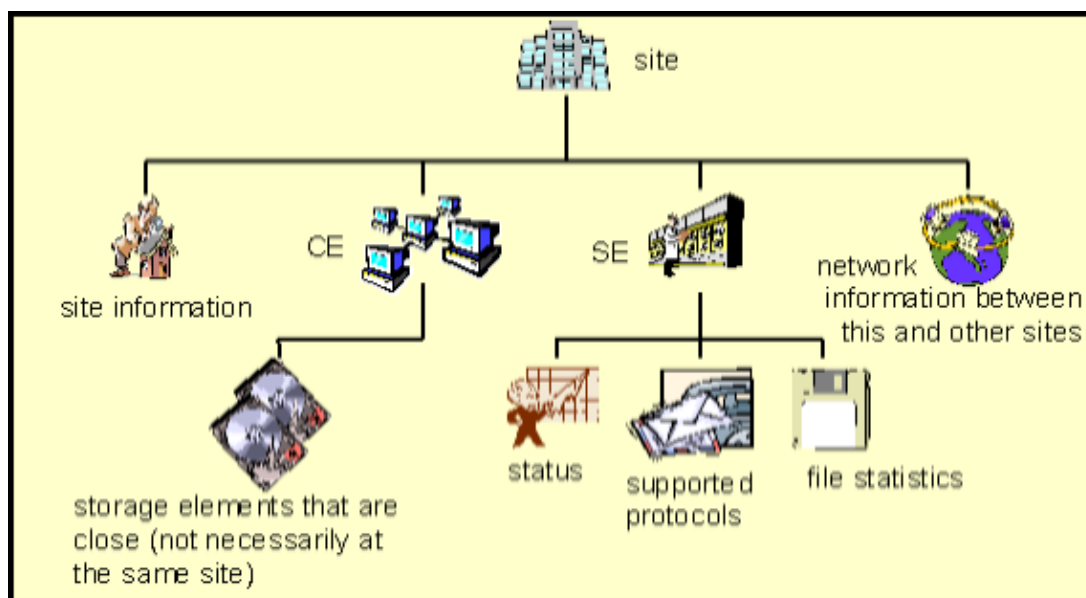


Figure 1: The Directory Information Tree (DIT)

Figure 1 shows an example of a Directory Information Tree (DIT) of a site. 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`), and 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 G.

<sup>1</sup>Actually, in the current IS some additional attributes (not part of the standard GLUE schema) are also being published and used.

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.

The MDS is not the only possible GLUE-based Information Service. R-GMA, which is, as stated earlier, already present in LCG-2, also implements it.

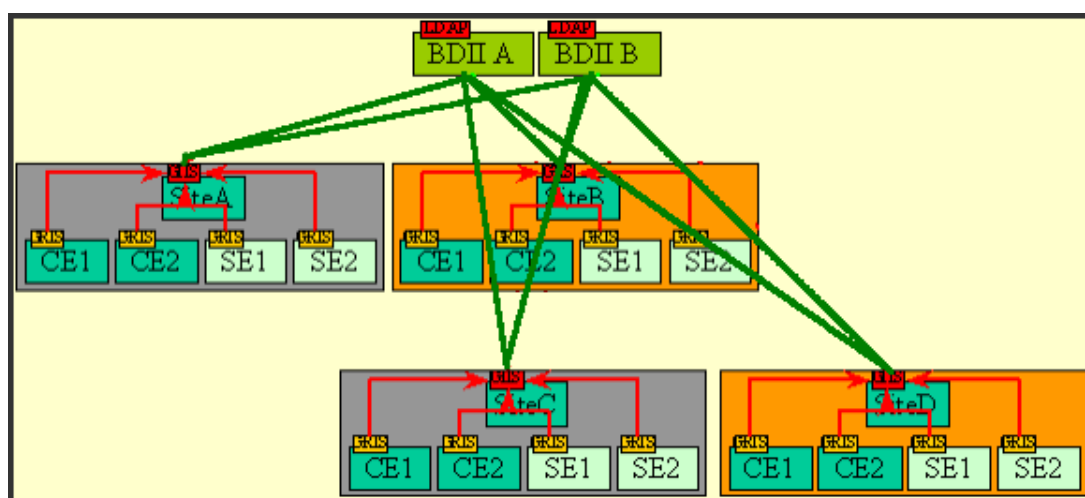


Figure 2: The MDS Information Service in LCG-2

Figure 2 shows how the information is stored and propagated in LCG-2. Computing and storage resources at a site implement an entity called *Information Provider*, which generates the relevant information of the resource based on some static configuration files and the gathering of dynamic information (e.g.: the used space in a SE). This information is published via an LDAP server by the *Grid Resource Information Servers, or GRISes*.

In each site an element called the *Site Grid Index Information Server (GIIS)* compiles all the information of the different GRISes and publishes it. Actually, current release of LCG-2 recommends using a *Berkeley DB Information Index (BDII)* instead of a GIIS since it increases the stability of the information published by a site. This BDII is called the *site BDII*.

Finally, a BDII is also used as the top level of the IS hierarchy. This BDII queries the GIISes and acts as a cache storing information about the Grid status in its database. Therefore, by querying the BDII a user or a service has all the available information about the Grid. Nevertheless, it is always possible to get information about specific resources (maybe more up-to-date) by querying directly the site GIISes (or site BDIIs) or even the local GRISes.

Each BDII contains information from the sites that are included in a configuration file, which it accesses through a web interface. In this way, each site can easily decide which information they desire to

publish and mechanisms to automatically exclude sites (or individual resources) that presents temporary problems can be put in place.

### Relational Grid Monitoring Architecture (R-GMA)

R-GMA is an implementation of the *Grid Monitoring Architecture (GMA)* proposed by the *Global Grid Forum (GGF)*. In R-GMA, the information is presented as being in a global distributed relational database. This model is more powerful than the LDAP-based, since relational databases support more advanced query operations than LDAP does, and also support schema modifications in an easier way.

The architecture consists of three major components:

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

The user or any user application does not need to know the registry; this will be handled directly by the consumers and producers behind it. From the user's point of view, the information and monitoring system appears like a large relational database and it can be queried as such. Hence, R-GMA imposes a subset of SQL as standard query language. The Producers publish tuples (database rows) with an SQL `insert` statement and Consumers query them using SQL `select` statements.

Figure 3 shows the Grid Monitoring architecture. In this figure, the Registry communication is shown by a dotted line and the main flow of data by a solid line.

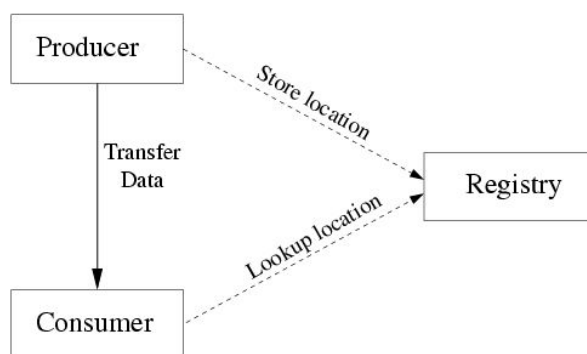


Figure 3: The Grid Monitoring Architecture (GMA)

For a given VO, R-GMA presents the resources information as a single virtual database containing a set of virtual tables. As Figure 4 shows, a schema contains the name and structure (column names, types

and settings) of each virtual table in the system. The registry contains a list of producers who publish information for each table. A consumer runs a SQL query for a table and the registry selects the best producers to answer the query through a process called *mediation*. The consumer then contacts each producer directly, combines the information and returns a set of n-tuples. This process is hidden for the user. Since there is no central repository holding the contents of the virtual table, it can be considered a virtual database.

To be sure, it is the registry and the schema that define an R-GMA system. What information will be seen by a consumer depends on what producers are registered within the registry. There is only one registry and one schema for the whole LCG-2 production. If a VO creates a different registry, then the consumers using it will have only access to the information registered within it (which may be of no interest to the rest of LCG-2 users).

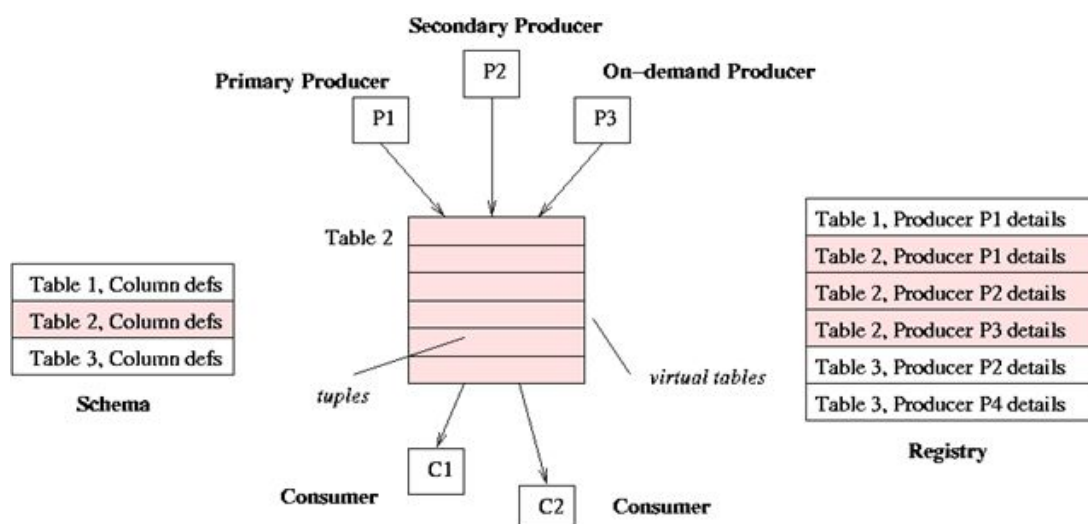


Figure 4: The virtual database of R-GMA

There are two types of producers:

- **Primary producers** publish information coming from a user or an Information Provider.
- **Secondary producers** consume information from Primary Producers and republish it. Normally they use a database to store the data.

The producers can also be classified depending on the type of queries they may accept:

- **Continuous (or stream)**: Tuples are sent directly to the consumer when produced.
- **Latest**: Only the latest tuples are sent to the consumer.



- **History:** All the tuples are kept to later retrieval by the consumers.

Currently, primary producers are of type stream, and secondary producers (that use a database) must be set up to archive interesting information and be able to reply to latest and history queries. Secondary producers are also required for joins to be supported in the consumer queries. That is, for queries involving different tables, all the necessary data must have been stored first in a secondary producer's database.

R-GMA is called to be the replacement for MDS in the future. Although MDS is still (and will be for some time) the main source of information for LCG-2 middleware components, R-GMA is already deployed and many applications use it, specially for accounting and monitoring purposes.

### 3.2.5. Data Management

In a Grid environment, the data files can be replicated to many different sites depending on where the data is needed. Ideally, 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 referred to by different names: *Grid Unique Identifier (GUID)*, *Logical File Name (LFN)*, *Storage URL (SURL)* and *Transport URL (TURL)*. While the GUIDs and LFNs refer to files and not replicas, and say nothing about locations, the SURLs and TURLs give information about where a physical replica is located.

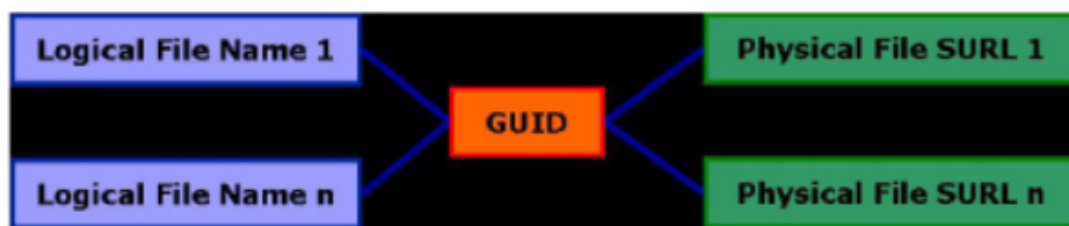


Figure 5: Different filenames in LCG-2

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 provides informations about the physical location of the file (hostname and path). Currently, the SURLs have the following format `sfn:<SE_hostname>/<local_string>` or `srm:<SE_hostname>/<local_string>` depending on the type of SE holding 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 (i.e. the TURL must be understood by the entity serving the file).

Figure 5 shows the relationship between different names of the file.

The mappings between the different names are kept in an appropriate service called *File Catalog*; while files are stored in Storage Elements. Currently, two types of File Catalogs are available: the *Replica Location Service (RLS)* (used so far by all VOs) and the *LCG File Catalog (LFC)* (created to overcome performance and functionality issues observed in RLS). The different types of file catalog and Storage Elements will be described in more details in Chapter 7.

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

The Workload Management System can interact with the Data Management components in order to fulfill requirements of data access or storage from a job. The relationship between the different Data Management services and other Grid elements is illustrated in Figure 6 (some of the components appearing in the figure are described in the next sections).

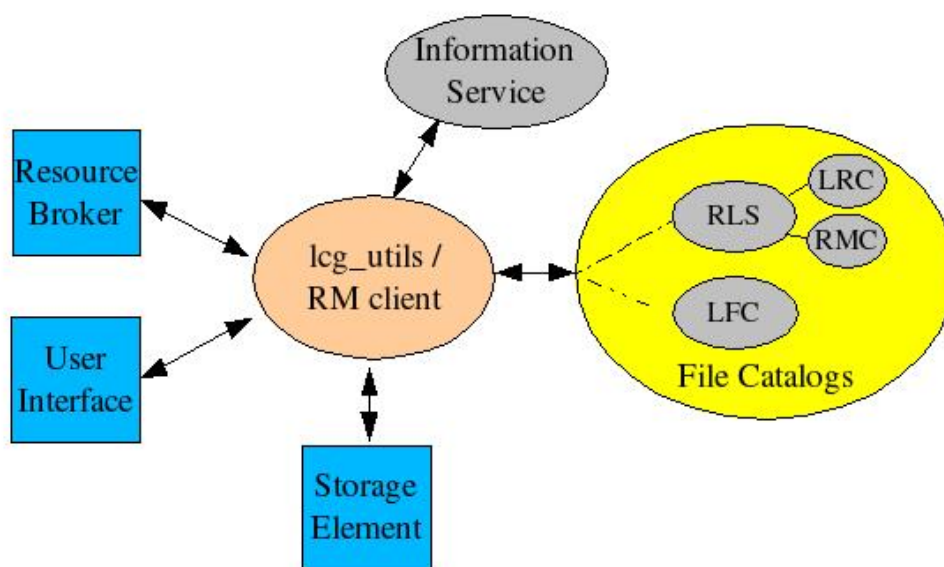


Figure 6: Data Management services and other Grid components



### 3.2.6. Job Management

The services of the *Workload Management System (WMS)* are responsible for the acceptance of submitted jobs and for sending those jobs to the appropriate CE (depending on the job requirements and the available resources). For that purpose, the WMS must retrieve information from the IS and the File Catalog. The *Resource Broker (RB)* is the machine where the WMS services run. A detailed description of these services can be found in [R10].

It is important to say that the submission of a job requires GSI authentication both between the UI and the RB and between the RB and the CE. It also implies the fulfillment of the match-making process: the search of the best CE given the requirements of the submitted job: characteristics of the WNs, presence of specified files in close SEs<sup>2</sup>, etc.

For the task of locating requested files (the SEs where they sit), another service, the *Data Location Interface (DLI)* is contacted by the RB. In turn, this DLI will contact the appropriate file catalog to query for the file. In this way, the Resource Broker can talk to different file catalogs, depending on settings or user preferences.

Also task of the RB is the creation of the `.BrokerInfo` file, which is shipped with the job to the WN. This file contains informations such as which the close SE is, and what the result of the match-making process has been. Information can be retrieved from this file using the `BrokerInfo` CLI or an API.

It is also interesting to note that a wrapper script is created around the actual user job, and it is this script that gets executed in a WN. This wrapper performs additional tasks such as correctly setting the job environment and generating logging information.

Finally, the *Logging and Bookkeeping service (LB)* [R15] is usually also run on a RB machine or on a close one. 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.

### 3.3. 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.

---

<sup>2</sup>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 bandwidth connection) as "close". The other SEs are considered "not close".

### 3.3.1. Job Submission

Figure 7 illustrates the process that takes place when a job is submitted to the Grid. The individual steps are described as follows.

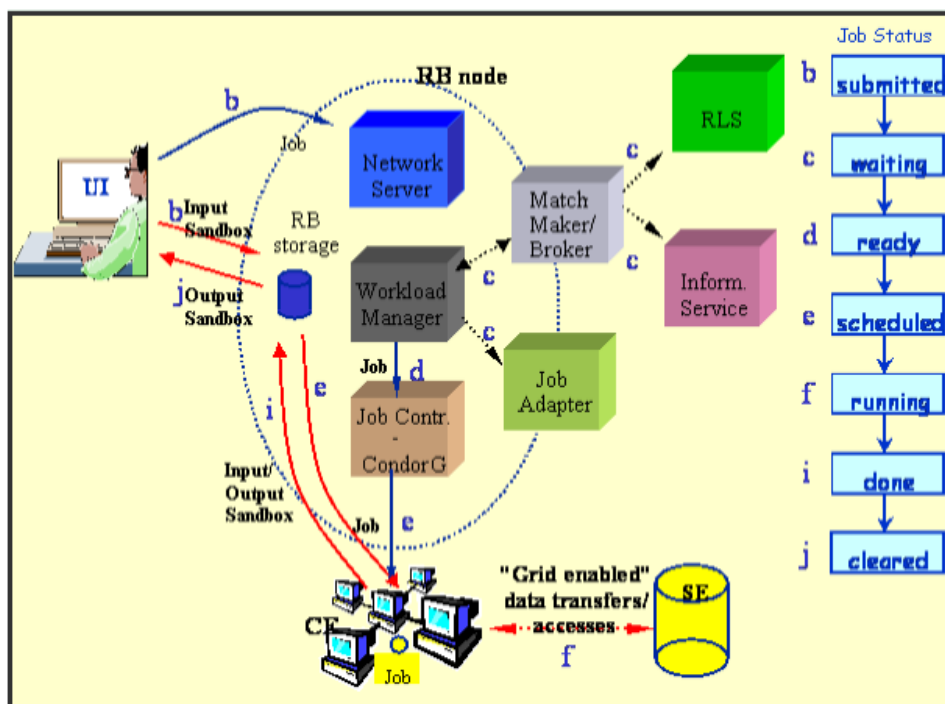


Figure 7: Job flow in the LCG-2

- a. After obtaining a digital certificate from one of the LCG-2 trusted CAs, registering with an LCG-2 VO and obtaining an account on an LCG-2 UI, the user is ready to use the LCG-2 Grid. He logs to the UI machine and creates a proxy certificate to authenticate himself in every secure interaction.
- b. The user submits the job from the UI to the Resource Broker node. In the job description file, one or more files to be copied from the UI to the RB node can be specified. 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 looks for the best available CE to execute the job. To do so, it interrogates the BDII to query the status of computational and storage resources and the File Catalog to find the location of required data. The event is logged in the LB and the status of the job is WAITING.
- d. The RB prepares the job for submission creating a wrapper script that will be passed, together with other parameters, to the selected CE. The event is logged in the LB and the status of the job is READY.



- e. The CE receives the request and sends the job for execution to the local LRMS. 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 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 from a close SE using either the RFIO or gsidcap protocols, or from remote SEs after copying them locally to the WN local filesystem with the Data Management tools.
- 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 it in the file catalogs. 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 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.
- l. If the site where the job is unable to accept it, the job will be automatically resubmitted to another CE that still satisfies the user requirements. After the maximum allowed number of resubmissions is reached, the job will be marked as aborted. Users can get information about what happened by querying the LB service.

### 3.3.2. Other operations

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

Concerning the File Catalog, the user is not supposed to directly interact with it; instead, he should use the LCG tools, or the POOL interface, when this exists (see Section 7.10). A wrapper around the LCG DM tools with the same syntax as the old EDG Replica Manager is also provided for backward compatibility.



---

Users can interrogate the IS to retrieve static or dynamic information about the status of LCG-2. Although site GIISes/BDIIs, or even GRISes can be directly queried, it is recommended to query only a central BDII. Details and examples on how to interrogate GRIS, GIIS and BDII are given in Chapter 5.



## 4. GETTING STARTED

This section describes the preliminary steps to gain access to the 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 Organisations (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

#### 4.1.1. X.509 Certificates

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

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

[http://lcg-registrar.cern.ch/pki\\_certificates.html](http://lcg-registrar.cern.ch/pki_certificates.html)

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
```



### 4.1.2. Generating a request

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

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

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

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

The `grid-cert-request` command will create the following 3 files:

|                           |   |
|---------------------------|---|
| <code>userkey.pem</code>  | contains the private key associated with the certificate. ( <b>This should be set with permissions so that only the owner can read it</b> ) (i.e. <code>chmod 400 userkey.pem</code> ). |
| <code>userreq.pem</code>  | contains the request for the user certificate.  |
| <code>usercert.pem</code> | should be replaced by the actual certificate when sent by the CA. (This should be readable by everyone) (i.e. <code>chmod 444 usercert.pem</code> ).                                    |

Then the `userreq.pem` file is sent (usually by e-mail using a particular format) to the desired CA.

### 4.1.3. Getting the Certificate

No matter how a request is generated and sent to a CA, the CA will have to confirm that the user asking for a certificate is who he claims he is. This usually involves a physical encounter or a phone call. After approval, the resulting certificate is delivered to the user. This can be done via e-mail, or by giving instructions to the user to download it from a web page. If the certificate was directly installed in the user's browser, then it must be exported (saved) to disk for Grid use. Details of how to do this will depend on supported browser versions and are described on the CA's website.

The received certificate will usually be in one of two formats: PEM (extension `.pem`) or PKCS12 format (extension `.p12`). This last one is the most common one for certificates installed in a browser, but it is the other one, the PEM format, which must be used in LCG-2. Fortunately, the certificates can be converted from one format to the other.





If the certificate is in PKCS12 format, then it can be converted to PEM using the `pkcs12` command on a machine with the `openssl` package installed, 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:

|                           |   |
|---------------------------|---|
| <code>my_cert.p12</code>  | is the path for the input PKCS12 format file.   |
| <code>userkey.pem</code>  | is the path to the output private key file.     |
| <code>usercert.pem</code> | 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 pass phrase 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.

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

#### 4.1.4. Renewing the Certificate

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

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

## 4.2. REGISTERING WITH LCG-2

### 4.2.1. The Registration Service

Before a user can use the LCG-2 service, registration of some personal data and acceptance of some Grid usage rules are necessary. Each VO must ensure that all its members have provided the necessary information to the VO's database and have accepted the usage rules. The procedure through which this is accomplished may vary from VO to VO.



As an example of registration service, we describe here the example of the *LCG Registrar*, which serves several important VOs. 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 loaded 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. They require the PKCS12 format 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:

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

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

[http://lcg-registrar.cern.ch/load\\_certificates.html](http://lcg-registrar.cern.ch/load_certificates.html)

#### 4.2.2. Virtual Organisations

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

Becoming member of a VO usually requires being a member of the respective 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 VO when registering with LCG-2 and the supplied information is forwarded to the VO managers ([R4]) for validation before the registration process is completed. In the case of LCG Registrar, this forwarding is accomplished by the registration interface back-end automatically. 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 your respective registration service.

As it was explained in Chapter 3, in order to find which VOs are currently defined in EGEE, the following URL can be checked:

`http://goc.grid.sinica.edu.tw/gstat/service.html`

### 4.3. SETTING UP THE USER ACCOUNT

#### 4.3.1. The User Interface

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

As an example, the CERN LXPLUS service can be used as UI as described in [R16]. This use could be extended to other (non LXPLUS) machines mounting AFS.

As an alternative, the user can install the UI software on his machine (see the Installation and Administration Guide [R17]).

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.

#### 4.3.2. 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 4.3.2.1 (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
    Validity
      Not Before: Sep 11 11:37:57 2002 GMT
      Not After : Nov 30 12:00:00 2003 GMT
    Subject: O=Grid, O=CERN, OU=cern.ch, CN=John Doe
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:ab:8d:77:0f:56:d1:00:09:b1:c7:95:3e:ee:5d:
          c0:af:8d:db:68:ed:5a:c0:17:ea:ef:b8:2f:e7:60:
          2d:a3:55:e4:87:38:95:b3:4b:36:99:77:06:5d:b5:
          4e:8a:ff:cd:da:e7:34:cd:7a:dd:2a:f2:39:5f:4a:
          0a:7f:f4:44:b6:a3:ef:2c:09:ed:bd:65:56:70:e2:
          a7:0b:c2:88:a3:6d:ba:b3:ce:42:3e:a2:2d:25:08:
          92:b9:5b:b2:df:55:f4:c3:f5:10:af:62:7d:82:f4:
          0c:63:0b:d6:bb:16:42:9b:46:9d:e2:fa:56:c4:f9:
          56:c8:0b:2d:98:f6:c8:0c:db
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      Netscape Base Url:
        http://home.cern.ch/globus/ca
      Netscape Cert Type:
        SSL Client, S/MIME, Object Signing
      Netscape Comment:
        For DataGrid use only
      Netscape Revocation Url:
        http://home.cern.ch/globus/ca/bc870044.r0
      Netscape CA Policy Url:
        http://home.cern.ch/globus/ca/CPS.pdf
    Signature Algorithm: md5WithRSAEncryption
      30:a9:d7:82:ad:65:15:bc:36:52:12:66:33:95:b8:77:6f:a6:
      52:87:51:03:15:6a:2b:78:7e:f2:13:a8:66:b4:7f:ea:f6:31:
      aa:2e:6f:90:31:9a:e0:02:ab:a8:93:0e:0a:9d:db:3a:89:ff:
      d3:e6:be:41:2e:c8:bf:73:a3:ee:48:35:90:1f:be:9a:3a:b5:
      45:9d:58:f2:45:52:ed:69:59:84:66:0a:8f:22:26:79:c4:ad:
      ad:72:69:7f:57:dd:dd:de:84:ff:8b:75:25:ba:82:f1:6c:62:
      d9:d8:49:33:7b:a9:fb:9c:1e:67:d9:3c:51:53:fb:83:9b:21:
      c6:c5
```

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



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

#### **Example 4.3.2.2** (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/does/.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
```

## **4.4. PROXY CERTIFICATES**

### **4.4.1. Proxy Certificates**

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

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

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

To create a proxy certificate, issue the command:

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

If the command is successful, the output will be like



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

and the proxy certificate will be written in `/tmp/x509up_u<uid>`, where `<uid>` is the Unix UID of the user, unless the environment variable `X509_USER_PROXY` is defined (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 pass phrase or bad file permissions
key file location: /home/does/.globus/userkey.pem
Use -debug for further information.
```

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

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

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

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

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

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

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

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



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

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

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

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

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

#### ***Example 4.4.1.3 (Destroying a proxy certificate)***

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

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

If no proxy certificate exists, the result will be:

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

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

#### **4.4.2. Virtual Organisation Membership Service**

The *Virtual Organisation Membership Service (VOMS)* is a new service to manage authorization information in VO scope. This service can already be used in LCG-2 yet, but not all the features it provides are already functional.

The VOMS system should 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 gives the `voms-proxy-init` command instead of `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 performs. Therefore, the user will not have to specify it using a `--vo` option.

The VOMS system will give additional capabilities to the management of VOs and users. Unlike the current situation, where all users in a VO have the same rights and capabilities, when VOMS is fully deployed, users will be divided in groups inside the VO and will hold different roles. In this way, some users will be able to access resources and perform actions that others will not.

**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 6), even if he 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 proxy) is checked for the authorization, and the job is aborted if the user's real VO is not supported in the destination CE.

#### 4.4.3. 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 Grid Operations Centre database, which will be presented in Section 4.5.

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 20 minutes. An error message like the following will be produced:

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





The advanced proxy management offered by the UI of 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 Chapter 6). 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 4.4.3.1** (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 pass phrase 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 4.4.3.2 (Retrieving information about a long-term proxy)***

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

```
$ myproxy-info -s <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 4.4.3.3 (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.

## **4.5. THE LCG GRID OPERATIONS CENTRE**

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 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 5.3, information on security in LCG-2, news...



## 5. INFORMATION SERVICE

The architecture of the LCG-2 Information Services, both MDS and R-GMA, was described in Chapter 3. In this chapter, we have a closer look at the structure of the information published by different elements of those architectures, and we examine the tools that can be used to get information from them.

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

Information of current tools used for monitoring in LCG-2 is also provided.

### 5.1. THE MDS

In the following sections examples are given on how to interrogate the MDS Information Service in LCG-2. In particular, the different servers from which the information can be obtained are discussed. These are the local GRISes, the site GISes/BDIIs and the global (or top) BDIIs. Of them, the BDII is usually the one queried, since it contains all the interesting information for a VO in a single place.

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

As explained in Chapter 3, the data in the IS of LCG-2 conforms to the LDAP implementation of the GLUE Schema, although some extra attributes (not initially in the schema) are also being published and actually queried and used by clients of the IS. For a list of the defined object classes and their attributes, as well as for a reference on the Directory Information Tree used to publish those attributes, please check Appendix G.

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

#### 5.1.1. `lcg-infosites`

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

**USAGE:** `lcg-infosites --vo <vo name> options -v <verbose level> --is <BDII to query>`



---

## Description of the Attributes:

vo : The name of the user vo (mandatory)

options : The tool admits the following options:

ce : The information related to number of CPUs, running jobs, waiting jobs and names of the CEs are provided. All these data group all VOs together.

-v 1 only the names of the queues will be printed.

-v 2 The RAM Memory together with the operating system and its version and the processor included in each CE are printed.

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

-v 1 only the names of the SEs will be printed.

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

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

lfc : The name of the machine hosting the LFC catalog is printed.

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

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

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

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

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

```
$ lcg-infosites --vo dteam ce
```

A typical output is as follows:

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



```
#CPU    Free  Total Jobs    Running Waiting ComputingElement
-----
 20     20     1           0         1    ce01.pic.es:2119/jobmanager-torque-dteam
 40     39     0           0         0    ceitep.itep.ru:2119/jobmanager-torque-dteam
 52     52     0           0         0    ce.prd.hp.com:2119/jobmanager-pbs-dteam
  8      8     2           0         2    ce01.lip.pt:2119/jobmanager-torque-dteam
  7      5     0           0         0    ce00.inta.es:2119/jobmanager-torque-dteam
  3      1     1           1         0    ce001.ibm.bas.bg:2119/jobmanager-pbs-long
 24     24     0           0         0    ingvar.nsc.liu.se:2119/jobmanager-torque-dteam
  2      2     0           0         0    lcg03.gsi.de:2119/jobmanager-torque-dteam
332    232     2           2         0    lcg06.gsi.de:2119/jobmanager-lcglsf-dteam
 55     55     0           0         0    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-A
 55     55     0           0         0    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-G
 89      0     1           0         1    cclcgceli01.in2p3.fr:2119/jobmanager-bqs-T
[...]
```

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

To know the status of the storage resources:

```
$ lcg-infosites --vo dteam se
```

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

```
Avail Space(Kb) Used Space(Kb) Type    SEs
-----
823769960      1760604      disk    seitep.itep.ru
68185000      4830436      disk    se01.lip.pt
221473672      4232672      disk    castorgrid.pic.es
69369504      1641044      disk    lcg04.gsi.de
79081684      66938752     disk    se00.inta.es
74759668      778299916    disk    teras.sara.nl
1000000000000  500000000000 mss     lcgse02.ifae.es
1000000000000  500000000000 mss     lcgse03.ifae.es
1316556216     508695064    mss     lcgse04.ifae.es
1000000000000  500000000000 mss     lcgse05.ifae.es
[...]
```

### **Example 5.1.1.3** (Listing the close Storage Elements)

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



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

```
Name of the CE: ce01.pic.es:2119/jobmanager-torque-dteam  
Name of the close SE:  castorgrid.pic.es
```

```
Name of the CE: ceitep.itep.ru:2119/jobmanager-torque-dteam  
Name of the close SE:  seitep.itep.ru
```

```
Name of the CE: ce.prd.hp.com:2119/jobmanager-pbs-dteam  
Name of the close SE:  se.prd.hp.com
```

```
Name of the CE: ce01.lip.pt:2119/jobmanager-torque-dteam  
Name of the close SE:  se01.lip.pt
```

```
Name of the CE: ce00.inta.es:2119/jobmanager-torque-dteam  
Name of the close SE:  se00.inta.es
```

```
Name of the CE: ce001.ibm.bas.bg:2119/jobmanager-pbs-long  
Name of the close SE:  se001.ibm.bas.bg
```

```
Name of the CE: ingvar.nsc.liu.se:2119/jobmanager-torque-dteam  
Name of the close SE:  ingvar-se.nsc.liu.se
```

```
Name of the CE: lcg03.gsi.de:2119/jobmanager-torque-dteam  
Name of the close SE:  lcg04.gsi.de  
[...]
```

#### ***Example 5.1.1.4 (Listing tags of installed software)***

In order to retrieve the tags corresponding to the software installation of a certain VO, use the command as follows:

```
$ lcg-infosites --vo atlas tag
```

```
Name of the TAG: VO-atlas-release-9.0.4  
  Name of the CE: wn-04-07-01-a.cr.cnaf.infn.it
```

```
Name of the TAG: VO-atlas-release-8.0.6  
Name of the TAG: VO-atlas-release-8.0.5  
Name of the TAG: VO-atlas-lcg-release-0.0.2  
Name of the TAG: VO-atlas-release-8.0.8  
Name of the TAG: VO-atlas-release-9.0.3  
Name of the TAG: VO-atlas-release-8.0.7  
Name of the TAG: VO-atlas-release-9.0.4
```



Name of the CE:bohr0001.tier2.hep.man.ac.uk

[...]

### 5.1.2. lcg-info

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

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

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

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

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

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

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

#### *Example 5.1.2.1 (Get the list of supported attributes)*

To have a list of the supported attributes, give:

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

the output is similar to:

| Attribute name | Glue object class | Glue attribute name              |
|----------------|-------------------|----------------------------------|
| EstRespTime    | GlueCE            | GlueCEStateEstimatedResponseTime |
| WorstRespTime  | GlueCE            | GlueCEStateWorstResponseTime     |





```
TotalJobs      GlueCE          GlueCEStateTotalJobs
TotalCPUs      GlueCE          GlueCEInfoTotalCPUs
[...]
```

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

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

You want to know how many jobs are running and how many free CPUs there are on CEs that have more than 100 CPUs and have Scientific Linux:

```
$ lcg-info --list-ce --query 'TotalCPUs>=100,OS=SL*' --attrs 'RunningJobs,FreeCPUs'
```

The output could be:

```
- CE: cclcgceli02.in2p3.fr:2119/jobmanager-bqs-T
  - RunningJobs      631
  - FreeCPUs         70

- CE: cclcgceli04.in2p3.fr:2119/jobmanager-bqs-T
  - RunningJobs      631
  - FreeCPUs         3
```

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

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

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

Similarly, suppose that you want to know which SEs have least 1 TB of available space for CMS and the CEs close to them:

```
$ lcg-info --list-se --vo cms --query 'AvailableSpace>=1000000000' --attrs CloseCEs
```

the output will be like:



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

The `--bdii` option can be used to specify a particular `bdii` (e.g. `--bdii lxn1187.cern.ch:2170`), and the `--sed` option can be used to output the results of the query in a format easy to parse in a script, in which values for different attributes are separated by `%` and values of list attributes are separated by `&`.

### 5.1.3. The Local GRIS

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

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

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

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

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

The same effect can be obtained with:

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

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



---

### **Example 5.1.3.1**    *(Interrogating the GRIS on a Computing Element)*

The command used to interrogate the GRIS located on host lxnl181 is:

```
$ ldapsearch -x -h lxnl181.cern.ch -p 2135 -b "mds-vo-name=local, o=grid"
```

or:

```
$ ldapsearch -x -H ldap://lxnl181.cern.ch:2135 -b "mds-vo-name=local, o=grid"
```

And the obtained reply will be:

```
version: 2

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

# lxnl181.cern.ch/siteinfo, local, grid
dn: in=lxnl181.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

# lxnl181.cern.ch:2119/jobmanager-lcgpbs-infinite, local, grid
dn: GlueCEUniqueID=lxnl181.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 G.

### ***Example 5.1.3.2 (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.

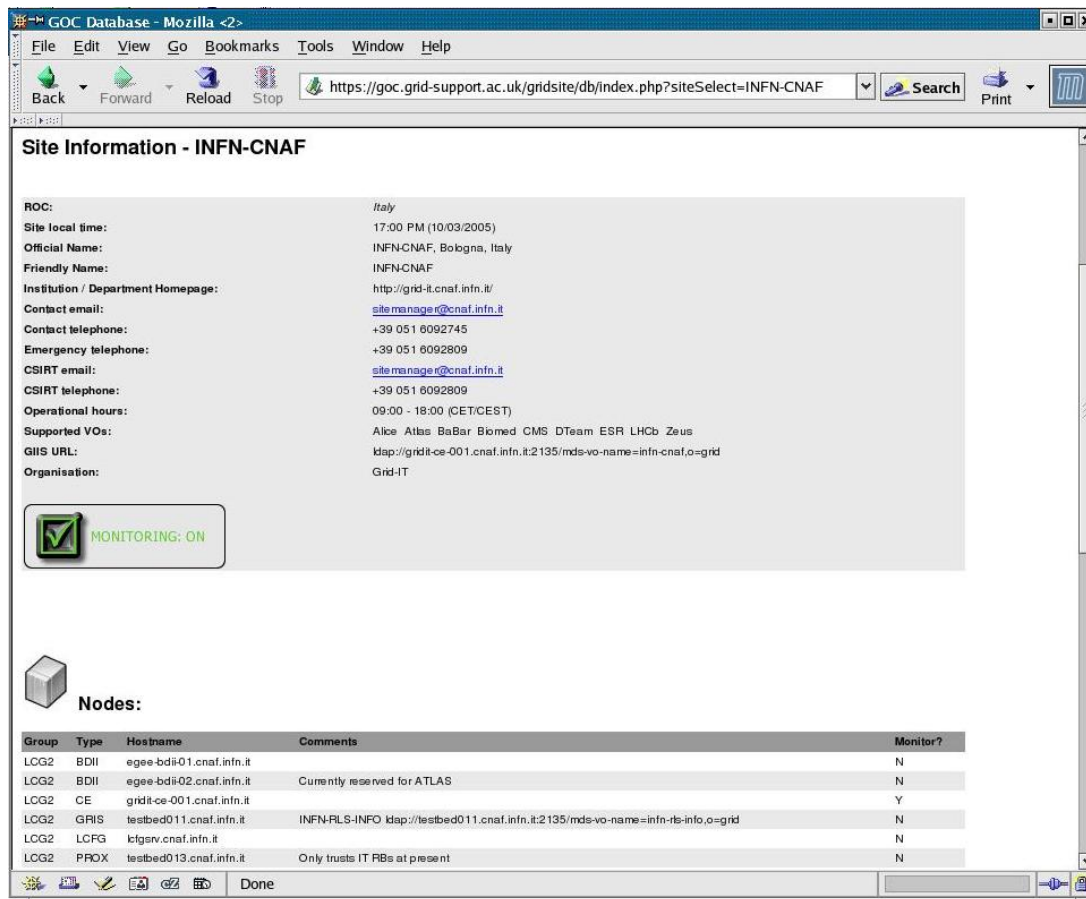
```
$ ldapsearch -LLL -x -h lxn1181.cern.ch -p 2135 -b "mds-vo-name=local,o=grid" \  
'objectclass=SiteInfo' siteName  
  
dn: in=lxn1181.cern.ch/siteinfo,Mds-Vo-name=local,o=grid  
siteName: CERN-LCG2
```

#### 5.1.4. The Site GIIS/BDII

At each site, a site GIIS or BDII collects information about all resources present at a site (i.e. data from all GRISes of the site). Site BDIIs are preferred to site GIISes and are the default in LCG-2 releases. In this section we explain how to query a site GIIS/BDII.

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

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



**Site Information - INFN-CNAF**

ROC: Italy

Site local time: 17:00 PM (10/03/2005)

Official Name: INFN-CNAF, Bologna, Italy

Friendly Name: INFN-CNAF

Institution / Department Homepage: <http://grid-it.cnaf.infn.it/>

Contact email: [site.manager@cnaf.infn.it](mailto:site.manager@cnaf.infn.it)

Contact telephone: +39 051 6092745

Emergency telephone: +39 051 6092809

CSIRT email: [site.manager@cnaf.infn.it](mailto:site.manager@cnaf.infn.it)


CSIRT telephone: +39 051 6092809

Operational hours: 09:00 - 18:00 (CET/CEST)

Supported VOs: Alice Atlas BaBar Blommed CMS DTeam ESR LHCb Zeus

GIIS URL: <kmap://gridit-ce-001.cnaf.infn.it:2135/mds-vo-name=infncnaf,o=grid>

Organisation: Grid-IT

 MONITORING: ON

**Nodes:**

| Group | Type | Hostname                   | Comments  | Monitor? |
|-------|------|----------------------------|---|----------|
| LCG2  | BDII | egee-bdi01.cnaf.infn.it    |   | N        |
| LCG2  | BDII | egee-bdi02.cnaf.infn.it    | Currently reserved for ATLAS  | N        |
| LCG2  | CE   | gridit-ce-001.cnaf.infn.it |   | Y        |
| LCG2  | GRIS | testbed011.cnaf.infn.it    | INFN-RLS-INFO <a href="kmap://testbed011.cnaf.infn.it:2135/mds-vo-name=infncnaf,o=grid">kmap://testbed011.cnaf.infn.it:2135/mds-vo-name=infncnaf,o=grid</a> | N        |
| LCG2  | LCFG | lctgen.cnaf.infn.it        |   | N        |
| LCG2  | PPOX | testbed013.cnaf.infn.it    | Only trusts IT PBs at present   | N        |

Figure 8: The status page of the INFN-CNAF site

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



a look to the following URL:

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

You will retrieve the information shown in Figure 8, the GIIS URL is `ldap://gridit-ce-001.cnaf.infn.it:2135/mds-vo-name=infncnaf,o=grid`. In this case, the site still has a site GIIS. In order to interrogate it, we can use the command shown in the following example:

#### ***Example 5.1.4.1 (Interrogating the site BDII)***

```
$ ldapsearch -x -H ldap://lcgce02.ifaef.es:2170 -b "mds-vo-name=piclchg2,o=grid"

version: 2

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

# https://edt003.cnaf.infn.it:7772, infncnaf, grid
dn: GlueServiceURI=https://edt003.cnaf.infn.it:7772,Mds-Vo-name=infncnaf,o=grid
   id
objectClass: GlueService
objectClass: GlueSchemaVersion
GlueServiceURI: https://edt003.cnaf.infn.it:7772
GlueServiceAccessPointURL: https://edt003.cnaf.infn.it:7772
GlueServiceType: ResourceBroker
GlueServicePrimaryOwnerName: LCG
GlueServicePrimaryOwnerContact: mailto:sitemanager@cnaf.infn.it
GlueServiceHostingOrganization: INFN-CNAF
GlueServiceMajorVersion: 1
GlueServiceMinorVersion: 00

[...]

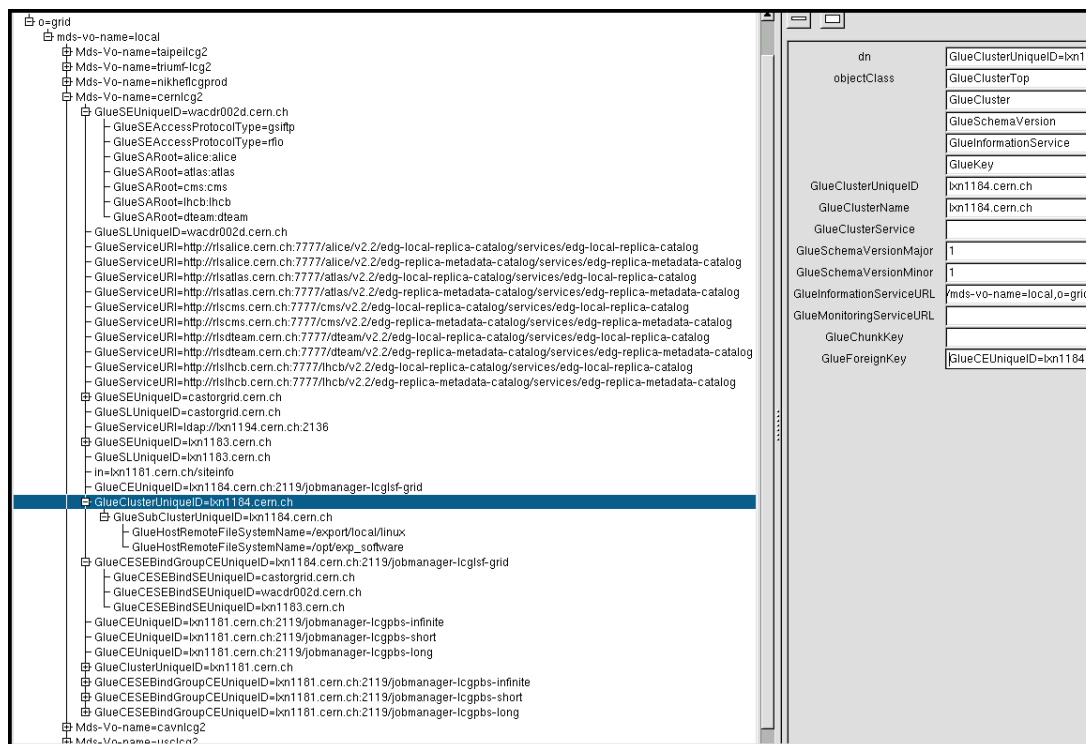
# gridit-ce-001.cnaf.infn.it:2119/jobmanager-lcgpbs-cert, infncnaf, grid
dn: GlueCEUniqueID=gridit-ce-001.cnaf.infn.it:2119/jobmanager-lcgpbs-cert, Mds
   -Vo-name=infncnaf,o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
objectClass: GlueCEInfo
objectClass: GlueCEPolicy
objectClass: GlueCEState
objectClass: GlueInformationService
```

```
objectClass: GlueKey
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1
```

[...]

### 5.1.5. The top BDII

A top BDII collects all information coming from site GIISes/BDIIs and stores them in a permanent database. The top BDII can be configured to get published information from resources in all sites, or only from some of them. In order to find out the location of a top BDII in a site, you can consult the GOC page of this site. The BDII will be listed with the rest of the nodes of the site (refer to Figure 8).



The screenshot shows an LDAP directory browser interface. On the left, a tree view displays a hierarchy of nodes under the root 'o=grid'. The selected node is 'GlueClusterUniqueID=bxn1184.cern.ch'. On the right, a detailed view of this node is shown, listing various attributes and their values.

|                           |                                     |
|---------------------------|-------------------------------------|
| dn                        | GlueClusterUniqueID=bxn1184.cern.ch |
| objectClass               | GlueClusterTop                      |
|                           | GlueCluster                         |
|                           | GlueSchemaVersion                   |
|                           | GlueInformationService              |
|                           | GlueKey                             |
| GlueClusterUniqueID       | bxn1184.cern.ch                     |
| GlueClusterName           | bxn1184.cern.ch                     |
| GlueClusterService        |                                     |
| GlueSchemaVersionMajor    | 1                                   |
| GlueSchemaVersionMinor    | 1                                   |
| GlueInformationServiceURL | /mds-vo-name=local,o=grid           |
| GlueMonitoringServiceURL  |                                     |
| GlueChunkKey              |                                     |
| GlueForeignKey            | GlueCEUniqueID=bxn1184.cern.ch      |

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

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

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





In Figure 9, a view of the DIT of the BDII of the LCG-2 is shown. In the figure, only the sub-tree that corresponds to the CERN site is expanded. The DN for every entry in the DIT is shown. Entries for storage and computing resources, as well as for the bindings between CEs and SEs can be seen in the figure.

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

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

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

#### **Example 5.1.5.1 (Interrogating a BDII)**

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

```
$ ldapsearch -x -LLL -H ldap://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=cernlclg1,Mds-Vo-name=lcgeast,Mds-Vo-name=local,o=grid  
GlueCESEBindCEUniqueID: adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite  
[...]
```

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

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

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

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

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

```
$ ldapsearch -LLL -h 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
```



```
GlueSASStateAvailableSpace: 1464065168
GlueSASStateUsedSpace: 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
GlueSASStateAvailableSpace: 1000000000000
GlueSASStateUsedSpace: 1000000000000
GlueChunkKey: GlueSEUniqueID=wacdr002d.cern.ch
[...]
```

## 5.2. R-GMA

As explained in section 3.2.4, R-GMA is called to be the replacement of MDS in the future. Moreover, R-GMA is already being used to publish information and users can get it through a Command Line Interface included in any UI or through a web interface. Many applications already use it, specially for accounting and monitoring purposes.

R-GMA can be queried using a CLI, one of the APIs or the web R-GMA web interface. Some (non-exhaustive) information on each of this is given in this section. For further information refer to [R14].

### 5.2.1. R-GMA Browser

The *R-GMA browser* is installed in the machine running the registry and the schema. It allows the user to easily navigate the schema (what tables are available, what is their definition), see all available producers for a table and query the (selected) producers. All this can be achieved using a web interface.

Figure 10 shows this R-GMA browser web interface. It is accessible on the following web page:

```
http://lcgic01.gridpp.rl.ac.uk:8080/R-GMA/index.html
```

The user can easily perform any query and retrieve the corresponding information just clicking in the corresponding tables and attributes.

### 5.2.2. R-GMA CLI

An R-GMA CLI is available in every UI. This interface allows the user to perform queries to the registry and also to add new information using the SQL language. It includes a consumer and can initiate a primary producer and a secondary one.

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

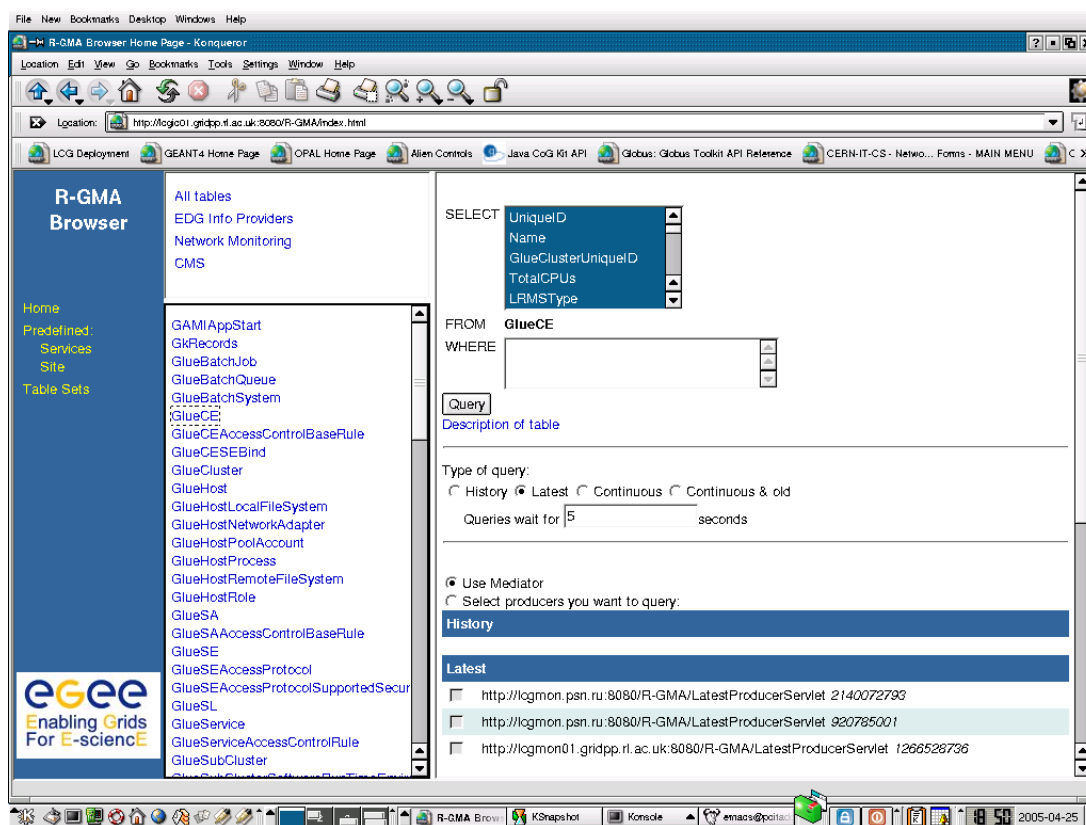


Figure 10: The R-GMA Web Interface

```
rgma -c "select Name from Site where Name='RALPP-LCG'"
+-----+
| Name          |
+-----+
| RALPP-LCG    |
+-----+
1 Rows in set
```

If only `rgma` is invoked, an interactive session is started. `gets` into this interface by just typing `rgma` in any UI:

```
Welcome to the R-GMA virtual database for Virtual Organisations.
You are connected to the following R-GMA registry services:
```

```
http://lcgic01.gridpp.rl.ac.uk:8080/R-GMA/RegistryServlet
```

Type "help" for a list of commands.



```
rgma> select Name from Site where Name='RALPP-LCG'
+-----+
| Name          |
+-----+
| RALPP-LCG     |
+-----+
1 Rows in set
```

As it is shown, the CLI is pointing to the main registry, which holds pointers to all the R-GMA producers for all the sites and VOs. Our queries will get the information from the appropriate producers wherever they are located.

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

| Command   | Description  |
|---|--|
| <code>help [&lt;command&gt;]</code>                                   | Information (general or about command)                             |
| <code>exit / quit</code>  | Exit the R-GMA command line  |
| <code>show [tables   producers of &lt;table&gt;]</code>               | Show the tables in the schema, the producers of a given table      |
| <code>describe &lt;table&gt;</code>                                   | Show column names and types for specified table                    |
| SQL select  | Query R-GMA  |
| <code>set query latest   continuous   historical</code>               | Set type of query  |
| SQL insert  | Insert tuple into the primary producer                             |
| <code>Secondaryproducer &lt;table&gt;</code>                          | Declare table to be consumed and republished by secondary producer |
| <code>set [secondary]producer latest   continuous   historical</code> | Set supported type for the producer or the secondary producer      |
| <code>set [timeout   maxage] &lt;timeout&gt; [&lt;units&gt;]</code>   | Set timeout for queries or maximum age of tuples to return         |

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



### Example 5.2.2.1 (Querying the Information System)

Inside the interface the user can easily perform any query to the registry using the SQL syntax:

```
rgma> set query continuous
Set query type to continuous

rgma> set maxage 1 days
Set max age to 1.000000 days

rgma> set timeout 10 seconds
Set timeout to 10.000000 seconds

rgma> select UniqueID, TotalCPUs from GlueCE
```

| UniqueID  | TotalCPUs |
|---|-----------|
| ce00.inta.es:2119/jobmanager-lcgpbs-atlas         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-alice         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-lhcb          | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-cms           | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-dteam         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-sixt          | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-biomed        | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-swetest       | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-atlas         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-alice         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-lhcb          | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-cms           | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-dteam         | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-sixt          | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-biomed        | 5         |
| ce00.inta.es:2119/jobmanager-lcgpbs-swetest       | 5         |
| node001.grid.auth.gr:2119/jobmanager-lcgpbs-atlas | 21        |
| node001.grid.auth.gr:2119/jobmanager-lcgpbs-alice | 21        |
| node001.grid.auth.gr:2119/jobmanager-lcgpbs-lhcb  | 21        |
| [ ... ]   |           |
| ce.ulakbim.gov.tr:2119/jobmanager-lcgpbs-seegrid  | 16        |

In this example, we first set the type of query to continuous. That is, new tuples are received as they are published and will not terminate unless the user aborts or a maximum time for the query is indicated. Then, the maximum age of tuples to return is the defined. Afterwards, we define a timeout for the query of 10 seconds. Finally, we query for the Id and the number of CPUs of all CEs publishing information into R-GMA.



### 5.2.3. R-GMA API

There exist R-GMA APIs in Java, C, C++ and Python. They include methods for creating consumers, as well as primary and secondary producers; setting type of queries, type of produces, retention periods, time outs; retrieving tuples, inserting data.

By using the APIs the user can create his own producer/consumer, to publish/consume the information he is interested in.

Documentation exists for all APIs, including example code. Please check [R14].

## 5.3. MONITORING

The ability to monitor resource related parameters is currently considered a necessary functionality in any network. In such an heterogeneous and complex system as the Grid, this necessity becomes fundamental. A proper monitoring system permits the existence of a central point of operational information (i.e.: in 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. Apart from R-GMA that was explained previously, several different monitoring tools can be used, including general purpose monitoring tools and Grid specific systems like GridIce [R18].

Also important are the web pages publishing the results of functional tests applied periodically to the all the sites registered within LCG-2. The results of this tests show if a site is responding correctly to standard Grid operations; otherwise, an investigation on the cause of the unexpected results is undertaken. Some VOs may even decide to automatically exclude from their BDII the sites that are not passing the functional tests successfully, so that they do not appear in the IS and are not considered for possible use by their applications.

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

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

<http://lcg-testzone-reports.web.cern.ch/lcg-testzone-reports/cgi-bin/lastreport.cgi>

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

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



### 5.3.1. GridIce

The GridIce monitoring service is structured in a five layer architecture. The resource information is obtained from the 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 [R18] for details on this.

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

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

The GridIce web page that shows the monitoring information for 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.

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





## 6. WORKLOAD MANAGEMENT

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.

### 6.1. 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* [R19] defined by the *Condor Project* [R20], 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 [R21] and [R22].

The JDL syntax consists on statements 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"};
```

Some JDL attributes allow the user to specify requirements about input data that his job will use and also to express that some output data will be produced. This attributes are the following: `InputData`, `DataAccessProtocol`, `OutputSE`, `OutputData`, `OutputFile`, `LogicalFileName`, `StorageElement` and `ReplicaCatalog`.



All these data requirement attributes are described later on in section 7.8

## Imposing Constraints on the CE

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

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

### *Example 6.1.1 (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 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";
```

**Note:** As explained in 6.2.7, normally the condition that a CE is in production state is automatically added to the requirements clause. Thus, CEs that do not correctly publish this will not match. This condition is, nevertheless, configurable.

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

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

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

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



### **Example 6.1.2** (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 6.1.3** (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 [R22].

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<sup>3</sup>:

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

### **Example 6.1.4** (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 before 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 ) ;
```

<sup>3</sup>The function used to calculate the available space in a SE can be inaccurate if the SE uses NFS mounted file systems. 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 `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.

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 number of free CPUs is used.

## 6.2. 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].



### 6.2.1. 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 6.2.1.1 (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 canceling it. The format of the jobId is:

```
https://lserver_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<sup>4</sup>, he 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:

```
**** 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 match making functionality (see Section 6.2.3) will not be carried out. That is, the `BrokerInfo` file, which provides information about the evolution of the job, will not be created.

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

---

<sup>4</sup>and currently this must be the case.



<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 6.2.1.2 (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`.





## 6.2.2. 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 6.2.2.1 (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 shown, along with the time when that status was reached, and the reason for being in that state (which may be especially helpful for the ABORTED state). The possible states in which a job can be found were introduced in Section 3.3.1, and are summarized in Appendix C. Finally, the `destination` field contains the ID of the CE where the job has been submitted.

Much more information is provided if the verbosity level is increased by using `-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/E9R0Y14J7qgsq7FYTnhmsA
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 instead, the status of all the jobs owned by the user submitting the command is retrieved. As the number of jobs owned by a single user may be large, there are some options that limit that job selection. The `--from / --to [MM:DD:]hh:mm[:[CC]YY]` options make the command query LB for jobs that were submitted after/before the specified date and time. The `--status <state>` (`-s`) option makes the command retrieve only the jobs that are in the specified state, and the `--exclude <state>` (`-e`) option makes it retrieve jobs that are not in the specified state. This two lasts options are mutually exclusive, although they can be used with `--from` and `--to`.

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

```
$ 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 6.2.2.2** (Canceling a job)

A job can be canceled 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/C6n5Hqlex9-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 successful cancellation is returned to the user. In such cases, just cancel the job again when its status is SCHEDULED or RUNNING.

### **Example 6.2.2.3** (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/snPegplYMJcnS22yF5pFlg
```

```
Retrieving files from host lxshare0234.cern.ch
```

```
*****
```

```
JOB GET OUTPUT OUTCOME
```

```
Output sandbox files for the job:
```

```
- https://lxshare0234.cern.ch:9000/snPegplYMJcnS22yF5pFlg  
have been successfully retrieved and stored in the directory:  
/tmp/jobOutput/snPegplYMJcnS22yF5pFlg
```

```
*****
```

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`.

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

### **Example 6.2.2.4** (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
[...]
```

### 6.2.3. The BrokerInfo

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

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

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

Detailed information about the `BrokerInfo` file, the `edg-brokerinfo` CLI, and its respective API can be found in [R23].

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 `VirtualOrganisation 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 `$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 output of the submitted job, and retrieve it with the rest of generated output (by specifying the file in the Output Sandbox), when the job finishes. This can be done for example by including the following lines:

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

in a submitted shell script.

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

#### 6.2.4. 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 release, although it may be in the future. Any site installing or using it



will do it only under its own responsibility.

This section gives an overview of how interactive jobs should work in 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, which listens on a port for the job standard streams. Moreover, the `edg-job-submit` command opens a new window where the incoming job streams are forwarded. The port on which the shadow process listens is assigned by the Operating System (OS), but can be forced through the `ListenerPort` attribute in the JDL.

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

#### **Example 6.2.4.1** (Simple interactive job)

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

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

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

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

The `interactive.sh` script just presents a welcome message to the user, and then asks and waits for an input. After the user has entered a name, this is shown back just to check that the input was received correctly. Figure 11 shows the result of the program (after the user has entered his name) in the generated X window.

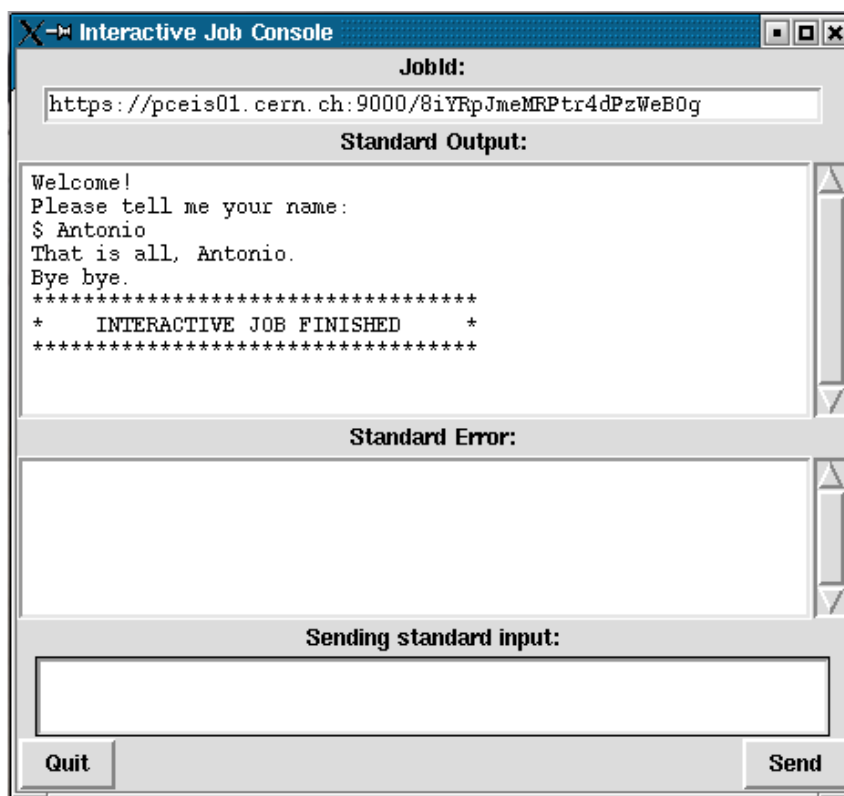


Figure 11: X window for an interactive job

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 his own tools. It is important to note that when this option is specified, the UI has no more control over the launched listener process that has hence to be killed by the user (through the returned process id) when the job is finished.

**Example 6.2.4.2** *(Interacting with the job through a bash script)*

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

As arguments, the script accepts the names of the three pipes (input, output, and error) that the job will use, and the process id (pid) of the listener process. All this information is returned when submitting the job, as can be seen in the returned answer for the submission of the same `interactive.jdl` and





interactive.sh used before:

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

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

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

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

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

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

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

The code of `dialog.sh` follows:

```
#!/bin/bash

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



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

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

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

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

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

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

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

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

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

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

```
-----
```



---

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

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

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

### 6.2.5. 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 release. Any site installing or using it will do it only under its own responsibility.

This section gives a brief overview of how checkpointable jobs should work in 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 an intermediate state (e.g., after a crash), the `--chkpt <state_file>` option may be used, where `state_file` must be a valid JDL file, where the state of a previously submitted job was saved. In this way, the job will first load the given state and then continue running until it finishes. That JDL job state file can be obtained by using the `edg-job-get-chkpt <jobid>` command.



## 6.2.6. MPI Jobs

*Message Passing Interface (MPI)* applications are run in parallel on several processors. Some LCG-2 sites support jobs that are run in parallel using the MPI standard. It is not mandatory for LCG-2 clusters to support MPI, so those clusters who do must publish this in the IS. They do so by adding the value "MPICH" to the `GlueHostApplicationSoftwareRunTimeEnvironment` GLUE attribute. User's jobs can then look for this attribute in order to find clusters that support MPI. This is done transparently for the user by the use of a requirements expression in the JDL file, as shown later.

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

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

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

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

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

**Attention:** The executable that is specified in the JDL must not be the MPI application directly, but a wrapper script that invokes this MPI application by calling `mpirun`<sup>5</sup>. This allows the user to perform some pre- and post-execution steps in the script. This usually includes also the compilation of the MPI application, since the resulting binary may be different depending on the MPI version and configuration.

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

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

<sup>5</sup>This is so because the job type that is used at the globus gatekeeper level is `multiple`. If you bypass the LCG middleware and submit a job using globus directly, and if you specify `mpi` as the job type, then globus calls `mpirun` directly on the specified executable. This is rather limiting because no pre- or post-MPI activity can be performed.

The following example shows a complete example of MPI job.

### ***Example 6.2.6.1 (MPI job submission)***

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

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

The JDL file would be:

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

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

```
#!/bin/sh -x

# Binary to execute
EXE=$1

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



```
echo "As:          " `whoami`

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

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

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

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

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

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

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

In the script, the MPI application is first compiled, then the list of hosts where to run is created by reading from the appropriate batch system information. The variable `CPU_NEEDED` stores the number of



nodes that are available. The script also checks that ssh works for all listed nodes. This step should not be required but it is a good safety measure to detect misconfigurations in the site and avoid future problems. Finally, `mpirun` is called with the `-np` and `-machinefile` options specified.

The retrieved output of a job execution follows:

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

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

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



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

### 6.2.7. Advanced Command Options

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

The main configuration file is located 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.

Among others, it must include a requirements clause, and that condition is added to the requirements set by the user. This condition is normally `other.GlueCEStateStatus == "Production"`, so that CEs that are in funny states (e.g. Closed) do not match.

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 `$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf`, consisting essentially in the list of Network Servers, Proxy Servers and LB servers to be used when submitting jobs. A different file can be specified using the variable `$EDG_WL_UI_CONFIG_VO` or the `--config-vo <file>` option of the `edg-job-*` commands. This can be useful for example to use a RB that is different from the default one.

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





### **Example 6.2.7.1** (Changing the default VO)

A user can change his 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)
```

### **Example 6.2.7.2** (Using several RBs)

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

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

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

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

## **6.3. 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 [R24] for a complete description of the functionalities provided by the GUI, together with some example screen shots.

## 7. DATA MANAGEMENT

### 7.1. INTRODUCTION

This chapter describes the client tools that are available to deal with the data in LCG-2, both directly from a UI or via a job from a WN. The also important Data Management APIs will be described in more depth in [R5]. Nevertheless, a complete overview of the available APIs is given in Appendix F.

**NOTE:** In this chapter, several examples will be shown. The commands entered by the user are leaded by a '\$' symbol, and the answers of the shell are usually preceded by '>' (unless the difference is obvious).

### 7.2. STORAGE ELEMENTS

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

#### 7.2.1. Data Channel Protocols

The data access protocols supported in current LCG-2 are summarized in the next table:

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

In the current LCG-2 release, every SE must have a **GSIFTP** server [R25]<sup>6</sup>. The GSIFTP protocol offers basically the functionality of FTP, i.e. the transfer of files, but enhanced to support GSI security. It is responsible for secure, fast and efficient file transfers to/from Storage Elements. It provides third party control of data transfer as well as parallel streams data transfer.

GSIFTP is currently supported by every Grid SE and it is thus the main file transfer protocol in LCG-2. However, for the remote access (and not only copy) of files stored in the SEs, the protocols to

<sup>6</sup>In the literature, the terms *GridFTP* and *GSIFTP* are sometimes used interchangeably. Strictly speaking, GSIFTP is a subset of GridFTP. Please refer to [R25] for more information.



be used are the *Remote File Input/Output protocol (RFIO)* [R26] and the *GSI dCache Access Protocol (gsidcap)*.

RFIO was developed to access tape archiving systems, such as CASTOR (CERN Advanced STORAGE manager)<sup>7</sup>. The insecure version does not support GSI yet and therefore can only be used to access data within a Local Area Network (LAN) and from a WN (not the UI); the only authentication is through UID and GID. The secure RFIO on the contrary is completely GSI enabled, uses Grid certificates and can be used for remote file access to a distant site and also from a UI.

There is some more information about RFIO in F.

The gsidcap protocol is the GSI secure version of the dCache<sup>8</sup> access protocol, *dcap*. Being GSI-secure, gsidcap can be used for inter-site remote file access.

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

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

### 7.2.2. The Storage Resource Manager interface

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

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

Unfortunately, at the moment, not all SEs implement the same version of the SRM interface, and none of these versions offers all of the functionalities that the SRM standard ([R11]) defines. The high level Data Management tools and APIs will, in general, interact transparently with an SRM.

<sup>7</sup>A CERN hierarchical Storage Manager [R27]

<sup>8</sup>A storage Manager developed by the Deutsches Elektronen-Synchrotron (DESY) and the Fermi National Accelerator Laboratory (FERMI). More information in [R28]

### 7.2.3. Types of Storage Elements

There are different types of possible SEs in LCG-2.

- **Classic SE:** it consists of a GridFTP server and an insecure RFIO daemon (*rfiod*) in front of a physical single disk or disk array. The GridFTP server supports *secure* data transfers. The *rfiod* daemon ensures LAN-limited file access through RFIO. If the same Classic SE serves multiple Virtual Organisations, the only way to allocate disk quota per VO is through physical partitioning of the disk, which is an option site managers in general do not like. In other words, a single VO might fill up the entire SE. Once again, it is user responsibility to monitor disk usage in the case of a Classic SE. Furthermore, the classic SE **does NOT support the SRM interface** (and never will).
- **Mass Storage System:** it consists of a *Hierarchical Storage Management (HSM)* system for files that may be migrated between front-end disk and back-end tape storage hierarchies. The migration of files between disk and tape storage is managed by a *stager* process. The stager manages one or more disk pools or groups of one or more UNIX filesystems, residing on one or more disk servers. The stager is responsible for space allocation and for file migration between the disk pool and tape storage. The MSS exposes to the user a virtual file system which hides the complexity of the internal details.

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

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

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

- **dCache Disk pool manager:** it consists of a dCache server and one or more pool nodes. The server represents the single point of access to the SE and presents files in the pool disks under a single virtual filesystem tree. Nodes can be dynamically added to the pool. File transfer is managed through GridFTP while the native *gsidcap* protocol allows POSIX-like data access. It presents an SRM interface which allows to overcome the limitations of the Classic SE.
- **LCG Disk pool manager:** is the LCG lightweight alternative to dCache; easy to install and, although not so powerful as dCache, offers all the functionality required by small sites. Disks



can be added dynamically to the pool at any time. Like in dCache, a virtual file system hides the complexity of the disk pool architecture. The LCG DPM includes a GridFTP server for file transfer and ensures file access through secure RFIO. It also presents an SRM interface. In addition, disk quota allocation per VO is supported. For these reasons, once the DPM is deployed, it will replace the Classic SE. Old Classic SEs will be converted to DPMs with only one disk in the pool.

### 7.3. FILES NAMING CONVENTION IN LCG-2

As an extension of what was introduced in Chapter 3, the different types of names that can be used within the LCG-2 files catalogs are summarized below:

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

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

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

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

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

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

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

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

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

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

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

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



---

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

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

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

where `<protocol>` must be a valid protocol (supported by the SE) to access the contents of the file (GSIFTP, RFIO, gsidcap), and the string after the double slash may have any format that can be understood by the SE serving the file. While SURLs are in principle invariable (they are entries in the file catalog, see Section 7.4), TURLs are obtained dynamically from the SURL through the Information System or the SRM interface (for SRM managed SEs). The TURL therefore can change with time and should be considered only valid for a relatively small period of time after it has been obtained.

#### 7.4. FILE CATALOGS IN LCG-2

Users and applications need to locate files (or replicas) on the Grid. The **File Catalog** is a service which fulfills such requirement, maintaining mappings between LFN(s), GUID and SURL(s).

In LCG-2, two types of file catalogs are currently deployed: the old **Replica Location Server (RLS)** and the new **LCG File Catalog (LFC)**. Both of them are deployed as centralized catalogs. The catalogs publish their **endpoints** (service URL) in the Information Service so that the LCG Data Management tools and any other interested services (the RB for example) can find the way to them (and then to the Grid files information). Be aware that for the RLS there are two different endpoints (one for LRC and one for RMC) while for LFC, being it a single catalog, there is only one.

The user can decide which catalog to use setting the environmental variable `LCG_CATALOG_TYPE` equal to `edg` for RLS or `lfc` for the LFC.

**Attention:** the RLS and LFC are not respectively mirrored. Entries in the LFC will not appear in RLS and viceversa. Choose to use any of the two catalogs but be consistent with your choice.

The RLS in fact it consists of two catalogs: the **Local Replica Catalog (LRC)** and the **Replica Metadata Catalog (RMC)**. See Figure 12

The LRC keeps mappings between GUIDs and SURLs, while the RMC keeps mappings between GUIDs and LFNs. Both RMC and LRC support the use of metadata. User metadata should all be confined in RMC while LRC should contain only system metadata (file size, creation date, checksum etc.).

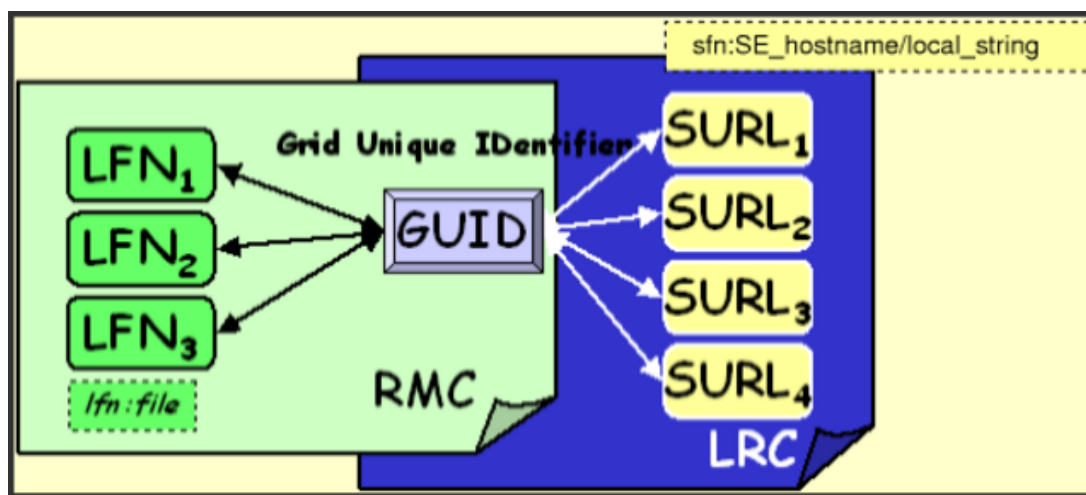


Figure 12: Architecture of RLS

The LFC was developed to overcome some serious performance and security problems of the old RLS catalogs; it also adds some new functionalities such as transactions, roll-backs and a hierarchical namespace for LFNs. It consists of a unique catalog, where the LFN is the main key, see Figure 13. Further LFNs can be added as symlink to the mail LFN. System metadata is supported while for user metadata only a single string entry is available (rather a *description field* than real metadata).

LFC should become the primary choice for any VO deploying a new file catalog. Not only that, but, since migration tools from RLS to LFC have been provided, most existing VOs should consider moving from RLS catalogs to the new LFC.

**IMPORTANT:** A file is considered to be a *Grid file* if it is **both** physically present in a SE **and** registered in the file catalog. In this chapter several tools will be described. In general high level tools like `lcf-utils` (see Sec. 7.7.1) will ensure consistency between files in the SEs and entries in the File Catalog. However, usage of low level tools, for both data transfer and catalog entries management could cause inconsistencies between SEs physical files and catalog entries; what would imply the corruption of GRID files. This is why the usage of low level tools is strongly discouraged unless really necessary.

## 7.5. LFC INTERACTION COMMANDS

In general terms, the user should usually interact with the file catalog through high level utilities (`lcf-utils`, see Section 7.7.1). The CLIs and APIs that are available for catalog interaction provide further functionality and more fine-grained control for the operations with the catalog. In some situations, they represent the only possible way to achieve the desired functionality with the LFC.

### Environment:



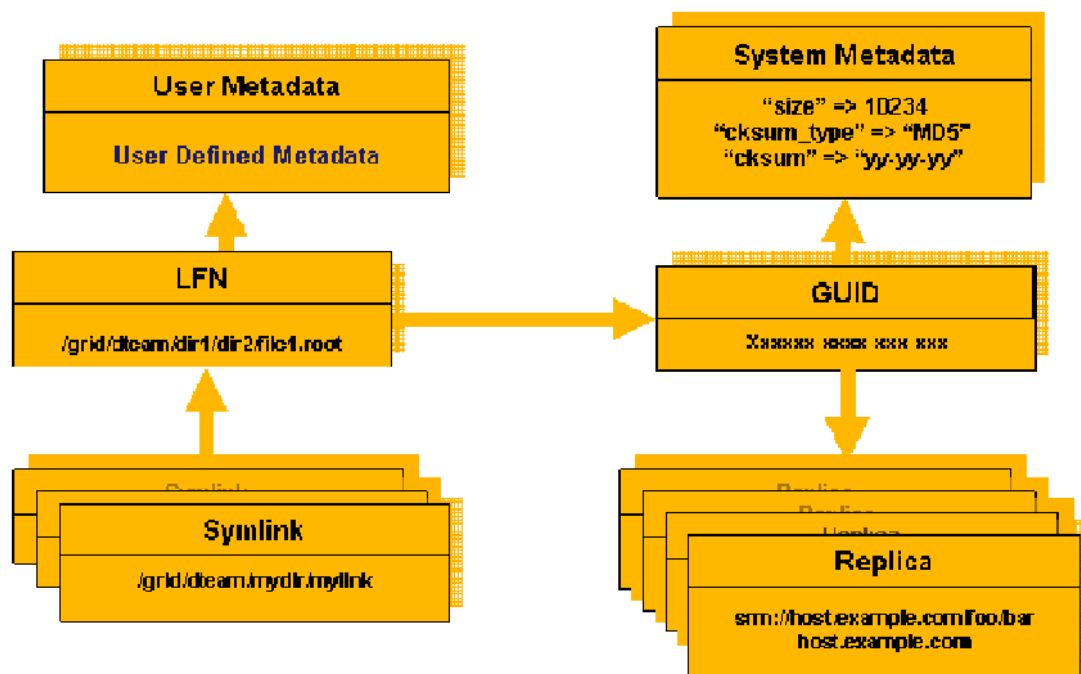


Figure 13: Architecture of the LFC

1. The variable `$LCG_CATALOG_TYPE` must be set to `lfc`
2. the variable `$LFC_HOST` must be set to hold the hostname of the LFC server

**Attention:** For GFAL and the `lcg-utils`, the `$LFC_HOST` variable is only another way to define the location of the LFC, but for the `lfc-*` commands the variable is required, since these tools do not use the information published in the IS.

The directory structure of the LFC namespace initiates with the `grid` directory. Under this, there is a directory for each one of the supported VOs. Users of a VO will have read and write permissions only under the directory of their VO (e.g.: `/grid/beam` for users of VO Biomed). If such a directory does not exist, then this means that this LFC server does not support that VO.

Once the correct environment has been set, the following commands can be used:



---

|                             |  |
|-----------------------------|--|
| <code>lfc-chmod</code>      | Change access mode of a LFC file/directory.          |
| <code>lfc-chown</code>      | Change owner and group of a LFC file/directory.      |
| <code>lfc-delcomment</code> | Delete the comment associated with a file/directory. |
| <code>lfc-getacl</code>     | Get file/directory access control lists.             |
| <code>lfc-ln</code>         | Make a symbolic link to a file/directory.            |
| <code>lfc-ls</code>         | List file/directory entries in a directory.          |
| <code>lfc-mkdir</code>      | Create directory.                                    |
| <code>lfc-rename</code>     | Rename a file/directory.                             |
| <code>lfc-rm</code>         | Remove a file/directory.                             |
| <code>lfc-setacl</code>     | Set file/directory access control lists.             |
| <code>lfc-setcomment</code> | Add/replace a comment.                               |

Manpages are available for all the commands. Most of the commands work in a very similar way to their Unix equivalents, but operating on directories and files of the catalog namespace. Where the path of a file/directory is required, an absolute path can be specified (starting by /) or, otherwise, the path is prefixed by the contents of the `$LFC_HOME` environment variable.

Many of the commands provide administrative functionality. Their names indicate what their functions are. Users should use these commands carefully, keeping in mind that the operations they are performing affect the catalog, but not the physical files that the entries represent. This is especially true in the case of the `lfc-rm` command, since it can be used to remove the LFN for a file, but it does not affect the physical files. If all the LFNs pointing to a physical Grid file are removed, then the file is no longer visible to LCG-2 users (although it may still be accessed by using its SURL directly).

### ***Example 7.5.1 (Creating directories in the LFC)***

Users cannot implicitly create directories in the LFNs namespace when registering new files with LCG Data Management tools (see for example the `lcg-cr` command in Section 7.7.1). Directories must be created in advance using the `lfc-mkdir` command. This is currently the only main use of the `lfc-*` commands for the average user. For other tasks, the `lcg-utils` should be normally used.

```
$ lfc-mkdir /grid/dteam/MyExample

$ lfc-ls -l -d /grid/dteam/MyExample
drwxr-xrwx  6 delgadop cg          0 Mar 24 16:00 /grid/dteam/MyExample
```

### ***Example 7.5.2 (Listing the entries of a LFC directory)***

The `lfc-ls` command lists the LFNs of a directory. The command supports several options among which `-l` for long format and `--comment` for showing user-defined metadata information.



**Attention:** The `-R` option, for recursive listing, is also available for the command, but **it should not be used**. It is a very expensive operation on the catalog and should be avoided.

In the following example the directory `/grid/dteam/MyExample` and its subdirectory `day1` are listed (after being populated).

```
$ lfc-ls /grid/dteam/MyExample
```

```
/grid/dteam/MyExample:  
day1  
day2  
day3  
day4  
interesting
```

```
$ lfc-ls /grid/dteam/MyExample/day1
```

```
/grid/dteam/MyExample/day1:  
measure1  
measure2  
measure3  
measure4
```

### ***Example 7.5.3 (Creation of symbolic links)***

The `lfc-ln` may be used to create a symbolic link to a file. In this way two different LFNs will point to the same file.

In the following example, we create a symbolic link `/grid/dteam/MyExample/interesting/file1` to the original file `/grid/dteam/MyExample/day1/measure2`

```
$ lfc-ln -s /grid/dteam/MyExample/day2/measure2 /grid/dteam/MyExample/interesting/file1
```

Now we can check that link was created with a long listing.

```
$ lfc-ls -l /grid/dteam/MyExample/interesting  
> lrwxrwxrwx 1 delgadop cg 0 Mar 24 15:46 file1 -> /grid/dteam/MyExample/day2/measure2
```

Remember that links created with `lfc-ln` are soft. If the LFN they are pointing to is removed, the links themselves are not deleted, but keep existing as broken links.



### **Example 7.5.4** (Adding metadata information to LFC entries)

The `lfc-setcomment` and `lfc-delcomment` commands allow the user to associate a comment with a catalog entry and delete that comment respectively. This is the only user-defined metadata that can be associated with catalog entries. The comments for the files may be listed using the `--comment` option of the `lfc-ls` command. This is shown in the following example:

```
$ lfc-setcomment /grid/dteam/MyExample/interesting/file1 "Most promising measure"

$ lfc-ls --comment /grid/dteam/MyExample/interesting/file1
> /grid/dteam/MyExample/interesting/file1 Most promising measure
```

### **Example 7.5.5** (Removing LFNs from the LFC)

As explained before, the `lfc-rm` will only delete catalog entries, but not physical files. In principle, the deletion of replicas (`lcg-del`) should be used instead. When the last replica is removed, the entry is also removed from the catalog. Indeed, the `lfc-rm` commands will not allow a user to remove an LFN for which there are still SURLS associated (i.e.: physical replicas exist).

That said, there might be some use cases of the command, being the most important one the deletion of directories, by using the `-r` option. This action should be of course performed only in rare occasions and probably by only certain people within a VO.

In the next example, we remove the previously created `trash` directory.

```
$ lfc-ls -l -d /grid/dteam/MyExample/trash
> drwxr-xrwx  0 dteam004 cg          0 Jul 06 11:13 /grid/dteam/MyExample/trash

$ lfc-rm /grid/dteam/MyExample/trash
> /grid/dteam/MyExample/trash: Is a directory

$ lfc-ls -l -d /grid/dteam/MyExample/trash
> /grid/dteam/MyExample/trash: No such file or directory
```

## **7.6. RLS INTERACTION 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 catalogs.

**WARNING!** Usage of RLS command line tools could not only cause inconsistencies between catalog entries and physical files in SEs, but also corruption of catalog entries themselves: a GUID-SURL



mapping could be removed from the LRC but a corresponding GUID-LFN might still exist in the RMC so that the alias exists but can not be resolved in a physical file location. Once again, in normal operation, a user should preferably use the high level LCG Data Management tools described in Sec. 7.7.1, 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.

**WARNING:** 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 failure of the command.

Only some usage examples of the most important commands will be given here. For detailed information please refer to [R29] and [R30].

### 7.6.1. Local Replica Catalog Commands

The `edg-lrc` commands handle 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).

All the commands require the LRC *endpoint*, which can be obtained using the `lcg-infosites` command (see Sec. 5.1.1) and which 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:

|                                  |  |
|----------------------------------|--|
| <code>addMapping guid pfn</code> | Adds the given mapping to the catalog.               |
| <code>pfnExists pfn</code>       | Checks whether the given PFN exists in the catalog.  |
| <code>guidExists guid</code>     | Checks whether the given GUID exists in the catalog. |
| <code>guidForPfn pfn</code>      | Returns the GUID for a given PFN.                    |
| <code>pfnForGuid guid</code>     | Returns the PFNs for a given GUID.                   |
| <code>removePfn guid pfn</code>  | Removes a PFN from a given GUID.                     |



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

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

Attribute management commands (metadata associated with GUID-URL mappings):

|   |   |
|---|---|
| <code>listAttrDefns</code>                      | Lists all attributes (name and type) that are defined in the LRC.               |
| <code>attrDefnExists name</code>                | Checks whether the given attribute exists.                                      |
| <code>getPfnAttr pfn attrName</code>            | Gets the given attribute value.   |
| <code>setPfnAttr pfn attrName value</code>      | Sets the given attribute's value.   |
| <code>removePfnAttr pfn attrName</code>         | Removes the given attribute.  |
| <code>mappingsByAttr attributeConditions</code> | 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, URL and LFN assigned to:

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

In addition, some fake entries (not really in the catalog) are defined:

```
$ setenv GUID2 guid:c06a92ee-6911-11d8-a453-000000000000
$ setenv URL2 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 7.6.1.1 (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-d9claf867039'

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

### **Example 7.6.1.2 (Retrieving SURLs and GUIDs)**

Retrieving the GUID for a SURL.

```
$ edg-lrc guidForPfn $SURL --endpoint $LRC_ENDPOINT
> guid:c06a92ee-6911-11d8-a453-d9claf867039
```

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-a453-000000000000'
```

### **Example 7.6.1.3 (Retrieving GUIDs for a SURL pattern)**

```
$ edg-lrc mappingsByPfn '*my_test*' --endpoint $LRC_ENDPOINT
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30,
srm://wacdr002d.cern.ch/castor/cern.ch/grid/dteam/my_test3
> guid:c06a92ee-6911-11d8-a453-d9claf867039,
sfn://tbed0101.cern.ch/flatfile/SE00/dteam/my_test1
```

### **Example 7.6.1.4 (Adding a mapping)**

Adding a SURL-GUID mapping:



```
$ 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 7.6.1.5 (Removing a mapping)***

Removing the previously added mapping:

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

#### ***Example 7.6.1.6 (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 7.6.1.7 (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 7.6.1.8** (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 7.6.1.9** (Retrieving URLs 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 URLs 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, srm://castorgrid.cern.ch/castor/cern.ch/grid/dteam/output.txt  
> guid:3b43991d-637c-11d8-b4a4-adc4d7141a27, sfn://tbed0101.cern.ch/flatfile/SE00/dteam/output.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  
[...]
```

## **7.6.2. 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:



---

|                                     |   |
|-------------------------------------|---|
| <code>addAlias guid alias</code>    | Adds a new alias to the catalog.                      |
| <code>aliasExists alias</code>      | Checks whether the given alias exists in the catalog. |
| <code>guidExists guid</code>        | Checks whether the given GUID exists in the catalog.  |
| <code>guidForAlias alias</code>     | Returns the GUID for the given alias.                 |
| <code>aliasesForGuid guid</code>    | Returns the aliases for the given GUID.               |
| <code>removeAlias guid alias</code> | Removes an alias from the given GUID.                 |

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

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

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 can refer to the manpages or to [R30].

### 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-catalog/services/edg-replica-metadata-catalog
```

#### *Example 7.6.2.1 (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-rcm guidExists $GUID2 --endpoint $RMC_ENDPOINT  
> GUID does not exist : 'guid:c06a92ee-6911-11d8-a453-000000000000'
```

### ***Example 7.6.2.2 (Retrieving LFNs and GUIDs)***

Retrieving the GUID for a known alias.

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

Retrieving the existing aliases for a GUID.

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

### ***Example 7.6.2.3 (Adding new LFNs)***

In order to add a new alias, the `guid:` and `lfn:` prefixes must be used:

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

### ***Example 7.6.2.4 (Retrieving with wildcards)***

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

### ***Example 7.6.2.5 (Deleting an LFN)***

The previously added mapping is removed:

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



## 7.7. FILE AND REPLICA MANAGEMENT CLIENT TOOLS

LCG offers a variety of Data Management Client tools to upload/download files to/from the Grid, replicate data and locate the best replica available, and interact with the file catalogs.

Every user should deal with data management through the LCG Data Management tools (usually referred to as *lcg-utils* or `lcg-*` commands). They provide a high level interface (both command line and APIs) to the basic DM functionality, hiding the complexities of catalog and SEs interaction. Furthermore, such high level tools ensure the consistency between Storage Elements and catalog in DM operations and try to minimize the risk of grid files corruption.

The same functionalities are exploited by the `edg-replica-manager` wrapper. More details on this are given below.

Some lower level tools (like `edg-gridftp-*` commands and `globus-url-copy` ) are also available. These low level tools are quite helpful in some particular cases (see examples for more details). Their usage however is strongly discouraged for non expert users, since such tools do not try to ensure consistency between physical files in the SE and entries in the file catalog and their usage could be very dangerous.

Command line tools to be used for catalog interaction (both for RLS and LFC) have been already described in detail in the previous section.

### 7.7.1. LCG Data Management Client Tools

The LCG Data Management tools (usually called *lcg-utils*) allow users to copy files between UI, CE, WN and a SE, to register entries in the File Catalog and replicate files between SEs.

**NOTE:** Up to the LCG release 2.3.0, the `edg-replica-manager` command (also in abbreviated `edg-rm` form) provided the same functionality than the current `lcg_utils` offer. For performance reasons, the `edg-rm` was dismissed in favor of the `lcg_utils`. The current `edg-replica-manager` command is just a wrapper script around `lcg_utils`. In this way, it ensures the performance and functionalities of `lcg_utils`, mantaining the interface of the old Java CLI. More information about the `edg-replica-manager` wrapper script can be found in [R32]

The name and functionality overview of the available commands is shown in the following table.

#### Replica Management



---

|                      |   |
|----------------------|---|
| <code>lcg-cp</code>  | Copies a Grid file to a local destination (download).   |
| <code>lcg-cr</code>  | Copies a file to a SE and registers the file in the catalog (LFC or LRC) (upload).                |
| <code>lcg-del</code> | Deletes one file (either one replica or all replicas).  |
| <code>lcg-rep</code> | Copies a file from one SE to another SE and registers it in the catalog (LFC or LRC) (replicate). |
| <code>lcg-gt</code>  | Gets the TURL for a given SURL and transfer protocol.   |
| <code>lcg-sd</code>  | Sets file status to "Done" for a given SURL in an SRM's request.                                  |

### File Catalog Interaction

|                     |  |
|---------------------|--|
| <code>lcg-aa</code> | Adds an alias in the catalog (LFC or RMC) for a given GUID.              |
| <code>lcg-ra</code> | Removes an alias in the catalog (LFC or RMC) for a given GUID.           |
| <code>lcg-rf</code> | Registers in the the catalog (LFC or LRC/RMC), a file residing on an SE. |
| <code>lcg-uf</code> | Unregisters in the the catalog (LFC or LRC) a file residing on an SE.    |
| <code>lcg-la</code> | Lists the aliases for a given LFN, GUID or SURL.                         |
| <code>lcg-lg</code> | Gets the GUID for a given LFN or SURL.                                   |
| <code>lcg-lr</code> | Lists the replicas for a given LFN, GUID or SURL.                        |

Each command has a different syntax (arguments and options), but the `--vo <vo_name>` option to specify the virtual organisation of the user is present and mandatory in all the commands, except for `lcg-gt`, unless the environment variable `LCG_GFAL_VO` has been set, in which case the VO for the user is taken from the value of that variable<sup>9</sup>.

The `--config <file>` option (allows to specify a configuration file) and the `-i` option (allows to connect insecurely to the File Catalog) are currently ignored.

**Timeouts:** The commands `lcg-cr`, `lcg-del`, `lcg-gt`, `lcg-rf`, `lcg-sd` and `lcg-rep` all have timeouts implemented. By using the option `-t`, the user can specify a number of seconds for the tool to time out. The default is 0 seconds, i.e.: no timeout. If a tool times out in the middle of operations, all actions performed till that moment are undone, so no broken files are left on a SE and not unexisting files are registered in the catalogs.

### Environment:

- For all `lcg-*` commands to work, the environment variable `LCG_GFAL_INFOSYS` must be set to point to the IS provider (the BDII) in the format `hostname.domain:port`, so that the commands can retrieve the necessary information for their operation. Remember that the BDII read port is 2170.
- The endpoint(s) for the catalogs can also be specified (taking precedence over that published in the IS) through environmental variables: `LRC_ENDPOINT`, `RMC_ENDPOINT` for the RLS and `LFC_HOST` for the LFC. If no endpoints are specified, the ones published in the Information System are taken<sup>10</sup>.

---

<sup>9</sup>This variable can also be used by GFAL when resolving LFNs and GUIDs, as described in Appendix F

<sup>10</sup>As discussed in 7.5, these is not true for the LFC interaction commands (`lfc-*`), which require that the `LFC_HOST` variable is defined explicitly.



- If the variable `LCG_GFAL_VO` is set in the environment, then the `--vo` option is not required for the `lcg-*` commands, since they take the value of this variable.

In respect of authentication and authorization, `lcg-utils` manage data transfer securely through `gsiftp`. For this reason, the user must have a valid proxy and must appear in the `grid-mapfile` of the SE in order to use `lcg-cr`, `lcg-cp`, `lcg-rep` and `lcg-del`.

On the other side, **the information in the LRC and RMC catalogs is not protected** (RLS allows insecure access) and no proxy certificate is required for the rest of `lcg-*` commands, which do not deal with physical replicas. Although this situation is different for the LFC, currently the information stored in the RLS file catalogs **can be altered by anyone**.

**NOTE:** The user will often need to gather information on the existing Grid resources in order to perform DM operations. For instance, in order to specify the destination SE for the upload of a file, the information about the available SEs must be retrieved in advance. There are several ways to retrieve information about the resources on the Grid. The Information System may be queried directly through the `ldapsearch` command or via the `lcg-info` wrapper, the `lcg-infosites` wrapper may be used, or a monitoring tool (e.g. a web page displaying information on Grid resources) can be checked. All these methods are described in Chapter 5.

In what follows, some usage examples are given. Most command can run in verbose mode (`-v` or `--verbose` option). For details on the options of each command, please use the manpages of the commands.

#### *Example 7.7.1.1 (Uploading a file 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 and register it in the catalog, the `lcg-cr` command (which stands for copy and register) can be used:

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch file:/home/antonio/file1  
> guid:6ac491ea-684c-11d8-8f12-9c97cebf582a
```

where the only argument is the local file to be uploaded (a fully qualified URI) and the `-d <destination>` option indicates the SE used as the destination for the file. The command returns the unique GUID.

If no destination is given, the SE specified by the `VO_<VO-name>_DEFAULT_SE` environmental variable is taken. Such variable should be set in all WNs and UIs.

The `-P` option allows the user to specify a relative path name for the file in the SE. The absolute path is built appending the relative path to a root directory which is VO and SE specific and is determined through the Information System. If no `-P` option is given, the relative path is automatically generated following a certain schema.



There is also the possibility to specify the destination as a complete SURL, including SE hostname, the path, and a chosen filename. The action will only be allowed if the specified path falls under the user's VO directory.

The following are examples of the different ways to specify a destination:

```
-d lxb0710.cern.ch
-d sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/my_file
-d lxb0710.cern.ch -P my_dir/my_file
```

The option `-l <lfn>` can be used to specify a LFN:

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

**REMINDER:** If the RLS catalog is used, the LFN takes the form `lfn:<someLFN>` where `<someLFN>` can be any string. In the case that the LFC is used, the LFNs are organized to a hierarchical namespace (like UNIX directory trees). So the LFN will take the form `lfn:/grid/<voname>/<dir1>/...`. Remember that subdirectories in the namespace are **not** created automatically by `lcg-cr` and you should manage them yourself through the `lfc-mkdir` and `lfc-rmdir` command line tools described in the previous section.

The `-g` option allows to specify a GUID (otherwise automatically created):

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch \
-g guid:baddb707-0cb5-4d9a-8141-a046659d243b file:`pwd`/file2
> guid:baddb707-0cb5-4d9a-8141-a046659d243b
```

**Attention!** This option should not be used except for expert users and in very particular cases. Because the specification of an existing GUID is also allowed, a misuse of the tool may end up in a corrupted GRID file in which replicas of the same file are in fact different from each other.

Finally, in this and other commands, the `-n <#streams>` options can be used to specify the number of parallel streams to be used in the transfer (default is one).

**Known problem:** When multiple streams are requested, the GridFTP protocol establishes that the GridFTP server must open a new connection back to the client (the original connection, and only one in the case of one stream, is opened from the client to the server). This may become a problem when a file is requested from a WN and this WN is firewalled to disable inbound connections (which is usually the case). The connection will in this case fail and the error message returned (in the logging information of the job performing the data access) will be "425 can't open data connection".



### **Example 7.7.1.2** (Replicating a file)

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

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

> Source URL: sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1a6bef
File size: 30
Destination specified: lxb0707.cern.ch
Source URL for copy: gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1a6bef
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/2004-07-09/file50c0752c-f61f-4bc3-b48e-af3f22924b57
```

where the file to be replicated can be specified using a LFN, GUID or even a particular SURL, and the `-d` option is used to specify the SE where the new replica will be stored. This destination can be either an SE hostname or a complete SURL, and it is expressed in the same format as with `lcg-cr`. The command also admits the `-P` option to add a relative path to the destination (as with `lcg-cr`).

For one GUID, there can be only one replica per SE. If the user tries to use the `lcg-rep` command with a destination SE that already holds a replica, the command will exit successfully, but no new replica will be created.

### **Example 7.7.1.3** (Listing replicas, GUIDs and aliases)

The `lcg-lr` command allows users to list all the replicas of a file that have been successfully registered in the File Catalog:

```
$ lcg-rep --vo dteam lfn:my_alias1
> sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-09/file79aee616-6cd7-4b75-8848-f09110ade178
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1a6bef
```

Again, LFN, GUID or SURL can be used to specify the file for which all replicas must be listed. The SURLs of the replicas are returned.

The `lcg-lg` command (list GUID) returns the GUID associated with a specified LFN or SURL:





```
$ lcg-lg --vo dteam sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-09/file79aee616-6cd7-4b75-8848-f09110ade178
> guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
```

The `lcg-la` command (list aliases) can be used to list the LFNs associated with a particular file, which can be, identified by its GUID, any of its LFNs, or the SURL of one of its replicas.

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

The `lfc-*` commands for LFC and the tools `edg-lrc` and `edg-rmc` for the RLS offer more functionalities for catalog interaction, although the ones provided by the `lcg-*` commands should be enough for a normal user.

#### ***Example 7.7.1.4 (Copying files out of the Grid)***

The `lcg-cp` command can be used to copy a Grid file to a non-grid storage resource. The first argument (source file) can be a LFN, GUID or one SURL of a valid Grid file, the second argument (destination file) must be a local filename or valid TURL. In the following example, the verbose mode is used and a timeout of 100 seconds is specified:

```
$ lcg-cp --vo dteam -t 100 -v lfn:/grid/dteam/hosts file:/tmp/f2
Source URL: lfn:/grid/dteam/hosts
File size: 104857600
Source URL for copy:
gsiftp://lxb2036.cern.ch/storage/dteam/generated/2005-07-17/fileea15c9c9-abcd-4e9b-8724-1ad60c5afe5b
Destination URL: file:///tmp/f2
# streams: 1
# set timeout to 100 (seconds)
      85983232 bytes   8396.77 KB/sec avg   9216.11
Transfer took 12040 ms
```

Notice that although this command is designed to copy files from a SE to non-grid resources, if the proper TURL is used (using the `gsiftp:` protocol), a file could be transferred from one SE to another, or from out of the Grid to a SE. **This should not be done**, since it has the same effect as using `lcg-rep` BUT **skipping the file registration**, making in this way this replica invisible to Grid users.

#### ***Example 7.7.1.5 (Obtaining a TURL for a replica (actually more than that ... )***

The `lcg-gt` allows to get a TURL from a SURL and a supported protocol. The command behaves very differently if the Storage Element exposes an SRM interface or not. The command always returns



three lines of output: the first is always the TURL of the file, the last two are meaningful only in case of SRM interface.

- In case of classic SE or a MSS without SRM interface, the command obtains the TURL by simple string manipulation of the SURL (obtained from the File Catalog) and the protocol (checking in the Information System if it is supported by the Storage Element). No direct interaction with the SE is involved. The last two lines of output are always zeroes.

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

Be aware that in case of MSS, the file could be not staged on disk but only stored on tape. For this reason, an operation like

```
$ lcg-cp gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-08/file0dca46-2214-4db8-9ee8-2930de1a6bef file://tpm/somefile.txt
```

could hang forever, waiting for the file to be staged (a timeout mechanism is not implemented in `lcg-utils`).

- In the case of SRM interface, the TURL is returned to the `lcg-gt` command by the SRM itself. In case of MSS, the file will be staged on disk (if not present already) before a valid TURL is returned. It could take `lcg-gt` quite a long time to return the TURL (depending on the conditions of the stager) but a successive `lcg-cp` of such TURL will not hang. This is one of the reasons for which a SRM interface is desirable for all MSS.

The second and third lines of output represent the requestID and fileID for the `srm_put` request (hidden to the user) which will remain open unless explicitly closed (at least in SRM v1 currently deployed). It is important to know that some SRM Storage Elements are limited in the maximum number of open requests. Further requests will fail, once this limit has been reached. It is therefore good practice to close the request once the TURL is not needed anymore. This can be done with the `lcg-sd` command which needs as arguments the TURL of the file, the requestID and fileID.

```
$ lcg-gt srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/generated/2005-04-12/filefadle7fb-9d83-4050-af51-4c9af7bb095c gsiftp
> gsiftp://castorgrid.cern.ch:2811//shift/lxfsrk4705/data02/cg/stage/filefadle7fb-9d83-4050-af51-4c9af7bb095c.43309
> -337722383
> 0
```

[ ... do something with the TURL ... ]



```
$ lcg-sd gsiftp://castorgrid.cern.ch:2811//shift/lxfsrk4705/data02/cg/stage/filefad1
e7fb-9d83-4050-af51-4c9af7bb095c.43309 -337722383 0
```

### ***Example 7.7.1.6 (Deleting replicas)***

A file that is stored on a Storage Element and registered with a catalog can be deleted using the `lcg-del` command. If a SURL is provided as argument, then that particular replica will be deleted. If a LFN or GUID is given instead, then the `-s <SE>` option must be used to indicate which one of the replicas must be erased, unless the `-a` option is used, in which case all replicas of the file will be deleted and unregistered (on a best-effort basis). If all the replicas of a file are removed, the corresponding GUID-LFN mappings are removed as well.

```
$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> sfn://lxb0707.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file78ef5a13-166f-4701-
8059-e70e397dd2ca
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file21658bfb-6eac-409b-
9177-88c07bb1a57c

$ lcg-del --vo=dteam -s lxb0707.cern.ch guid:91b89dfe-ff95-4614-bad2-c538bfa28fac

$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file21658bfb-6eac-409b-
9177-88c07bb1a57c

$ lcg-del --vo dteam -a guid:91b89dfe-ff95-4614-bad2-c538bfa28fac

$ lcg-lr --vo dteam guid:91b89dfe-ff95-4614-bad2-c538bfa28fac
> lcg_lr: No such file or directory
```

The last error indicates that the GUID is no longer registered within the catalogs of LCG-2, as the last replica was deleted.

### ***Example 7.7.1.7 (Registering and unregistering Grid files)***

The `lcg-rf` (register file) command allows to register a file physically present in a SE, creating a GUID-SURL mapping in the catalog. The `-g <GUID>` allows to specify a GUID (otherwise automatically created).

```
$ lcg-rf -v --vo dteam -g guid:baddb707-0cb5-4d9a-8141-a046659d243b sfn://lxb0710.cern.ch/
```



```
flatfiles/SE00/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1a6bef  
> guid:baddb707-0cb5-4d9a-8141-a046659d243b
```

Likewise, `lcg-uf` (unregister file) allows to delete a GUID-SURL mapping (respectively the first and second argument of the command) from the catalog.

```
$ lcg-uf --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-12/file04eec6b2-9ce5-4fae-bf62-b6234bf334d6
```

If the last replica of a file is unregistered, the corresponding GUID-LFN mapping is also removed.

**WARNING:** `lcg-uf` just removes entries from the catalog, it does not remove any physical replica from the SE. Watch out for consistency.

### ***Example 7.7.1.8 (Managing aliases)***

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

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

Correspondingly, the `lcg-ra` command (remove alias) allows a user to remove an LFN from an existing GUID:

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

In order to list the aliases of a file, the `lcg-la` command, discussed previously, has been used.

## **7.7.2. 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:



---

|   |  |
|---|--|
| <code>edg-gridftp-exists URL</code>               | Checks the existence of a file or directory on a SE. |
| <code>edg-gridftp-ls URL</code>                   | Lists a directory on a SE.                           |
| <code>edg-gridftp-mkdir URL</code>                | Creates a directory on a SE.                         |
| <code>edg-gridftp-rename sourceURL destURL</code> | Renames a file on a SE.                              |
| <code>edg-gridftp-rm URL</code>                   | Removes a file from a SE.                            |
| <code>edg-gridftp-rmdir URL</code>                | Removes a directory on a SE.                         |
| <code>globus-url-copy sourceURL destURL</code>    | Copies files between SEs.                            |

The commands `edg-gridftp-rename`, `edg-gridftp-rm`, and `edg-gridftp-rmdir` should be used with extreme care and only in case of serious problems. In fact, these commands do not interact with any of the catalogs and therefore they can compromise the consistency/coherence of the information contained in the Grid. Also `globus-url-copy` is dangerous since it allows the copy of a file into the Grid without enforcing its registration.

All the `edg-gridftp-*` commands accept `gsiftp` as the only valid protocol for the TURL.

Some usage examples are shown. They are by no means exhaustive. To obtain help on these commands use the option `--usage` or `--help`. General information on gridFTP is available in [R25].

#### ***Example 7.7.2.1 (Listing and checking the existence of Grid files)***

The `edg-gridftp-exists` and `edg-gridftp-ls` commands can be useful in order to check if a file is physically in a SE, regardless of its presence in the Grid catalogs.

```
$ lcg-lr --vo dteam guid:27523374-6f60-44af-b311-baa3d29f841a
> sfn://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/file42ff7086-8063-414d-9000-75c459b71296

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

$ edg-gridftp-exists gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/my_fake_file
> error gsiftp://lxb0710.cern.ch/flatfiles/SE00/dteam/generated/2004-07-13/my_fake_file does not exist

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

#### ***Example 7.7.2.2 (Copying a file with globus-url-copy)***



The `globus-url-copy` command can be used to copy files between any two Grid resources, and from/to a non-grid resource. Its functionality is similar to that of `lcg-cp`, but source and destination must be specified as TURLs.

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

## 7.8. JOB SERVICES AND DATA MANAGEMENT

Some specific JDL attributes allow the user to specify requirements about input and output data:

### *Example 7.8.1 (Specifying input data in a job)*

If a job requires one or more input file stored in an LCG Storage Element, the `InputData` JDL attribute list can be used. Files can be specified by both by LFN and GUID.

An example of JDL specifying input data looks like:

```
InputData = {"lfn:doe/prod/kin\_1",
             "guid:136b48a64-4a3d-87ud-3bk5-8gnn46m49f3"};
DataAccessProtocol = {"rfio", "gsiftp", "gsidcap"};
```

The `DataAccessProtocol` attribute is used to specify the protocols that the application can use to access the file and is mandatory if `InputData` is present. Only data in SEs which support one or more of the listed protocols are considered. The Resource Broker will schedule the job to a *close* CE (in the sense of the Glue Schema, see Appendix G) to the SE holding the largest number of input files requested. In case several CEs are suitable, they will be ranked according to the ranking expression.

To be sure, the `InputData` attribute is used only during the match making process (to match CEs and SEs). It has nothing to do with the real access to files that the job can do while running. However, it is obviously reasonable that files listed in the attribute are really accessed by the job and viceversa.

**Warning:** Sometimes using the `InputData` attribute might result to be a bad idea: if many jobs need to access a single file and a unique replica of such file exists in the Grid, all jobs will land at the site containing the replica. Very soon all the CPUs of the site will be filled and jobs will be put in waiting state. It would be more efficient to schedule the job in a site with free CPUs and replicate the file in the *close* SE of the site.

### *Example 7.8.2 (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 abort the job if there is no CE close to the `OutputSE` specified by the user.

### ***Example 7.8.3 (Automatic upload and registration of output files)***

The `OutputData` attribute list allows the user to automatically upload and register files produced by the job on the WN. For each file, three attributes can be set:

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

The following code shows an example `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"  
  ]  
};
```

### ***Example 7.8.4 (Selecting the file catalog to use for match making)***

In order for the RB to select CEs that are close to the files required by a job (by the `InputData` attribute), it has to locate the SEs where these files are stored. In order to do this, the RB uses the `Data`



Location Interface service, which in turn contacts some file catalog. Since it is possible that several different file catalogs exist on the Grid, the user has the possibility to select which one he wants to talk to by using the JDL attribute `ReplicaCatalog`.

The user will specify the attribute and the endpoint of the catalog as value, as in this example:

```
ReplicaCatalog = "http://lfc-lhcb-test.cern.ch:8085/";
```

If no value is specified, then the first catalog supporting the DLI interface that is found in the information system will be used (which should probably be the right choice for the normal user).

**NOTE:** Only catalogs that present a DLI interface can be talked to by the DLI service. Currently, the RLS catalog does not provide such interface, and so in order to use this catalog, the RB is configured to interface directly with it (without using the DLI). Thus, summarising, the RB can be configured to either talk to the RLS catalog or to use the DLI. In the second possibility, the user has the option to manually specify the endpoint of the catalog he wants to use.

## 7.9. ACCESSING GRID FILES FROM A JOB

In order to access grid files from a job a user might think about two scenarios:

1. The user downloads the file into the WN and accesses the file through POSIX calls
2. The user access the data in the Storage Element directly through some file access protocol supported by the SE

Both scenarios present advantages and disadvantages and one might think also of a mixed situation, where *local* files (in the CloseSE) are accessed directly while *remote* files are downloaded into the WN and accessed through posix calls. Both scenarios are discussed in more details in the LCG-2 User Scenario (see Applicable Documents in Sec. 1.5). Appendix F gives some examples about how to perform data management operations and file access from a C (or C++) application, using respectively the `lcg_util` and the GFAL APIs.

## 7.10. POOL AND LCG-2

The *Pool Of persistent Objects for LHC (POOL)* tool is used by most of the LHC experiments as a common persistency framework for the LCG application area. It is through POOL that they store their data.





Objects created by users using POOL are stored into its own File Catalog (XML Catalog). In order to be able to operate in the Grid, it is certainly very important for these experiments to have an interaction between the POOL catalog and the LCG-2 file catalogs.

Currently, there is a satisfactory interoperability between the POOL catalogs and the RLS catalogs. That is, there is a way to migrate POOL catalog entries to the RLS catalogs (i.e.: register those files within LCG-2) and also the LCG-2 Data Management tools can access those files as with any other Grid file. A way to achieve the same kind of interoperability between POOL and the LFC has not been implemented yet.

### ***Example 7.10.1 (Migration from POOL(XML) to LCG(RLS))***

We assume that the user has used POOL and as result has created a file which has been registered into the XML catalog of POOL. Now the point is how to register this file into the LCG catalog, the RLS.

In the first place, it is necessary to obtain the connection to the RLS catalog. A contact string has to be specified through the environment variable `POOL_CATALOG` as follows:

```
$ export POOL_CATALOG=edgcatalog_http://<host>:<port>/<path> (bash shell)
$ setenv POOL_CATALOG edgcatalog_http://<host>:<port>/<path> (csh shell)
```

For example into LCG-2 this environment variable may be set to:

```
$ export POOL_CATALOG=edgcatalog_http://rlscert01.cern.ch:7777/$VO/v2.2
/edg-local-replica-catalog/services/edg-local-replica-catalog
```

In the case that the user has specified the file as a SURL into POOL, he can assign it an LFN with POOL as follows:

```
$ FCregisterLFN -p <SURL> -l <LFN>
```

Now the user can make some test to check whether the file is into the LRC with the RLS client:

```
$ edg-lrc mappingsByPfn <SURL> --endpoint <LRC>
```

Or into the RMC:

```
$ edg-rmc mappingsByAlias <LFN> --endpoint <RMC>
```

Finally he can check if the Data Management tools are able to find the file:



---

```
$ lcg-lr --vo <VO> lfn:<LFN>
```

Note that in case that the POOL user has defined the SURL entry following a ROOT format, he must use the command `FCrenamePFN` to create a SURL entry compatible with the RLS catalog.

A complete list of POOL commands can be found into [R31]. And in a machine where the interface is installed, the user can see them just by typing `FC<tab>` (or `FC<Ctrl-D>`, if that does not work).



---

## APPENDIX A THE GRID MIDDLEWARE

The operating systems supported by LCG-2 are Red Hat 7.3 and Scientific Linux 3, while the supported architectures are IA32 and IA64.

The LCG-2 middleware layer uses components from several Grid projects, including DataTag (EDT), DataGrid (EDG), EGEE, INFN-GRID, Globus and Condor.

In some cases, LCG patches are applied to the components, so the final software used is not exactly the same as the one distributed by the original project.

The components which are currently used in LCG-2 are listed in table 1.



| Component  | LCG                        | EGEE | EDG         | EDT | INFN-GRID | Globus | Condor | Other |
|--|----------------------------|------|-------------|-----|-----------|--------|--------|-------|
| <b>Basic middleware</b>  |                            |      |             |     |           |        |        |       |
| Globus 2.4.3<br>ClassAds 0.9.4   |                            |      |             |     |           | ✓      | ✓      |       |
| <b>Security</b>  |                            |      |             |     |           |        |        |       |
| MyProxy  |                            |      |             |     |           |        |        | ✓     |
| <b>VO management</b>   |                            |      |             |     |           |        |        |       |
| LDAP-based<br>VOMS   | ✓                          | ✓    | ✓           |     |           |        |        |       |
| <b>Workload management</b>   |                            |      |             |     |           |        |        |       |
| Condor/Condor-G 6.6.5<br>EDG WMS   | ✓                          |      | ✓           |     |           |        | ✓      |       |
| <b>Data management</b>   |                            |      |             |     |           |        |        |       |
| Replica Manager<br>Replica Location Service<br>LCG File Catalog<br>Disk Pool Manager<br>GFAL<br>LCG DM tools | ✓<br>✓<br>✓<br>✓<br>✓<br>✓ |      | ✓<br>✓      |     |           | ✓      | ✓      |       |
| <b>Fabric management</b>   |                            |      |             |     |           |        |        |       |
| LCFG<br>Quattor<br>YAIM<br>LCAS/LCMAPS   | ✓<br>✓<br>✓                |      | ✓<br>✓<br>✓ |     |           |        |        | ✓     |
| <b>Monitoring</b>  |                            |      |             |     |           |        |        |       |
| GridICE  |                            |      |             |     | ✓         |        |        |       |
| <b>Information system</b>  |                            |      |             |     |           |        |        |       |
| MDS<br>Glue Schema<br>BDII<br>R-GMA<br>LCG Information tools   | ✓<br>✓<br>✓                | ✓    | ✓           | ✓   |           | ✓      |        | ✓     |

Table 1: Software components of LCG-2 and projects that contributed to them.



## APPENDIX B CONFIGURATION FILES AND VARIABLES

Some of the configuration files and environmental variables that may be of interest for the Grid user are listed in the following tables. Unless explicitly stated, they are all located/defined in the User Interface.

### Environmental variables:

| Variable                  | Notes  |
|---------------------------|--|
| \$EDG_LOCATION            | Base of the installed EDG software.  |
| \$EDG_TMP                 | Temporary directory.   |
| \$EDG_WL_JOBID            | Job id (defined for a running job). <b>In a WN.</b>  |
| \$EDG_WL_LIBRARY_PATH     | Library path for EDG's WMS commands.   |
| \$EDG_WL_LOCATION         | Base of the EDG's WMS software.  |
| \$EDG_WL_PATH             | Path for EDG's WMS commands.   |
| \$EDG_WL_RB_BROKERINFO    | Location of the .BrokerInfo file. <b>In a WN.</b>  |
| \$EDG_WL_UI_CONFIG_VAR    | May be used to specify a configuration different that \$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf.      |
| \$EDG_WL_UI_CONFIG_VO     | May be used to specify a configuration different that \$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf.         |
| \$LCG_CATALOG_TYPE        | Type of file catalog used (edg or lfc) for lcg-utils and GFAL.   |
| \$LCG_GFAL_INFOSYS        | Location of the BDII for lcg-utils and GFAL.   |
| \$LCG_GFAL_VO             | May be used to tell lcg-utils or GFAL about a user's VO. <b>To set in a UI or ship with a job's JDL.</b> |
| \$LFC_HOST                | Location of the LFC catalog (only for catalog type lfc).   |
| \$LCG_LOCATION            | Base of the installed LCG software.  |
| \$LCG_RFIO_TYPE           | Type of RFIO (secure or insecure) to use by GFAL.  |
| \$LCG_TMP                 | Temporary directory.   |
| \$LRC_ENDPOINT            | May be used to define th LRC endpoint for lcg-utils and GFAL (overriding the IS).                        |
| \$RMC_ENDPOINT            | May be used to define the RMC endpoint for lcg-utils and GFAL (overriding the IS).                       |
| \$VO_<VO_name>_DEFAULT_SE | Default SE defined for a CE. <b>In a WN.</b>   |
| \$VO_<VO_name>_SW_DIR     | Base directory of the VO's software or "." for WNs with no shared file system among WNs. <b>In a WN.</b> |
| \$X509_USER_PROXY         | Location of the user proxy certificate (default is /tmp/x509up_u<uid>).                                  |

---

**Configuration files:**

| <b>Configuration File</b>                                    | <b>Notes</b>  |
|--|---|
| <code>\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf</code>    | WMS command line tools settings: Retry count, error and output directory, default VO...                           |
| <code>\$EDG_WL_LOCATION/etc/&lt;VO&gt;/edg_wl_ui.conf</code> | VO's specific WMS settings: Resource Broker, Logging and Bookkeeping server and Proxy Server to use.              |
| <code>\$EDG_LOCATION/var/edg-rgma/rgma-defaults</code>       | RGMA default values.  |
| <code>\$EDG_LOCATION/etc/edg_wl.conf</code>                  | <b>In the RB!*</b> Resource Broker's configuration file. It may be useful to find out which BDII the RB is using. |

\* This file can not be edited by the user (since it is located in the Resource Broker) but may be retrieved via GridFTP to look at its contains.

## APPENDIX C JOB STATUS DEFINITION

As it was already mentioned in Chapter 6, a job can find itself in one of several possible states. Also, only some transitions between states are allowed. These transitions are depicted in Figure 14.

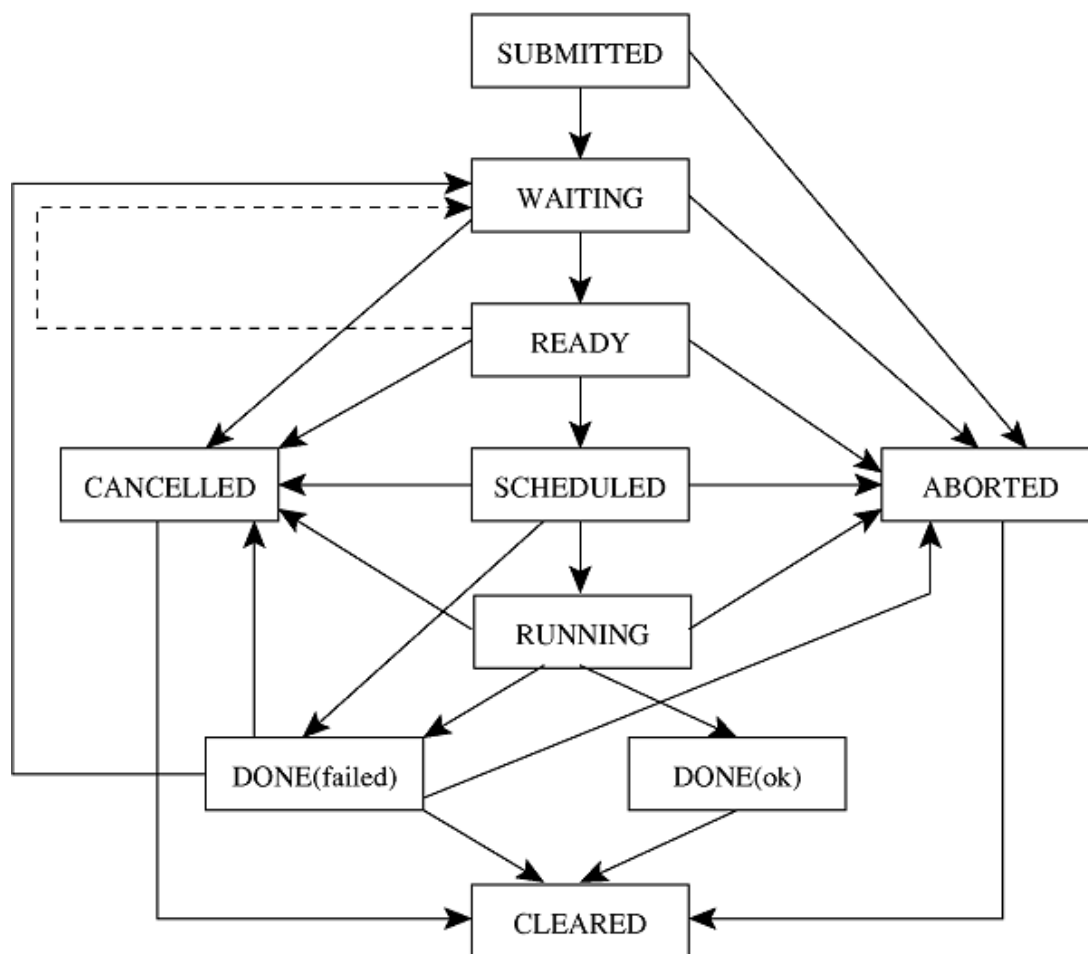


Figure 14: Possible job states in the LCG-2

And the definition of the different states is given in this table.

| <b>Status</b> | <b>Definition</b>   |
|---------------|---|
| SUBMITTED     | The job has been submitted by the user but not yet processed by the Network Server                          |
| WAITING       | The job has been accepted by the Network Server but not yet processed by the Workload Manager               |
| READY         | The job has been assigned to a Computing Element but not yet transferred to it                              |
| SCHEDULED     | The job is waiting in the Computing Element's queue   |
| RUNNING       | The job is running  |
| DONE          | The job has finished  |
| ABORTED       | The job has been aborted by the WMS (e.g. because it was too long, or the proxy certificated expired, etc.) |
| CANCELED      | The job has been canceled by the user   |
| CLEARED       | The Output Sandbox has been transferred to the User Interface   |





---

## APPENDIX D USER TOOLS

### D.1. INTRODUCTION

This section introduces some tools that are not really part of the middleware stack of LCG-2, but that can be very useful for user activities on the Grid. Certainly, there are potentially tens of tools and it is impossible to cover them all in this guide. Basically, the purpose of this guide is to introduce the functionality of some of them and to point to other sources of documentation where more detailed information about the tools can be found.

This section will probably evolve to include information of new tools, as they appear or we gain knowledge of them.

Detailed information on the different tools here summarised is provided in Wiki, under the following URL:

[http://goc.grid.sinica.edu.tw/gocwiki/User\\_tools](http://goc.grid.sinica.edu.tw/gocwiki/User_tools)

### D.2. JOB MANAGEMENT FRAMEWORK

The submission of large bunches of jobs to LCG resources is a common practice of the VOs during their production phases (software implementation, data production, analysis, etc). The monitoring of these bunches of jobs can become a difficult task without adapted tools that are able to handle the submission and retrieval of the outputs.

Most of the VOs have developed their own tools to perform such job. Here, a framework to automatically submit large bunches of jobs and keep track of their outputs is proposed. Its aim is to assist and guide the users or VOs with the intention of developing their own tools. They could seize parts of this framework to include them in their own applications.

The framework consists mainly of two tools:

- ***submitter\_general***: It perform the automatic job submission
- ***get\_output***: It retrieves and handles the corresponding outputs

Information on this tool can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/Job\\_Management\\_Framework](http://goc.grid.sinica.edu.tw/gocwiki/Job_Management_Framework)



### D.3. JOB MONITORING (LCG-JOB-MONITOR)

The `lcg-job-monitor` command can be used to monitor the process of a job currently running in a WN. The command is intended to be run on a UI. This tool gives information about all statistics for a given `jobId`, e.g.: memory, virtual memory, real memory, CPU time, DN etc...

The information is retrieved by querying the JobMonitor table (published via R-GMA). The command can return information either for a single job (given the job id), for a user (specifying the DN) or for a whole VO (by name). Standard R-GMA type of queries are supported: LATEST, HISTORY, CONTINUOUS.

Usage is:

```
lcg-job-monitor [-j <jobid>] [-v <VO>] [-u <DN>] [-q <query_type>]
```

Information on this tool can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/Job\\_Monitoring](http://goc.grid.sinica.edu.tw/gocwiki/Job_Monitoring)

### D.4. JOB STATUS MONITORING (LCG-JOB-STATUS)

This tool provides information about the status of a running job. It is intended to be run on a UI. The information is retrieved by querying the JobStatusRaw table (published via R-GMA). The command returns information for a specified job (given the job id).

Usage:

```
lcg-job-status.py [-j <jobid>] [-q <type>]
```

where the query type can be either LATEST, CONTINUOUS or HISTORY (these are standard R-GMA query types). For LATEST and HISTORY queries, some seconds are waited after the query is made. After that, the returned results, or a "no events" message, if none is got, are printed. In the case of CONTINUOUS queries, the status is checked every 5 seconds until the program is exited via `Ctrl-C` or the Done status is reached.

Information on this tool can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/Job\\_Status\\_Monitoring](http://goc.grid.sinica.edu.tw/gocwiki/Job_Status_Monitoring)

### D.5. TIME LEFT UTILITY (LCG-GETJOBSTATS, LCG\_JOBSTATS.PY)

The tools described in this section can be invoked by a running job to determine some statistics about itself. Right now the parameters that can be retrieved are:



- How much CPU or wall clock time it has consumed
- How long it can still run (before reaching the CPU or wall clock time limits).

There are scripts (CLI) and python modules (API) that can be used for these purposes. They are described in detail later. They work by querying the CE's batch system (using different ways depending on the batch system that the given CE uses).

The results returned by the tools are not always trustable, since some sites do not set time limits and some LRMSs cannot provide resource usage information to the tool. The underlying idea of these CLI and API is that the experiment decides how to use them and when to trust their results. Please read the documentation regarding the different batch systems supported.

**ATTENTION!!:** This command executes heavy processes on the Computing Elements, please do not use it too often (not every minutes or event processed!). Rather, limit the usage to something like once every hour, or when a sensitive percentage of your job is accomplished.

The following files should be present in the WN (either already in the release or shipped with the job in a tarball):

- `lcg-getJobStats` (small wrapper bash script around the corresponding python script)
- `lcg-getJobTimes` (small wrapper bash script around the corresponding python script)
- `lcg-getJobStats.py` (python script: executable)
- `lcg-getJobTimes.py` (python script: executable)
- `lcg_jobConsumedTimes.py` (python module)
- `lcg_jobStats.py` (python module)

In principle, one should only deal with `lcg-getJobStats` or with `lcg_jobStats.py` (for python API). `lcg-getJobTimes` (and `lcg_jobConsumedTimes.py`) provide a way to estimate the used CPU and wall clock time without querying the batch system, but by parsing the proc filesystem. It is internally called by `lcg-getJobStats` in the cases where it cannot get the information from the batch system (like when Condor is used).

Information on this tool can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/Time\\_Left\\_Utility](http://goc.grid.sinica.edu.tw/gocwiki/Time_Left_Utility)

## D.6. INFORMATION SYSTEM READER (LCG-INFO)

This command was already discussed in Section 5.1.2.



---

Nevertheless, the more up-to-date information can be always got from:

[http://goc.grid.sinica.edu.tw/gocwiki/Information\\_System\\_Reader](http://goc.grid.sinica.edu.tw/gocwiki/Information_System_Reader)



---

## APPENDIX E VO-WIDE UTILITIES

### E.1. INTRODUCTION

This section introduces some utilities that are only relevant to certain people in a VO (VO managers, experiment software managers...). They are basically administrative tools. The purpose of this section is to introduce the functionality of some of them and to point to other sources of documentation where more detailed information about the utilities can be found.

Detailed information on the different tools here summarised is provided in Wiki, under the following URL:

<http://goc.grid.sinica.edu.tw/gocwiki/VoDocs>

### E.2. FREEDOM OF CHOICE FOR RESOURCES

The *Freedom of Choice for Resources (FCR) Pages* is a web interface for VO Software Managers. The tool gives a possibility to set up selection rules for Computing and Storage Elements, which will affect the information published by top level BDIIs configured accordingly.

Resources can be permanently in- or excluded or be used in case if the execution of the *Sites Functional Tests (SFT)* was successful. There is also a possibility to use VO-specific test results to be taken in account on the top of the SFT results.

Certificate-based authentication method protects the pages. Access has to be requested.

Information on this can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/FCR\\_Pages](http://goc.grid.sinica.edu.tw/gocwiki/FCR_Pages)

### E.3. THE VO BOX

The VO-BOX is a type of node, which will be deployed in all sites, where the experiment can run specific agents and services. The access to the VO-BOX is restricted to the SOFTWARE GROUP MANAGER of the VO. If you are not your VO SGM, you will not be interested in the VO-BOX.

A description of the VO-BOX functionalities and usage is described in WIKI indicated below.

Information on this can be found under:

<http://goc.grid.sinica.edu.tw/gocwiki/VOBOX.HowTo>



---

#### **E.4. EXPERIMENTS SOFTWARE INSTALLATION**

Authorized users can install software in the computing resources of LCG-2. The installed software, which we will call Experiment Software, is also published in the Information Service, so that user jobs can run on nodes where the software they need is installed.

The Experiment Software Manager (ESM) is the member of the experiment VO entitled to install Application Software in the different sites. The ESM can manage (install, validate, remove...) Experiment Software on a site at any time through a normal Grid job, without previous communication to the site administrators. Such job has in general no scheduling priorities and will be treated as any other job of the same VO in the same queue. There would be therefore a delay in the operation if the queue is busy.

The site provides a dedicated space where each supported VO can install or remove software. The amount of available space must be negotiated between the VO and the site.

Information on this can be found under:

[http://goc.grid.sinica.edu.tw/gocwiki/Experiments\\_Software\\_Installation](http://goc.grid.sinica.edu.tw/gocwiki/Experiments_Software_Installation)

## APPENDIX F DATA MANAGEMENT AND FILE ACCESS THROUGH AN APPLICATION PROGRAMMING INTERFACE

The development of code for jobs submitted to LCG-2 is out of the scope of this guide, and therefore the different APIs for Data Management and Grid File Access will not be covered in full detail. This section just summarizes what APIs exist and gives some examples of `lcg_util` and GFAL use. For general information on APIs for accessing files and other Grid resources, refer to [R5].

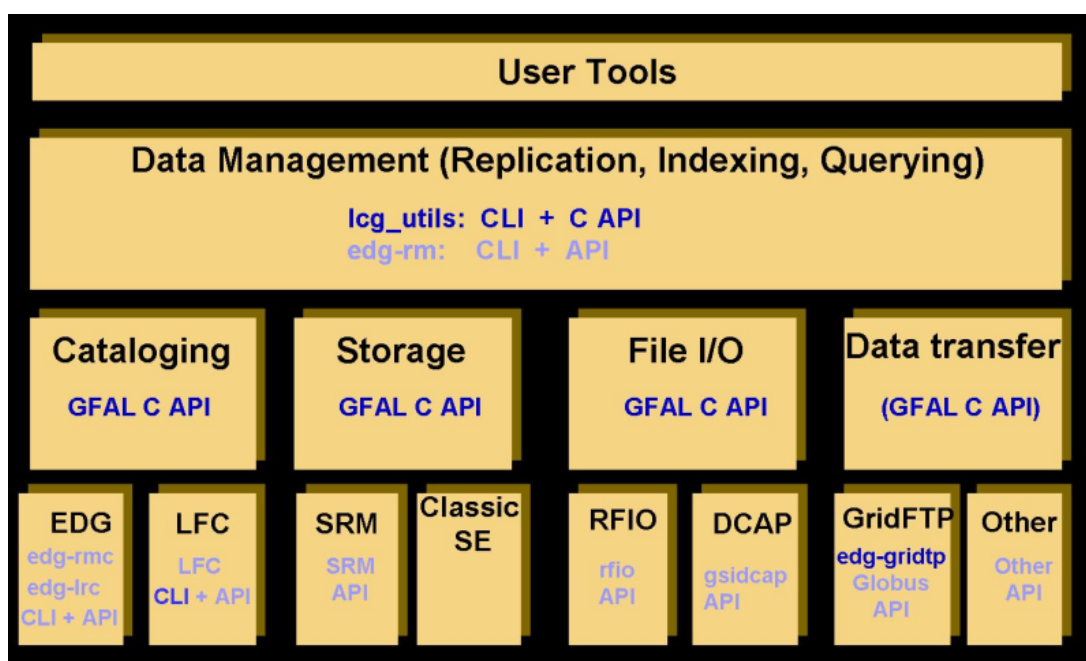


Figure 15: Layered view of the Data Management APIs and CLIs

Figure 15 shows a layered view of the different APIs that are available for Data Management operations in LCG-2. In the figure the CLIs and APIs whose use is discouraged are shadowed. It also includes the already described CLIs, which can be usually related to one of the APIs (as *being in the same layer*).

On the top, just below the tools developed by the users, we find the *lcg\_util* API. This is a C API that provides the same functionality as the `lcg-*` commands (`lcg-utils`). In fact, the commands are just a wrapper around the C calls. This layer should cover most basic needs of user applications. It is abstract in the sense that it is independent from the underlying technology, since it will transparently interact with either the RLS or the LFC catalog and will use the correct protocol (usually GSIFTP) for file transfer.

Apart from the basic calls `lcg_cp`, `lcg_cr`, etc., there are other calls that enhance this with a buffer for complete error messages (`lcg_cpx`, `lcg_crx`, ...), that include timeouts (`lcg_cpt`, `lcg_crt`, ...), and both (`lcg_cpxt`, `lcg_crxt`). Actually, all calls use the most complete version (i.e. `lcg_cpxt...`) with default

values for the arguments that were not provided.

Below the `lcg_util` API, we find *the Grid File Access Library (GFAL)*. GFAL provides calls for catalog interaction, storage management and file access and can be very handy when an application requires access to some part of a big Grid file but does not want to copy the whole file locally. The library hides the interactions with the LCG-2 catalogs and the SEs and SRMs and presents a POSIX interface for the I/O operations on the files. The function names are obtained by prepending `gfal_` to the POSIX names, for example `gfal_open`, `gfal_read`, `gfal_close`...

GFAL accepts GUIDs, LFNs, SURLS and TURLs as file names, and, in the first two cases, it tries to find the closest replica of the file. Depending on the type of storage where the file's replica resides in, GFAL will use one protocol or another to access it. GFAL can deal with GSIFTP, secure and insecure RFIO, or `gsidcap` in a transparent way for the user (unless a TURL is used, in which case the protocol is explicitly indicated).

**NOTE:** In the case where LFNs or GUIDs are used, the library needs to contact the file catalogs to get the corresponding TURL. Since the catalogs are VO dependant, and since the calls do not include any argument to specify the VO, GFAL requires the `LCG_GFAL_VO` environment variable to be set; along with the pointer for the Information Service: `LCG_GFAL_INFOSYS`. Alternatively, the endpoints of the catalogs may be specified directly, by setting: `LFC_HOST` (LFC) or `RMC_ENDPOINT` and `LRC_ENDPOINT` (RLS).

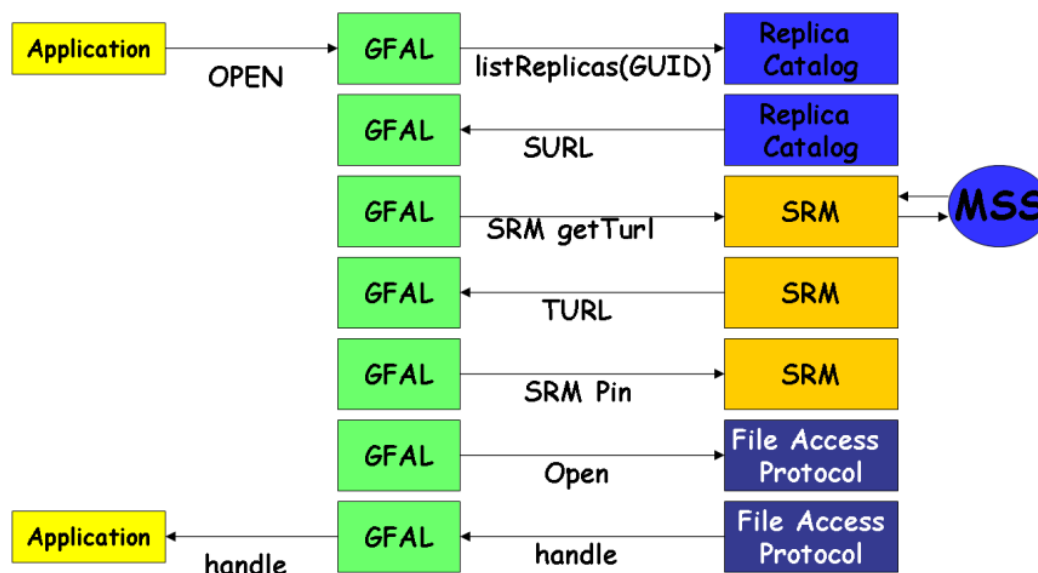


Figure 16: Flow diagram of a GFAL call

This behaviour is illustrated in Figure 16, which shows the flow diagram of a `gfal_open` call. This call will locate a Grid file and return a remote file descriptor so that the caller can read or write file remotely, as it would do for a local file. As shown in the figure, first, if a GUID is provided, GFAL will contact





a file catalog to retrieve the corresponding SURL. Then, it will access the SRM interface of the SE that the SURL indicates, it will get a valid TURL and also pin the file so that it is there for the subsequent access. Finally, with the TURL and using the appropriate protocol, GFAL will open the file and return a file handle to the caller.

Nevertheless, GFAL sometimes exposes functionality applicable only to a concrete underlying technology (or protocol) if this is considered useful. A good example of this is the exposed SRM interface that GFAL provides. Some code exploiting this functionality is shown later.

Finally, below GFAL, we find some other CLIs and APIs which are technology dependent. Their direct use is in general discouraged (except for the mentioned cases of the LFC client tools and the `edg-gridftp-*` commands). Nonetheless, some notes on the RFIO API are given later on.

### ***Example F.0.1 (Using `lcg_util` API to transfer a file)***

The following example copies a file from a SE to the WN where our job is running, using the call with `timeout`. The file can be then accessed locally with normal file I/O calls.

The source code follows:

```
#include <iostream>
#include <unistd.h> // For the unlink function
#include <fstream> // For file access inside "doSomethingWithFile" (ifstream)

extern "C"{
    #include "lcg_util.h"
}

using namespace std;

/* A function to further process the copied file... */
bool doSomethingWithFile(const char * pFile2){
    //....
}

int main(){
    /*
     * Parameters of the lcg_cp call
     * int lcg_cpt (char *src_file, char *dest_file, char *vo, int nbstreams,
     *             char *conf_file, int insecure, int verbose, int timeout);
     */
    char * src_file="lfn:my_lcg_cr_example";
    char * dest_file=new char[200];
    char * vo="dteam";
    int nbstreams=1;
```



```
// conf_file=0 (currently ignored)
// insecure=0 (currently ignored)
int verbose=1;
int timeout=180;

/* Form the name of the destination file */
char * my_file="my_retrieved_file";
char * pwd=getenv("PWD");
strcpy(dest_file,"file:");
strcat(dest_file,pwd);
strcat(dest_file,"/");
strcat(dest_file,my_file);

/* The lcg_cp call itself */
if(lcg_cp(src_file, dest_file, vo, nbstreams, 0, 0, verbose, timeout)==0){
    cout << "File correctly copied to local filesystem " << endl;
}
else{
    perror("Error with lcg_cp!");
}

/* Further processing */
doSomethingWithFile(my_file);

/* Cleaning... */

// Delete the temporary file
unlink(my_file);

/* That was it */
cout << endl;
return 0;

} //end of main
```

The program should be compilable with the following line:

```
$ /opt/gcc-3.2.2/bin/c++-3.2.2 -I$LCG_LOCATION/include \
-L$LCG_LOCATION/lib -L$GLOBUS_LOCATION/lib \
-llcg_util -lgfal -lglobus_gass_copy_gcc32 -o lcg_cp_example lcg_cp_example.cpp
```

**Note:** The linkage with `libglobus_gass_copy_gcc32.so` should not be necessary, and also the one with `libgfal.so` should be done transparently when linking `liblcg_util.so`. Nevertheless, their explicit linkage as shown in the example was necessary for the program to compile in the moment that this guide was written. At time of reading, it may not be necessary anymore.



### ***Example F.0.2 (Using GFAL to access a file)***

The following C++ code uses the `libgfal` library to access a Grid file, whose name is specified as a command line argument. The program opens the files, writes a set of numbers into it, and closes it. Afterwards, the files is opened again, and the previously written numbers are read and shown to the user. The source code (`gfal_example.cpp`) follows:

```
#include<iostream>
#include <fcntl.h>
#include <stdio.h>
extern "C" {
#include "/opt/lcg/include/gfal_api.h"
}

using namespace std;

/* Include the gfal functions (are C and not C++, therefore are 'extern') */
extern "C" {
    int gfal_open(const char*, int, mode_t);
    int gfal_write(int, const void*, size_t);
    int gfal_close(int);
    int gfal_read(int, void*, size_t);
}

/***** MAIN *****/
main(int argc, char **argv)
{
    int fd; // file descriptor
    int rc; // error codes
    size_t INTBLOCK=40; // how many bytes we will write each time (40 = 10 int a time)

    /* Check syntax (there must be 2 arguments) */
    if (argc != 2) {
        cerr << "Usage: " << argv[0]<<"filename\n";
        exit (1);
    }

    /* Declare and initialize the array of input values (to be written in the file) */
    int* original = new int[10];
    for (int i=0; i<10; i++) original[i]=i*10; // just: 0, 10, 20, 30...

    /* Declare and give size for the array that will store the values read from the file */
    int* readValues = new int[10];

    /* Create the file for writing with the given name */
    cout << "\nCreating file " << argv[1] << endl;
    if ((fd = gfal_open (argv[1], O_WRONLY | O_CREAT, 0644)) < 0) {
        perror ("gfal_open");
    }
}
```



```
    exit (1);
}
cout << " ... Open successful ... " ;

/* Write into the file (reading the 10 integers at once from the int array) */
if ((rc = gfal_write (fd, original, INTBLOCK )) != INTBLOCK) {
    if (rc < 0)    perror ("gfal_write");
    else cerr << "gfal_write returns " << rc << endl;
    (void) gfal_close (fd);
    exit (1);
}
cout << "Write successful ... ";

/* Close the file */
if ((rc = gfal_close (fd)) < 0) {
    perror ("gfal_close");
    exit (1);
}
cout << "Close successful" << endl;

/* Reopen the file for reading */
cout << "\nReading back " << argv[1] << endl;
if ((fd = gfal_open (argv[1], O_RDONLY, 0)) < 0) {
    perror ("gfal_open");
    exit (1);
}
cout << " ... Open successful ... ";

/* Read the file (40 bytes directly into the readValues array) */
if ((rc = gfal_read (fd, readValues, INTBLOCK )) != INTBLOCK) {
    if (rc < 0)    perror ("gfal_read");
    else cerr << "gfal_read returns " << rc << endl;
    (void) gfal_close (fd);
    exit (1);
}
cout << "Read successful ...";

/* Show what has been read */
for(int i=0; i<10; i++)
    cout << "\n\tValue of readValues[" << i << "] = " << readValues[i];

/* Close the file */
if ((rc = gfal_close (fd)) < 0) {
    perror ("gfal_close");
    exit (1);
}
cout << "\n ... Close successful";
cout << "\n\nDone" << endl;
```



```
}//end of main
```

The command used to compile and link the previous code (it may be different in your machine) is:

```
$ /opt/gcc-3.2.2/bin/c++-3.2.2 -L /opt/lcg/lib/ -l gfal -o gfal_example gfal_example.cpp
```

As temporary file, we may specify one in our local filesystem, by using the `file://` prefix. In that case we get the following output:

```
$ ./gfal_example file://`pwd`/test.txt

Creating file file:///afs/cern.ch/user/d/delgadop/gfal/test.txt
... Open successful ... Write successful ... Close successful

Reading back file:///afs/cern.ch/user/d/delgadop/gfal/test.txt
... Open successful ... Read successful ...
    Value of readValues[0] = 0
    Value of readValues[1] = 10
    Value of readValues[2] = 20
    Value of readValues[3] = 30
    Value of readValues[4] = 40
    Value of readValues[5] = 50
    Value of readValues[6] = 60
    Value of readValues[7] = 70
    Value of readValues[8] = 80
    Value of readValues[9] = 90
... Close successful

Done
```

We may define a JDL file and access a Grid file from a job. Indeed, a Grid file cannot be accessed directly from the UI if insecure RFIO is the protocol used for that access. However, the access from a job running in the same site where the file is stored is allowed. The reason for this is that insecure RFIO does not handle Grid certificates (secure RFIO does), and while the UID a user's job is mapped to will be allowed to access a file in a SE, the user's UID in the UI is different, and will not be allowed to perform that access.

In opposition to the insecure RFIO, the secure version, also called *gsirfio*, includes all the usual GSI security, and so it can deal with certificates rather than with users' UIDs. For this reason, it can be used with no problem to access files from UIs or in remote SEs. Just as *gsidcap* can.

**Attention:** Some SEs support only insecure RFIO (classic SEs and CASTOR), while others support only secure RFIO (dpm), but they all publish `rfio` as the supported protocol in the IS. The result is that currently GFAL has to figure out which one of the two RFIO versions it uses basing on an environmental



variable. This variable is called `LCG_RFIO_TYPE`. If its value is `dpm`, the secure version of RFIO will be used, if its value is `castor` or the variable is undefined, then insecure RFIO will be the one chosen.

Unfortunately, an insecure RFIO client cannot talk to a secure server and viceversa. Therefore, the user must correctly define the indicated variable depending on the SE he wants to talk to before using GFAL calls. Otherwise, the calls will not work.

Another important issue is that of the names used to access files.

For classic SEs, both the `SURL` and `TURL` names of the files must include a double slash between the hostname of the SE and the path of the file. This is needed by GFAL for RFIO. An example of correct `SURL` and `TURL` is:

```
sfn://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
rfio://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
```

These name requirements are imposed by the use of RFIO as access protocol. As seen in previous examples, the `lcg-*` commands will work with `SURLs` and `TURLs` registered in the catalogs, even if they do not follow this rules. This does not happen with RFIO. Therefore, it is always better to use `LFNs` or `GUIDs` when dealing with files, not to have to deal with `SURL` and `TURL` naming details.

**IMPORTANT!:** Nevertheless, the entries in the `RMC` and `LRC` may contain `SURLs` which do not comply with the described rules. As a result, when GFAL uses the `GUID` or `LFN` to retrieve the `SURL` of the file, it will get an incorrect one, and the call will fail. In those cases, using the correct `SURL` (which usually means doubling the slash after the hostname), instead of the `GUID` or `LFN`, is the only way to access the file.

Having this all in mind, let us build a `JDL` file to create and read a Grid file with our C++ program:

```
Executable="gfal_example";
StdOutput="std.out";
StdError="std.err";
Arguments="sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file";
InputSandbox={"gfal_example"};
OutputSandbox={"std.out", "std.err"};
```

After submitting the job, the output retrieved in `std.out` is as follows:

```
Creating file sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file
... Open successful ... Write successful ... Close successful

Reading back sfn://lxb0707.cern.ch//flatfiles/SE00/dteam/my_temp_file
... Open successful ... Read successful ...
    Value of readValues[0] = 0
    Value of readValues[1] = 10
```



```
Value of readValues[2] = 20
Value of readValues[3] = 30
Value of readValues[4] = 40
Value of readValues[5] = 50
Value of readValues[6] = 60
Value of readValues[7] = 70
Value of readValues[8] = 80
Value of readValues[9] = 90
... Close successful
```

Done

It is important to notice that the creation of a new file using GFAL does not imply the registration of that file. This means that if the created file has to be used as a Grid file, then it should be manually registered using `lcg-rf`. Otherwise, the file should be deleted using `edg-gridftp-rm`.

As seen, by using GFAL, an application can access a file remotely almost as if it was local (substituting POSIX calls by those of GFAL). For more information on GFAL, refer to [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.

Finally, as was explained previously, insecure RFIO can only be used in order to access files that are located in the same local area network where the CE holding the job is located. In order to access or move files between different sites, the user should use a different method. Of course, if the only way to remotely access a file is insecure RFIO (as is the case for classic SEs or CASTOR), then GFAL calls will also use insecure RFIO as the protocol for the interaction and therefore this last limitation will also apply.

Although direct use of RFIO's APIs is discouraged, information on it and its APIs can be found in [R26].

### ***Example F.0.3 (Explicit interaction with the SRM using GFAL)***

The following example program can be useful for copying a file that is stored in a MSS. It asks for the file to be staged from tape to disk first, and only tries to copy it when the file has been migrated.

The program uses both the `lcg_util` and the GFAL APIs. From `lcg_util`, just the `lcg_cp` call is used.



From GFAL, `srm_get`, which requests a file to be staged from tape to disk, and `srm_get_status`, which checks the status of the previous request, are used.

The source code follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <iostream>
#include <sstream> // for the integer to string conversion
#include <unistd.h> // for the sleep function
#include <fstream> // for the local file access
extern "C"{
    #include "gfal_api.h"
    #include "lcg_util.h"
}

using namespace std;

main(int argc, char ** argv){
/* Check arguments */
    if ((argc < 2) || (argc > 2)) {
        cerr << "Usage: " << argv[0] << " SURL\n";
        exit (1);
    }

/*
 * Try to get the file (stage in)
 * int srm_get (int nbfiles, char **surls, int nbprotocols, char **protocols, int *reqid,
 *             char **token, struct srm_filestatus **filestatuses, int timeout);
 *
 * struct srm_filestatus{
 *   char *surl;
 *   char *turl;
 *   int fileid;
 *   int status;};
 */
    int nbreplies; //number of replies returned
    int nbfiles=1; // number of files
    char **surls; // array of SURLs
    int nbprotocols; // number of bytes of the protocol array
    char * protocols[] = {"rfio"}; // protocols
    int reqid; // request ID
    //char **token=0; // unused
    struct srm_filestatus *filestatuses; // status of the files
    int timeout=100;

/* Set the SURL and the nbprotocols */
    surls = &argv[1];
```





```
nbprotocols = sizeof(protocols) / sizeof(char *);

/* Make the call */
if ((nbreplies = srm_get (nbfiles, surls, nbprotocols, protocols,
    &reqid, 0, &filestatuses, timeout)) < 0) {
    perror ("Error in srm_get");
    exit (-1);
}

/* Show the retrieved information */
cout << "\nThe status of the file is: " << endl;
cout << endl << filestatuses[0].status << " -- " << filestatuses[0].surl;
free(filestatuses[0].surl);
if(filestatuses[0].status == 1){
    cout << " (" << filestatuses[0].turl << ")" << endl;
    free(filestatuses[0].turl);
}
else {cout << endl;}
free(filestatuses);

if(filestatuses[0].status == -1){
    cout << endl << "Error when trying to stage the file. Not waiting..." << endl;
    exit(-1);
}

/*
 * Now watch the status until it gets to STAGED (1)
 * int srm_getstatus (int nbfiles, char **surls, int reqid, char **token,
 *                    struct srm_filestatus **filestatuses, int timeout);
 */
cout << "\nWaiting for the file to be staged in..." << endl;
int numiter=1;
int filesleft=1;

char * destfile = new char[200];
while((numiter<50) && (filesleft>0)){
    //sleep longer each iteration
    sleep(numiter++);
    cout << "#"; // just to show we are waiting and not dead
    cout.flush();

    if ((nbreplies = srm_getstatus (nbfiles, surls, reqid, NULL, &filestatuses, timeout))
        < 0) {
        perror ("srm_getstatus");
        exit (-1);
    }

    if (filestatuses[0].status == 1){
```



```
    cout << "\nREADY  -- " << filestatuses[0].surl << endl;
    filesleft--;
    // Create a name for the file to be saved
    strcpy(destfile, "file:/tmp/srm_gfal_retrieved");
    cout << "\nCopying " << filestatuses[0].surl << " to " << destfile << "... \n";
    // Copy the file to the local filesystem
    if(lcg_cp(filestatuses[0].surl, destfile, "dteam", 1, 0, 0 , 1)!=0){
        perror("Error in lcg_cp");
    }
}
free(filestatuses[0].surl);
if(filestatuses[0].status == 1)    free(filestatuses[0].turl);
free(filestatuses);
}

if(numiter>49){
    cout << "\nThe file did not reach the READY status. It could not be copied." << endl;
}

/* Cleaning */
delete [] destfile;

/* That was all */
cout << endl;
return reqid; // return the reqid, so that it can be used by the caller
} //end of main
```

The `srm_get` function is called once to request the staging of the file. In this call, we retrieve the corresponding TURL and some numbers identifying the request. If a LFN was provided, several TURLs (from several replicas) could be retrieved. In this case, only one TURL will be returned (stored in the first position of the `filestatuses` array).

The second part of the program is a loop that will repeatedly call `srm_getstatus` in order to get the current status of the previous request, until the status is equal to 1 (ready). There is a `sleep` call to let the program wait some time (time increasing with each iteration) for the file staging. Also a maximum number of iterations is set (50), so that the program does not wait for ever, but rather ends finally with an aborting message.

When the file is ready, it is copied using `lcg_cp` in the same way as seen in a previous example. This or other application should then perform some operation on the file (this is not shown here).

A possible output of this program is the following:

The status of the file is:



```
0 -- srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Waiting for the file to be staged in...
#####

READY -- srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Copying srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1 to
file:/tmp/srm_gfal_retrieved...
Source URL: srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
File size: 2331
Source URL for copy:
gsiftp://castorgrid.cern.ch:2811//shift/lxfs5614/data03/cg/stage/test_1.172962
Destination URL: file:/tmp/srm_gfal_retrieved
# streams: 1
Transfer took 590 ms
```

Where the 0 file status means that the file exists but it lays on the tape (not staged yet), the hash marks show the iterations in the looping and finally the READY indicates that the file has been staged in and it can be copied (what it is done afterwards as shown by the normal verbose output).

If the same program was run again, passing the same SURL as argument, it would return almost immediately, since the file has been already staged. This is shown in the following output:

```
The status of the file is:

1 -- srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
(rfio://lxfs5614//shift/lxfs5614/data03/cg/stage/test_1.172962)

Waiting for the file to be staged in...
#
READY -- srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1

Copying srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1 to
file:/tmp/srm_gfal_retrieved...
Source URL: srm://castorsrm.cern.ch/castor/cern.ch/grid/dteam/testSRM/test_1
File size: 2331
Source URL for copy:
gsiftp://castorgrid.cern.ch:2811//shift/lxfs5614/data03/cg/stage/test_1.172962
Destination URL: file:/tmp/srm_gfal_retrieved
# streams: 1
Transfer took 550 ms
```

Where the 1 file status means that the file is already in disk.



## APPENDIX G THE GLUE SCHEMA

As explained earlier, the GLUE Schema describes the Grid resources information that is stored by the Information System. This section gives information about the MDS, namely the LDAP implementation of the GLUE Schema, which is currently used in the LCG-2 IS. For information on the abstract GLUE Schema definition, please refer to [R12].

First of all, the tree of object classes definitions is shown. Then, the attributes for each one of the objectclasses (where the data are actually stored) are presented. Some of the attributes may be currently empty, even if they are defined in the schema. Furthermore, some new attributes may be published, although not yet collected in the schema. Finally, the DITs currently used in the IS for the publishing of these attributes are shown.

### G.1. THE GLUE SCHEMA LDAP OBJECT CLASSES TREE

```
Top
|
---- GlueTop 1.3.6.1.4.1.8005.100
|
---- .1. GlueGeneralTop
|
|   ---- .1. ObjectClass
|   |
|   |   ---- .1 GlueSchemaVersion
|   |   |
|   |   ---- .2 GlueCESEBindGroup
|   |   |
|   |   ---- .3 GlueCESEBind
|   |   |
|   |   ---- .4 GlueKey
|   |   |
|   |   ---- .5 GlueInformationService
|   |   |
|   |   ---- .6 GlueService
|   |   |
|   |   ---- .7 GlueServiceData
|   |   |
|   |   ---- .8 GlueSite
|   |
|   ---- .2. Attributes
|   |
|   |   ---- .1. Attributes for GlueSchemaVersion
|   |   |
|   |   |   . . .
|   |   |
|   |   ---- .8. Attributes for GlueSiteTop
```

```
|
|
|---- .2. GlueCETop
|   |
|   |---- .1. ObjectClass
|   |   |
|   |   |---- .1 GlueCE
|   |   |   |
|   |   |   |---- .2 GlueCEInfo
|   |   |   |   |
|   |   |   |   |---- .3 GlueCEState
|   |   |   |   |   |
|   |   |   |   |   |---- .4 GlueCEPolicy
|   |   |   |   |   |   |
|   |   |   |   |   |   |---- .5 GlueCEAccessControlBase
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |---- .6 GlueCEJob
|   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |---- .7 GlueVOView
|   |   |   |   |   |   |   |   |   |
|   |   |---- .2. Attributes
|   |   |   |
|   |   |   |---- .1. Attributes for GlueCE
|   |   |   |   |
|   |   |   |   |---- .7. Attributes for GlueVOView
|   |   |   |   |   |
|   |   |---- .3. MyObjectClass
|   |   |   |
|   |   |---- .4. MyAttributes
|   |   |   |
|   |---- .3. GlueClusterTop
|   |   |
|   |   |---- .1. ObjectClass
|   |   |   |
|   |   |   |---- .1 GlueCluster
|   |   |   |   |
|   |   |   |   |---- .2 GlueSubCluster
|   |   |   |   |   |
|   |   |   |   |   |---- .3 GlueHost
|   |   |   |   |   |   |
|   |   |   |   |   |   |---- .4 GlueHostArchitecture
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |---- .5 GlueHostProcessor
|   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |---- .6 GlueHostApplicationSoftware
|   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |---- .7 GlueHostMainMemory
|   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |---- .8 GlueHostBenchmark
```



```

|     ---- .3. MyObjectClass
|     |
|     ---- .4. MyAttributes
|
| ---- .5. GlueSLTop
|     |
|     ---- .1. ObjectClass
|     |
|     |     ---- .1 GlueSL
|     |     |
|     |     ---- .2 GlueSLLocalFileSystem
|     |     |
|     |     ---- .3 GlueSLRemoteFileSystem
|     |     |
|     |     ---- .4 GlueSLFile
|     |     |
|     |     ---- .5 GlueSLDirectory
|     |     |
|     |     ---- .6 GlueSLArchitecture
|     |     |
|     |     ---- .7 GlueSLPerformance
|     |
|     ---- .2. Attributes
|     |
|     |     ---- .1. Attributes for GlueSL
|     |     |
|     |     |     . . .
|     |     |
|     |     ---- .7 Attributes for GlueSLPerformance
|     |
|     ---- .3. MyObjectClass
|     |
|     ---- .4. MyAttributes
|
| ---- .6. GlueSATop
|     |
|     ---- .1. ObjectClass
|     |
|     |     ---- .1 GlueSA
|     |     |
|     |     ---- .2 GlueSAPolicy
|     |     |
|     |     ---- .3 GlueSAState
|     |     |
|     |     ---- .4 GlueSAAccessControlBase
|     |
|     ---- .2. Attributes
|     |
|     |     ---- .1. Attributes for GlueSA
|     |     |
|     |     |     . . .

```

```
|      |
|      |---- .4  Attributes for GlueSAAccessControlBase
|
|---- .3. MyObjectClass
|
|---- .4. MyAttributes
```

## G.2. GENERAL ATTRIBUTES

This group includes some base (top) object classes, which have no attributes, and, thus, no actual resource data, and some other that include general attributes that are defined in entries of both CEs and SEs. These are the version of the schema that is used, the URL of the IS server and finally the `GlueKey`, which is used to relate different entries of the tree and in this way overcome OpenLDAP limitations in query flexibility.

- **Base class (objectclass `GlueTop`)**
  - No attributes
- **Base class for general object classes, attributes, matching rules, etc. (objectclass `GlueGeneralTop`)**
  - No attributes
- **Schema Version Number (objectclass `GlueSchemaVersion`)**
  - `GlueSchemaVersionMajor`: Major Schema version number
  - `GlueSchemaVersionMinor`: Minor Schema version number
- **Internal attributes to express object associations (objectclass `GlueKey`)**
  - `GlueChunkKey`: Relative DN (`AttributeType=AttributeValue`) to reference a related entry in the same branch than this DN
  - `GlueForeignKey`: Relative DN (`AttributeType=AttributeValue`) to reference a related entry in a different branch
- **Information for the Information Service (objectclass `GlueInformationService`)**
  - `GlueInformationServiceURL`: The Information Service URL publishing the related information
- **Service entity (objectclass `GlueService`)**
  - `GlueServiceUniqueID`: unique identifier of the service
  - `GlueServiceName`: human-readable name of the service
  - `GlueServiceType`: type of service



- `GlueServiceVersion`: version of the service: major.minor.patch
- `GlueServiceEndpoint`: network endpoint for the service
- `GlueServiceStatus`: status of the service (OK, Warning, Critical, Unknown, Other)
- `GlueServiceStatusInfo`: textual explanation of the status
- `GlueServiceWSDL`: URI of the service WSDL
- `GlueServiceSemantics`: URL of a detailed description of the service
- `GlueServiceStartTime`: time of last service start
- `GlueServiceOwner`: owner of the service (e.g. the VO)

The attributes `GlueServicePrimaryOwnerName`, `GlueServicePrimaryOwnerContact`, `GlueServiceHostingOrganization`, `GlueServiceMajorVersion`, `GlueServiceMinorVersion`, `GlueServicePatchVersion`, `GlueServiceAccessControlRule` and `GlueServiceInformationServiceURL` are deprecated from version 1.2 of the GLUE schema.

### G.3. ATTRIBUTES FOR THE COMPUTING ELEMENT

These are attributes that give information about a CE and its composing WNs. In the GLUE Schema, they are defined in the UML diagram for the Computing Element.

- **Base class for the CE information (objectclass `GlueCETop`)**
  - No attributes
- **Base class for the cluster information (objectclass `GlueClusterTop`)**
  - No attributes
- **Computing Element (objectclass `GlueCE`)**
  - `GlueCEUniqueID`: unique identifier for the CE
  - `GlueCEName`: human-readable name of the service
- **General Info for the queue associated to the CE (objectclass `GlueCEInfo`)**
  - `GlueCEInfoLRMSType`: name of the local batch system
  - `GlueCEInfoLRMSVersion`: version of the local batch system
  - `GlueCEInfoGRAMVersion`: version of GRAM
  - `GlueCEInfoHostName`: fully qualified name of the host where the gatekeeper runs
  - `GlueCEInfoGateKeeperPort`: port number for the gatekeeper



- `GlueCEInfoTotalCPUs`: number of CPUs in the cluster associated to the CE
- `GlueCEInfoContactString`: contact string for the service
- `GlueCEInfoJobManager`: job manager used by the gatekeeper
- `GlueCEInfoApplicationDir`: path of the directory for application installation
- `GlueCEInfoDataDir`: path a shared the directory for application data
- `GlueCEInfoDefaultSE`: unique identifier of the default SE

- **CE State (objectclass `GlueCEState`)**

- `GlueCEStateStatus`: queue status: queueing (jobs are accepted but not run), production (jobs are accepted and run), closed (jobs are neither accepted nor run), draining (jobs are not accepted but those in the queue are run)
- `GlueCEStateTotalJobs`: total number of jobs (running + waiting)
- `GlueCEStateRunningJobs`: number of running jobs
- `GlueCEStateWaitingJobs`: number of jobs not running
- `GlueCEStateWorstResponseTime`: worst possible time between the submission of a job and the start of its execution, in seconds
- `GlueCEStateEstimatedResponseTime`: estimated time between the submission of a job and the start of its execution, in seconds
- `GlueCEStateFreeCPUs`: number of CPUs available to the scheduler
- `GlueCEStateFreeJobSlots`: number of jobs that could start, given the current number of jobs submitted

- **CE Policy (objectclass `GlueCEPolicy`)**

- `GlueCEPolicyMaxWallClockTime`: maximum wall clock time available to jobs submitted to the CE, in minutes
- `GlueCEPolicyMaxCPUtime`: maximum CPU time available to jobs submitted to the CE, in minutes
- `GlueCEPolicyMaxTotalJobs`: maximum allowed total number of jobs in the queue
- `GlueCEPolicyMaxRunningJobs`: maximum allowed number of running jobs in the queue
- `GlueCEPolicyPriority`: information about the service priority
- `GlueCEPolicyAssignedJobSlots`: maximum number of single-processor jobs that can be running at a given time

- **Access control (objectclass `GlueCEAccessControlBase`)**

- `GlueCEAccessControlBaseRule`: a rule defining any access restrictions to the CE. Current semantic: VO = a VO name, DENY = an X.509 user subject

- **Job (currently not filled, the Logging and Bookkeeping service can provide this information) (objectclass `GlueCEJob`)**



- `GlueCEJobLocalOwner`: local user name of the job's owner
- `GlueCEJobGlobalOwner`: GSI subject of the real job's owner
- `GlueCEJobLocalID`: local job identifier
- `GlueCEJobGlobalId`: global job identifier
- `GlueCEJobGlueCEJobStatus`: job status: SUBMITTED, WAITING, READY, SCHEDULED, RUNNING, ABORTED, DONE, CLEARED, CHECKPOINTED
- `GlueCEJobSchedulerSpecific`: any scheduler specific information

- **Cluster (objectclass `GlueCluster`)**

- `GlueClusterUniqueID`: unique identifier for the cluster
- `GlueClusterName`: human-readable name of the cluster

The attribute `GlueClusterService` is deprecated from version 1.2 of the GLUE schema.

- **Subcluster (objectclass `GlueSubCluster`)**

- `GlueSubClusterUniqueID`: unique identifier for the subcluster
- `GlueSubClusterName`: human-readable name of the subcluster
- `GlueSubClusterTmpDir`: path of temporary directory shared among worker nodes
- `GlueSubClusterWNTmpDir`: path of temporary directory local to the worker nodes
- `GlueSubClusterPhysicalCPUs`: total number of real CPUs in the subcluster
- `GlueSubClusterLogicalCPUs`: total number of logical CPUs (e.g. with hyperthreading)

- **Host (objectclass `GlueHost`)**

- `GlueHostUniqueId`: unique identifier for the host
- `GlueHostName`: human-readable name of the host

- **Architecture (objectclass `GlueHostArchitecture`)**

- `GlueHostArchitecturePlatformType`: platform description
- `GlueHostArchitectureSMPSize`: number of CPUs in a SMP node

- **Processor (objectclass `GlueHostProcessor`)**

- `GlueHostProcessorVendor`: name of the CPU vendor
- `GlueHostProcessorModel`: name of the CPU model
- `GlueHostProcessorVersion`: version of the CPU
- `GlueHostProcessorClockSpeed`: clock speed of the CPU
- `GlueHostProcessorInstructionSet`: name of the instruction set architecture of the CPU
- `GlueHostProcessorOtherProcessorDescription`: other description for the CPU

- 
- `GlueHostProcessorCacheL1`: size of the unified L1 cache
  - `GlueHostProcessorCacheL1I`: size of the instruction L1 cache
  - `GlueHostProcessorCacheL1D`: size of the data L1 cache
  - `GlueHostProcessorCacheL2`: size of the unified L2 cache
- **Application software (objectclass `GlueHostApplicationSoftware`)**
    - `GlueHostApplicationSoftwareRunTimeEnvironment`: list of software installed on this host
- **Main memory (objectclass `GlueHostMainMemory`)**
    - `GlueHostMainMemoryRAMSize`: physical RAM
    - `GlueHostMainMemoryRAMAvailable`: unallocated RAM
    - `GlueHostMainMemoryVirtualSize`: size of the configured virtual memory
    - `GlueHostMainMemoryVirtualAvailable`: available virtual memory
- **Benchmark (objectclass `GlueHostBenchmark`)**
    - `GlueHostBenchmarkSI00`: `SpecInt2000` benchmark
    - `GlueHostBenchmarkSF00`: `SpecFloat2000` benchmark
- **Network adapter (objectclass `GlueHostNetworkAdapter`)**
    - `GlueHostNetworkAdapterName`: name of the network card
    - `GlueHostNetworkAdapterIPAddress`: IP address of the network card
    - `GlueHostNetworkAdapterMTU`: the MTU size for the LAN to which the network card is attached
    - `GlueHostNetworkAdapterOutboundIP`: permission for outbound connectivity
    - `GlueHostNetworkAdapterInboundIP`: permission for inbound connectivity
- **Processor load (objectclass `GlueHostProcessorLoad`)**
    - `GlueHostProcessorLoadLast1Min`: one-minute average processor availability for a single node
    - `GlueHostProcessorLoadLast5Min`: 5-minute average processor availability for a single node
    - `GlueHostProcessorLoadLast15Min`: 15-minute average processor availability for a single node
- **SMP load (objectclass `GlueHostSMPLoad`)**
    - `GlueHostSMPLoadLast1Min`: one-minute average processor availability for a single node
    - `GlueHostSMPLoadLast5Min`: 5-minute average processor availability for a single node
    - `GlueHostSMPLoadLast15Min`: 15-minute average processor availability for a single node



- **Operating system (objectclass `GlueHostOperatingSystem`)**

- `GlueHostOperatingSystemOSName`: OS name
- `GlueHostOperatingSystemOSRelease`: OS release
- `GlueHostOperatingSystemOSVersion`: OS or kernel version

- **Local file system (objectclass `GlueHostLocalFileSystem`)**

- `GlueHostLocalFileSystemRoot`: path name or other information defining the root of the file system
- `GlueHostLocalFileSystemSize`: size of the file system in bytes
- `GlueHostLocalFileSystemAvailableSpace`: amount of free space in bytes
- `GlueHostLocalFileSystemReadOnly`: true if the file system is read-only
- `GlueHostLocalFileSystemType`: file system type
- `GlueHostLocalFileSystemName`: the name for the file system
- `GlueHostLocalFileSystemClient`: host unique identifier of clients allowed to remotely access this file system

- **Remote file system (objectclass `GlueHostRemoteFileSystem`)**

- `GlueHostRemoteFileSystemRoot`: path name or other information defining the root of the file system
- `GlueHostRemoteFileSystemSize`: size of the file system in bytes
- `GlueHostRemoteFileSystemAvailableSpace`: amount of free space in bytes
- `GlueHostRemoteFileSystemReadOnly`: true if the file system is read-only
- `GlueHostRemoteFileSystemType`: file system type
- `GlueHostRemoteFileSystemName`: the name for the file system
- `GlueHostRemoteFileSystemServer`: host unique id of the server which provides access to the file system

- **Storage device (objectclass `GlueHostStorageDevice`)**

- `GlueHostStorageDeviceName`: name of the storage device
- `GlueHostStorageDeviceType`: storage device type
- `GlueHostStorageDeviceTransferRate`: maximum transfer rate for the device
- `GlueHostStorageDeviceSize`: size of the device
- `GlueHostStorageDeviceAvailableSpace`: amount of free space

- **File (objectclass `GlueHostFile`)**

- `GlueHostFileName`: name for the file
- `GlueHostFileSize`: file size in bytes

- `GlueHostFileCreationDate`: file creation date and time
- `GlueHostFileLastModified`: date and time of the last modification of the file
- `GlueHostFileLastAccessed`: date and time of the last access to the file
- `GlueHostFileLatency`: time taken to access the file, in seconds
- `GlueHostFileLifeTime`: time for which the file will stay on the storage device
- `GlueHostFileOwner`: name of the owner of the file

- **Location (objectclass `GlueLocation`)**

- `GlueLocationLocalID`: local ID for the location
- `GlueLocationName`: name
- `GlueLocationPath`: path
- `GlueLocationVersion`: version

- **VO View (objectclass `GlueVOView`)**

- `GlueVOViewLocalID`: local ID for this VO view

#### **G.4. ATTRIBUTES FOR THE STORAGE ELEMENT**

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

It is worth noting that the `GlueSE` object class, which maps to the `StorageService` element in the GLUE Schema, publishes information of the manager service of a SE; the `GlueSL` object class, which maps to the `StorageLibrary` element, publishes information related to the access node for the SE; and the `GlueSA` object class, which maps to the `StorageSpace` element, gives information about the available space in the SE.

- **Base Class for the storage service (objectclass `GlueSETop`)**

- No attributes

- **Base Class for the storage library (objectclass `GlueSLTop`)**

- No attributes

- **Base Class for the storage space (objectclass `GlueSATop`)**

- No attributes

- **Storage Service (objectclass `GlueSE`)**



- `GlueSEUniqueId`: unique identifier of the storage service (URI)
- `GlueSEName`: human-readable name for the service
- `GlueSEPort`: port number that the service listens
- `GlueSEHostingSL`: unique identifier of the storage library hosting the service
- `GlueSESizeTotal`: the total size of the storage space managed by the service
- `GlueSESizeFree`: the size of the storage capacity that is free for new areas for any VO/user
- `GlueSEArchitecture`: underlying architectural system category

The attribute `GlueSEType` is deprecated from version 1.2 of the GLUE schema.

- **Storage Service State (objectclass `GlueSEState`)**

- `GlueSEStateCurrentIOLoad`: system load (for example, number of files in the queue)

- **Storage Service Access Protocol (objectclass `GlueSEAccessProtocol`)**

- `GlueSEAccessProtocolType`: protocol type to access or transfer files
- `GlueSEAccessProtocolPort`: port number for the protocol
- `GlueSEAccessProtocolVersion`: protocol version
- `GlueSEAccessProtocolSupportedSecurity`: security features supported by the protocol
- `GlueSEAccessProtocolAccessTime`: time to access a file using this protocol
- `GlueSEAccessProtocolLocalID`: local identifier
- `GlueSEAccessProtocolEndpoint`: network endpoint for this protocol
- `GlueSEAccessProtocolCapability`: function supported by this control protocol

- **Protocol details (objectclass `GlueSEControlProtocol`)**

- `GlueSEControlProtocolType`: protocol type (e.g. `srmv1`)
- `GlueSEControlProtocolVersion`: protocol version
- `GlueSEControlProtocolLocalID`: local identifier
- `GlueSEControlProtocolLocalID`: network endpoint for this protocol
- `GlueSEControlProtocolCapability`: function supported by this control protocol

- **Storage Library (objectclass `GlueSL`)**

- `GlueSLName`: human-readable name of the storage library
- `GlueSLUniqueID`: unique identifier of the machine providing the storage service
- `GlueSLService`: unique identifier for the provided storage service

- **Local File system (objectclass `GlueSLLocalFileSystem`)**

- `GlueSLLocalFileSystemRoot`: path name (or other information) defining the root of the file system



- `GlueSLLocalFileSystemName`: name of the file system
- `GlueSLLocalFileSystemType`: file system type (e.g. NFS, AFS, etc.)
- `GlueSLLocalFileSystemReadOnly`: `true` is the file system is read-only
- `GlueSLLocalFileSystemSize`: total space assigned to this file system
- `GlueSLLocalFileSystemAvailableSpace`: total free space in this file system
- `GlueSLLocalFileSystemClient`: unique identifiers of clients allowed to access the file system remotely

- **Remote File system (objectclass `GlueSLRemoteFileSystem`)**

- `GlueSLRemoteFileSystemRoot`: path name (or other information) defining the root of the file system
- `GlueSLRemoteFileSystemSize`: total space assigned to this file system
- `GlueSLRemoteFileSystemAvailableSpace`: total free space in this file system
- `GlueSLRemoteFileSystemReadOnly`: `true` is the file system is read-only
- `GlueSLRemoteFileSystemType`: file system type (e.g. NFS, AFS, etc.)
- `GlueSLRemoteFileSystemName`: name of the file system
- `GlueSLRemoteFileSystemServer`: unique identifier of the server exporting this file system

- **File Information (objectclass `GlueSLFile`)**

- `GlueSLFileName`: file name
- `GlueSLFileSize`: file size
- `GlueSLFileCreationDate`: file creation date and time
- `GlueSLFileLastModified`: date and time of the last modification of the file
- `GlueSLFileLastAccessed`: date and time of the last access to the file
- `GlueSLFileLatency`: time needed to access the file
- `GlueSLFileLifeTime`: file lifetime
- `GlueSLFilePath`: file path

- **Directory Information (objectclass `GlueSLDirectory`)**

- `GlueSLDirectoryName`: directory name
- `GlueSLDirectorySize`: directory size
- `GlueSLDirectoryCreationDate`: directory creation date and time
- `GlueSLDirectoryLastModified`: date and time of the last modification of the directory
- `GlueSLDirectoryLastAccessed`: date and time of the last access to the directory
- `GlueSLDirectoryLatency`: time needed to access the directory
- `GlueSLDirectoryLifeTime`: directory lifetime





- 
- GlueSLDirectoryPath: directory path
  - **Architecture (objectclass GlueSLArchitecture)**
    - GlueSLArchitectureType: type of storage hardware (i.e. disk, RAID array, tape library, etc.)
  - **Performance (objectclass GlueSLPerformance)**
    - GlueSLPerformanceMaxIOCapacity: maximum bandwidth between the service and the network
  - **Storage Space (objectclass GlueSA)**
    - GlueSARoot: pathname of the directory containing the files of the storage space
    - GlueSALocalID: local identifier
    - GlueSAPath: root path of the area
    - GlueSAType: guarantee on the lifetime for the storage area (permanent, durable, volatile, other)
    - GlueSAUniqueID: unique identifier
  - **Policy (objectclass GlueSAPolicy)**
    - GlueSAPolicyMaxFileSize: maximum file size
    - GlueSAPolicyMinFileSize: minimum file size
    - GlueSAPolicyMaxData: maximum allowed amount of data that a single job can store
    - GlueSAPolicyMaxNumFiles: maximum allowed number of files that a single job can store
    - GlueSAPolicyMaxPinDuration: maximum allowed lifetime for non-permanent files
    - GlueSAPolicyQuota: total available space
    - GlueSAPolicyFileLifeTime: lifetime policy for the contained files
  - **State (objectclass GlueSAState)**
    - GlueSAStateAvailableSpace: total space available in the storage space (in kilobytes)
    - GlueSAStateUsedSpace: used space in the storage space (in kilobytes)
  - **Access Control Base (objectclass GlueSAAccessControlBase)**
    - GlueSAAccessControlBase Rule: list of the access control rules



## G.5. ATTRIBUTES FOR THE CE-SE BINDING

The CE-SE binding schema represents a mean for advertising relationships between a CE and a SE (or several SEs). This is defined by site administrators and is used when scheduling jobs that must access input files or create output files from or to SEs. In the GLUE Schema, they are defined in the UML diagram for the Computing Element - Storage Service - Bind.

- **Associations between an CE and one or more SEs (objectclass `GlueCESEBindGroup`)**

- `GlueCESEBindGroupCEUniqueID`: unique ID for the CE
- `GlueCESEBindGroupSEUniqueID`: unique ID for the SE

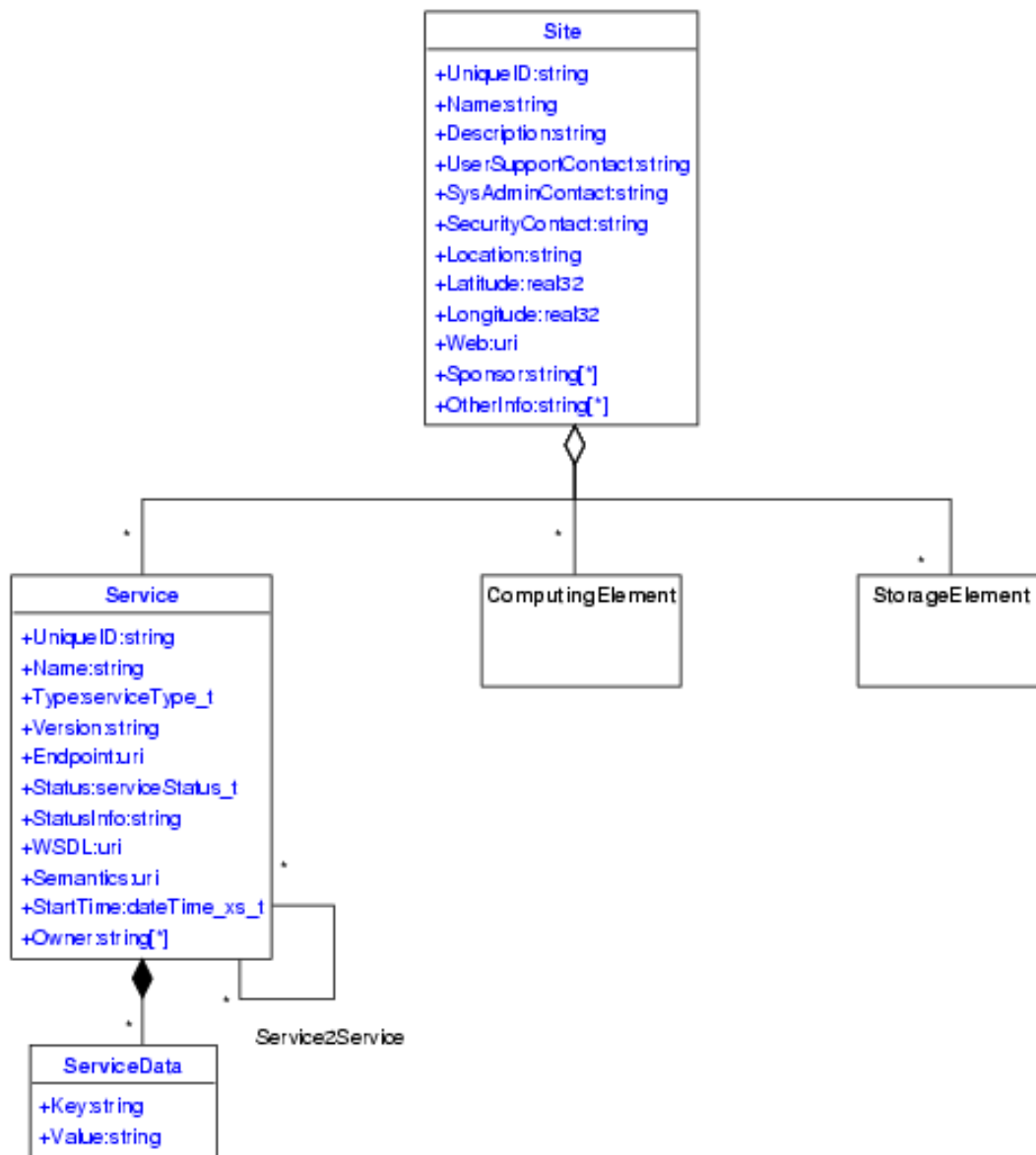
- **Associations between an SE and a CE (objectclass `GlueCESEBind`)**

- `GlueCESEBindCEUniqueID`: unique ID for the CE
- `GlueCESEBindCEAccesspoint`: access point in the cluster from which CE can access a local SE
- `GlueCESEBindSEUniqueID`: unique ID for the SE
- `GlueCESEBindMountInfo`: information about the name of the mount directory on the worker nodes of the CE and the exported directory from the SE
- `GlueCESEBindWeight`: it expresses a preference when multiple SEs are bound to a CE

## G.6. THE DIT USED BY THE MDS

The DITs used in the local BDIIs, the GRISes of a CE and a SE are shown in the Figures 17, 18, 19, 20 and 21. The GRIS of a CE contains information for computing resources (different entries for the different queues) and also for the computing-storage service relationships. A GRIS located in a SE publishes information for the storage resources.

The DITs of every GRIS in a site are included in the DIT of the site BDII, grouping all the information of the Grid resources in that site. The information of the different site BDIIs is compiled in a global BDII, as described in Section 5.1.5.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 17: DIT for the core information

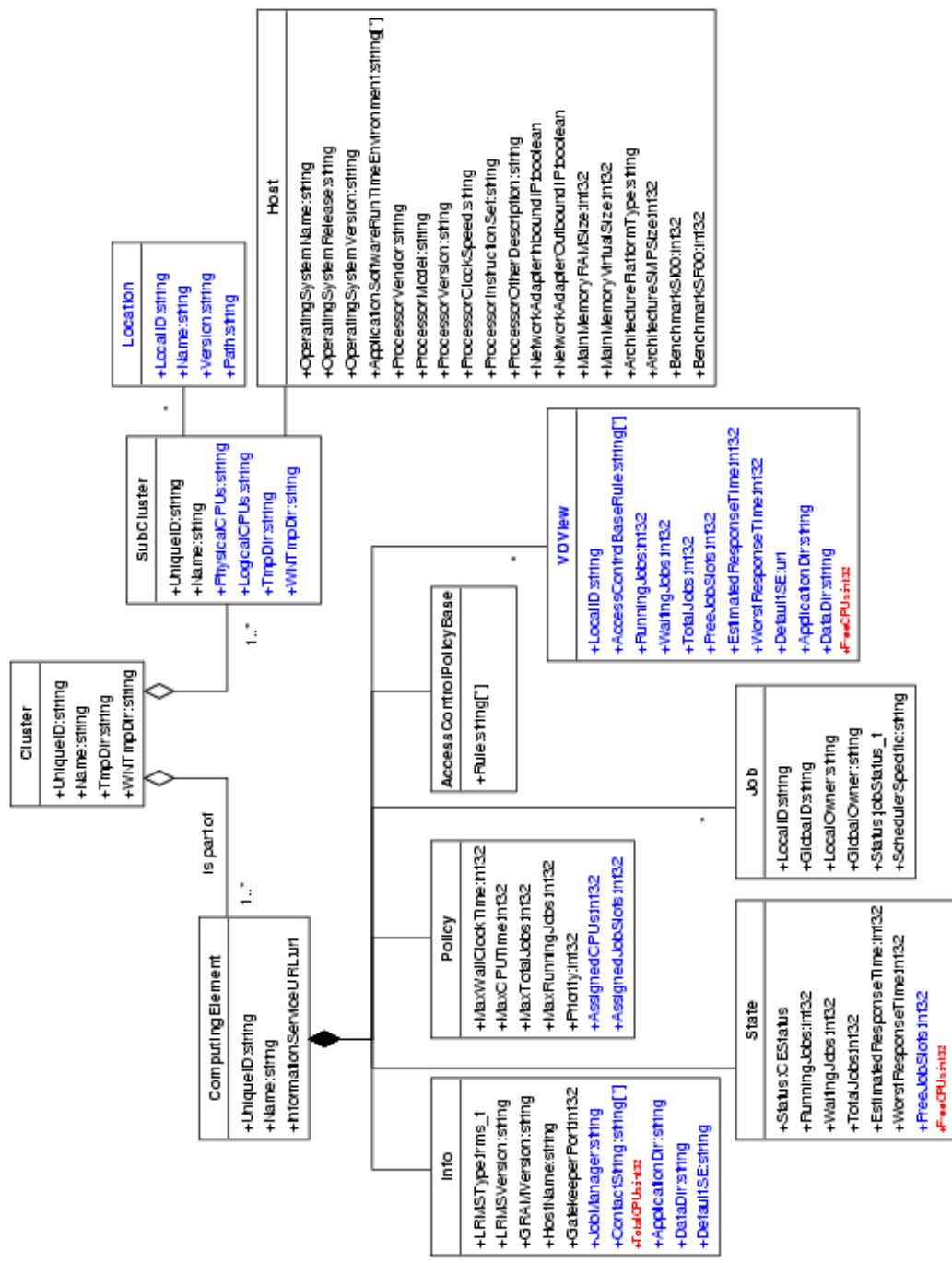
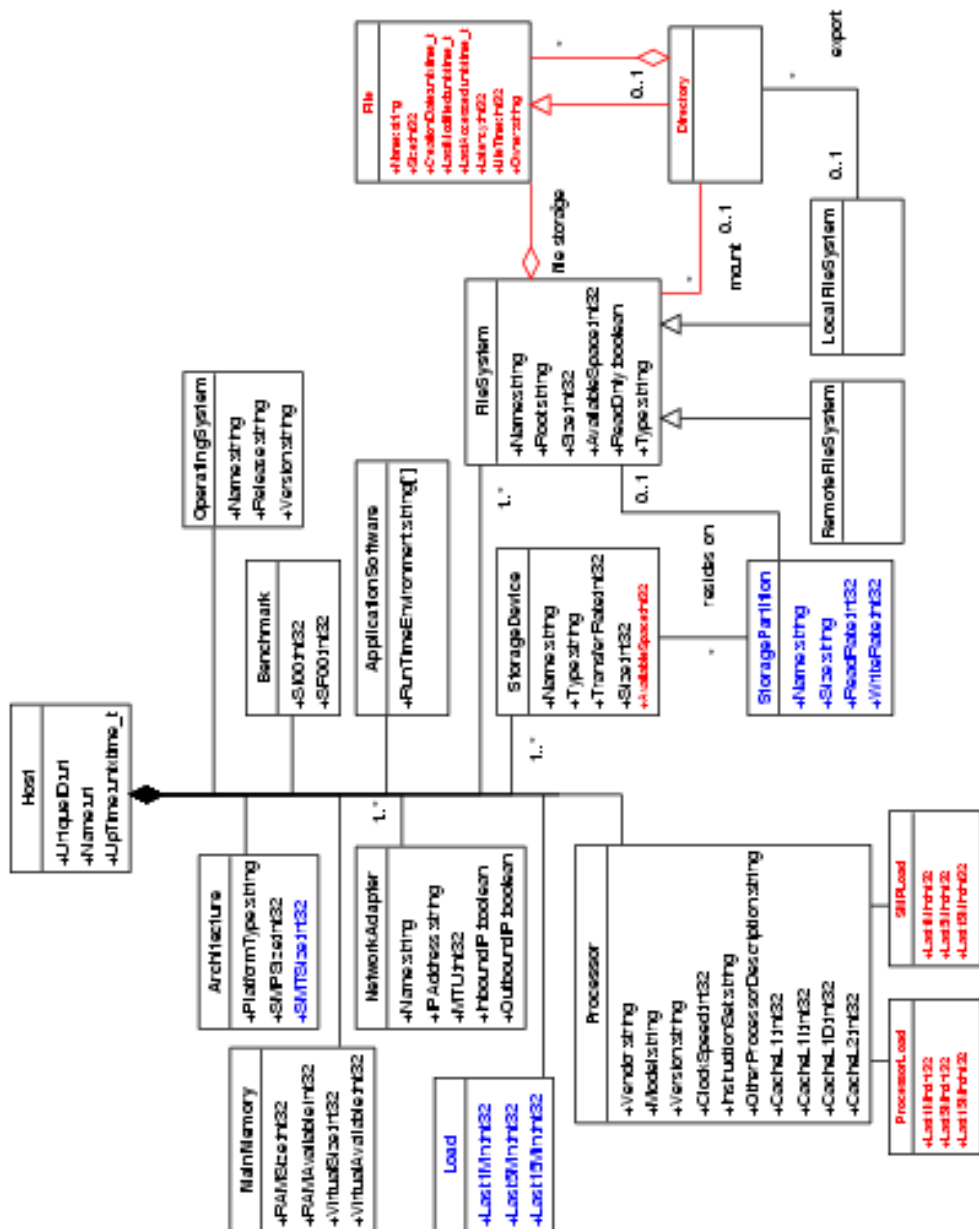


Figure 18: DIT for the computing resources

Created with Poseidon for UML Community Edition. Not for Commercial Use.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 19: DIT for the worker nodes

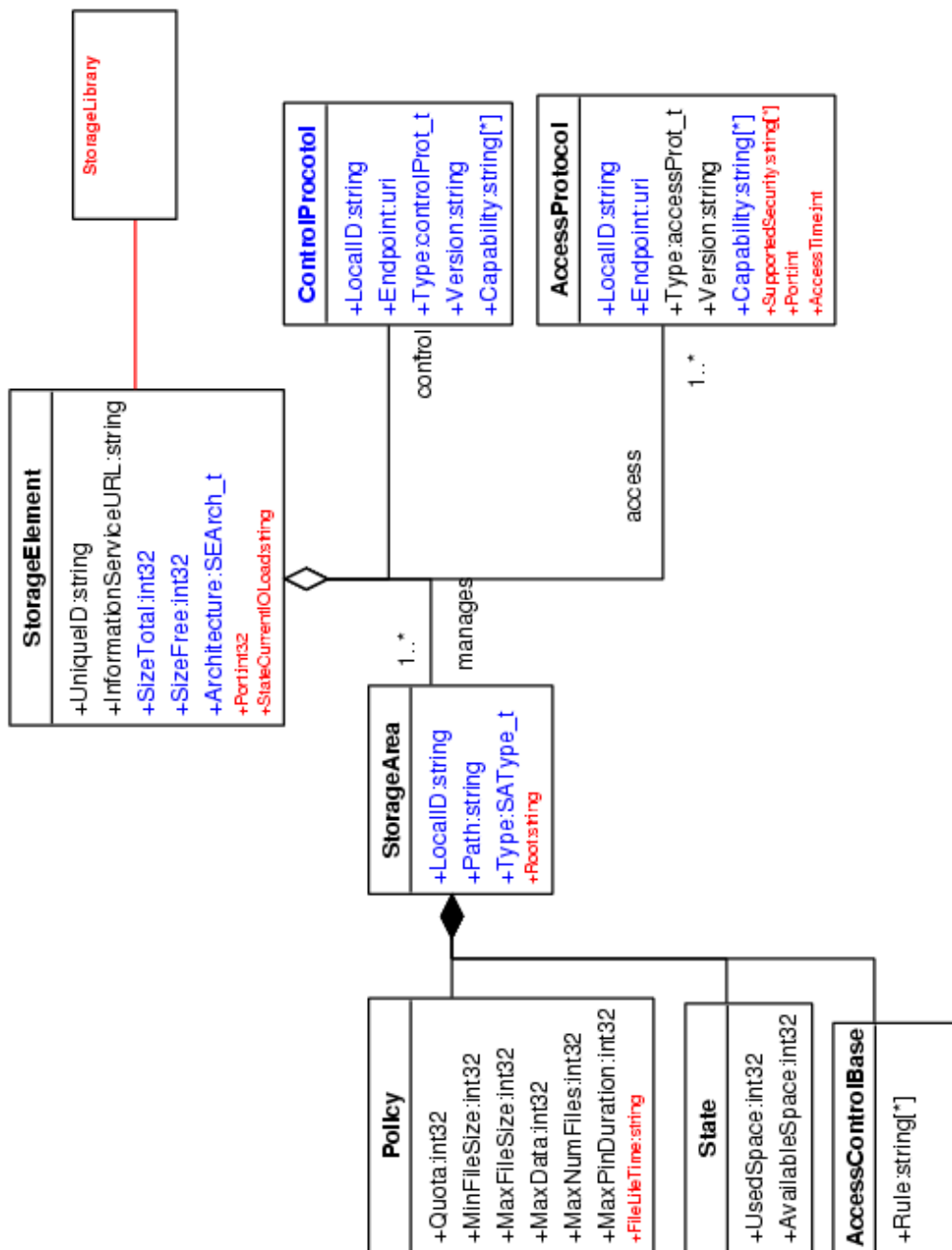
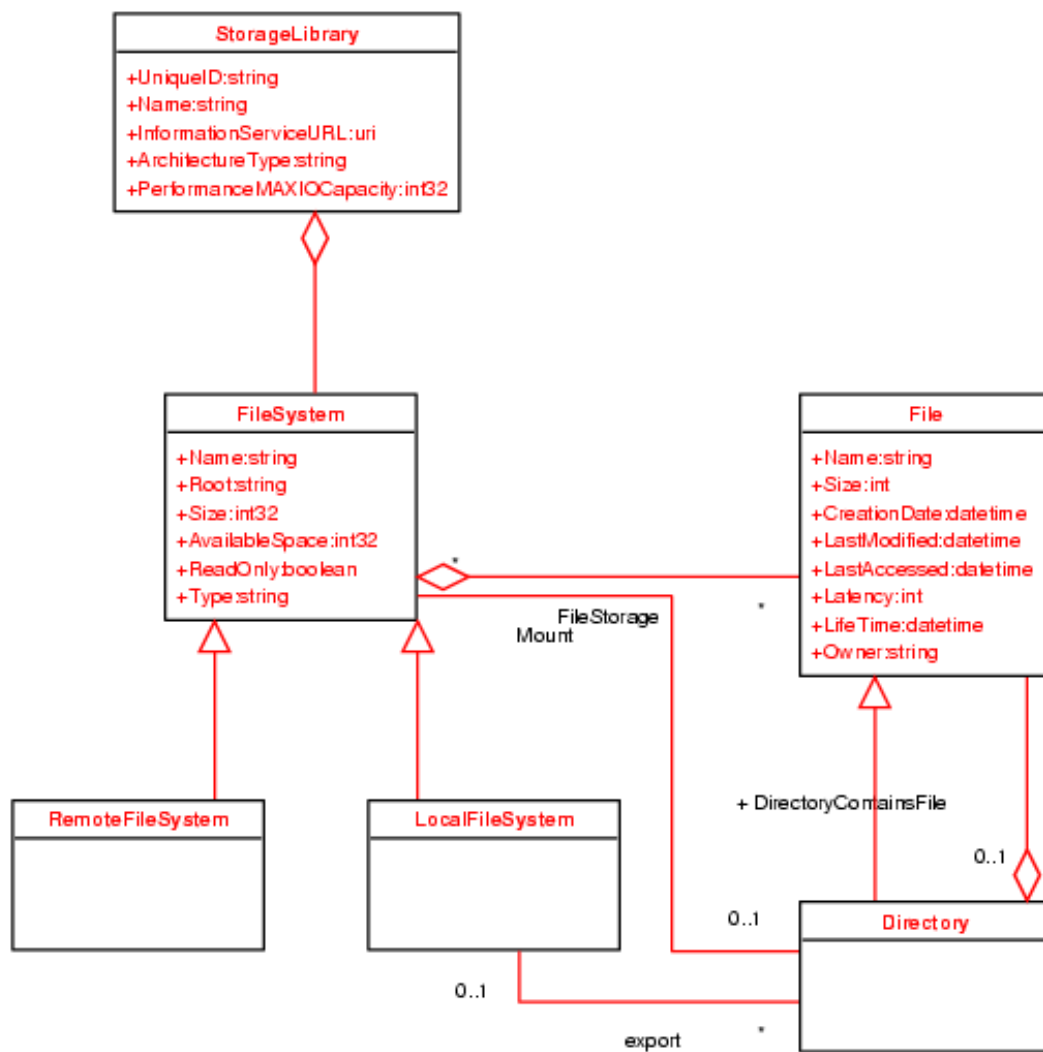


Figure 20: DIT for the storage resources

Created with Poseidon for UML Community Edition. Not for Commercial Use.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 21: DIT for the storage libraries