

WORLDWIDE LHC COMPUTING GRID

GLITE 3.1 USER GUIDE

MANUALS SERIES

Document identifier:	CERN-LCG-GDEIS-722398
EDMS id:	722398
Version:	1.2
Date:	December 18, 2009
Section:	Experiment Integration and Distributed Analysis
Document status:	DRAFT
Author(s):	Stephen Burke, Simone Campana, Elisa Lanciotti, Patricia Méndez Lorenzo, Vincenzo Miccio, Christopher Nater, Roberto Santinelli, Andrea Sciabà
Editor:	Andrea Sciabà
File:	gLite-3-UserGuide

Abstract: This guide is an introduction to the WLCG/EGEE Grid and to the gLite 3.1 middleware from a user's point of view.

COPYRIGHT NOTICE

Copyright © Members of the EGEE-II Collaboration, 2006.

See www.eu-egee.org for details on the copyright holders.

EGEE-II (Enabling Grids for E-science-II) is a project co-funded by the European Commission as an Integrated Infrastructure Initiative within the 6th Framework Programme. EGEE-II began in April 2006 and will run for 2 years.

For more information on EGEE-II, its partners and contributors please see www.eu-egee.org. You are permitted to copy and distribute, for non-profit purposes, verbatim copies of this document containing this copyright notice. This includes the right to copy this document in whole or in part, but without modification, into other documents if you attach the following reference to the copied elements: "Copyright © Members of the EGEE-II Collaboration 2006. See www.eu-egee.org for details".

Using this document in a way and/or for purposes not foreseen in the paragraph above, requires the prior written permission of the copyright holders.

The information contained in this document represents the views of the copyright holders as of the date such views are published.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE MEMBERS OF THE EGEE-II COLLABORATION, INCLUDING THE COPYRIGHT HOLDERS, OR THE EUROPEAN COMMISSION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks: EGEE and gLite are registered trademarks held by CERN on behalf of the EGEE collaboration. All rights reserved.

Document Change Record

Issue	Item	Reason for Change
15/01/10	V1.3	Improved version for gLite 3.1
15/01/08	V1.2	First version for gLite 3.1
17/01/07	v1.1	Revised version
20/04/06	v1.0	First draft

Files

Software Products	User files
PDF	https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.pdf
PS	https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.ps
HTML	https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html

CONTENTS

1	INTRODUCTION.....	10
1.1	ACKNOWLEDGMENTS	10
1.2	OBJECTIVES OF THIS DOCUMENT	10
1.3	APPLICATION AREA.....	10
1.4	DOCUMENT EVOLUTION PROCEDURE	10
1.5	REFERENCE AND APPLICABLE DOCUMENTS	10
1.6	TERMINOLOGY	14
1.6.1	Glossary	15
2	EXECUTIVE SUMMARY.....	18
3	OVERVIEW	19
3.1	PRELIMINARY MATTERS.....	20
3.1.1	Code Development.....	20
3.1.2	Troubleshooting	20
3.1.3	User and VO utilities	20
3.2	THE WLCG/EGEE INFRASTRUCTURE.....	21
3.3	THE WLCG/EGEE ARCHITECTURE.....	21
3.3.1	Security.....	21
3.3.2	User Interface	22
3.3.3	Computing Element	22
3.3.4	Storage Element	23
3.3.5	Information Service	24
3.3.6	Data Management.....	28
3.3.7	Workload Management	29

3.4	JOB FLOW	30
3.4.1	Job Submission	30
3.4.2	Other Operations	31
4	GRID SECURITY AND GETTING STARTED	32
4.1	BASIC SECURITY CONCEPTS	32
4.1.1	Private and Public Keys	32
4.1.2	Encryption	32
4.1.3	Signing	32
4.1.4	Certificates	33
4.1.5	Certification Authorities	33
4.1.6	Proxies	33
4.1.7	VOMS Proxies	34
4.2	FIRST STEPS	34
4.3	OBTAINING A CERTIFICATE	35
4.3.1	X.509 Certificates	35
4.3.2	Requesting the Certificate	35
4.3.3	Getting the Certificate	35
4.3.4	Renewing the Certificate	36
4.3.5	Taking Care of Private Keys	37
4.4	REGISTERING WITH WLCG/EGEE	37
4.4.1	The Registration Service	37
4.4.2	Virtual Organisations	38
4.5	SETTING UP THE USER ACCOUNT	39
4.5.1	The User Interface	39
4.5.2	Checking a Certificate	39

4.6	PROXIES.....	42
4.6.1	Standard Proxies.....	42
4.6.2	VOMS Proxies	44
4.6.3	Proxy Renewal	46
5	INFORMATION SERVICE.....	49
5.1	THE MDS	49
5.1.1	lcg-info.....	49
5.1.2	lcg-infosites	52
5.1.3	The Local GRIS	54
5.1.4	Using the ldapsearch command to read the MDS.....	55
5.1.5	The Site BDII	57
5.1.6	The top-level BDII	59
5.2	SERVICEDISCOVERY	61
5.2.1	Running a Service Discovery query	62
5.3	MONITORING.....	63
6	WORKLOAD MANAGEMENT	64
6.1	INTRODUCTION	64
6.2	THE JOB DESCRIPTION LANGUAGE	64
6.3	THE COMMAND LINE INTERFACE	71
6.3.1	Single Job Submission.....	72
6.3.2	Job Operations.....	76
6.3.3	Advanced Sandbox Management.....	81
6.3.4	Real Time Output Retrieval	83
6.3.5	The BrokerInfo	84
6.3.6	Direct Submission to CREAM CE	85

6.4	ADVANCED JOB TYPES	87
6.4.1	Job Collections	87
6.4.2	DAG jobs	90
6.4.3	Parametric jobs	90
6.4.4	MPI Jobs	91
6.5	COMMAND LINE INTERFACE CONFIGURATION	91
6.5.1	WMProxy Configuration.....	91
7	DATA MANAGEMENT.....	94
7.1	INTRODUCTION	94
7.2	STORAGE ELEMENTS.....	94
7.2.1	Data Channel Protocols.....	94
7.2.2	Types of Storage Elements	94
7.2.3	The Storage Resource Manager interface.....	95
7.3	FILE NAMES IN GLITE 3.1	95
7.4	FILE CATALOGUE IN GLITE 3.1	96
7.4.1	LFC Commands	97
7.4.2	Access Control Lists	100
7.5	FILE AND REPLICA MANAGEMENT CLIENT TOOLS	102
7.5.1	LCG Data Management Client Tools	102
7.6	FILE TRANSFER SERVICE.....	109
7.6.1	Basic Concepts	109
7.6.2	Transfer job states.....	110
7.6.3	Individual file states	110
7.6.4	FTS Commands	111

7.7	LOW LEVEL DATA MANAGEMENT TOOLS.....	114
7.7.1	GSIFTP.....	114
7.7.2	CASTOR and RFIO.....	115
7.7.3	dCache and DCAP.....	116
7.8	JOB SERVICES AND DATA MANAGEMENT.....	116
7.9	THE AMGA METADATA CATALOG.....	119
7.9.1	Introduction.....	119
7.9.2	Configuration of the client.....	119
7.9.3	Metadata access from the shell.....	120
7.9.4	Some commands to manipulate entries and attributes of a table.....	121
7.9.5	Using the API's.....	123
A	THE GRID MIDDLEWARE.....	125
B	ENVIRONMENT VARIABLES AND CONFIGURATION FILES.....	126
C	JOB STATUS DEFINITION.....	128
D	VO-WIDE UTILITIES.....	130
D.1	INTRODUCTION.....	130
D.2	FREEDOM OF CHOICE FOR RESOURCES.....	130
D.3	SERVICE AVAILABILITY MONITORING.....	130
D.4	THE VO BOX.....	130
D.5	VO SOFTWARE INSTALLATION.....	131
D.6	USING LCG-TAGS.....	131
D.7	USING LCG-MANAGEVOTAG.....	132
E	DATA MANAGEMENT AND FILE ACCESS THROUGH AN APPLICATION PRO- GRAMMING INTERFACE.....	133

F	THE GLUE SCHEMA.....	145
F.1	BASIC CONCEPTS	145
F.2	MAPPINGS.....	145
F.3	INFORMATION PROVIDERS.....	146
F.4	GLUE ATTRIBUTES	146
F.4.1	Site information	147
F.4.2	Service information	147
F.4.3	Attributes for the Computing Element	148
F.4.4	Attributes for the Storage Element	151
F.4.5	Attributes for the CE-SE Binding	154

1. INTRODUCTION

1.1. ACKNOWLEDGMENTS

This work received support from the following institutions:

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

1.2. OBJECTIVES OF THIS DOCUMENT

This document gives an overview of the gLite 3.1 middleware. It helps users to understand the building blocks of the Grid and the available interfaces to the Grid services in order to run jobs and manage data.

This document is neither an administration nor a developer guide.

1.3. APPLICATION AREA

This guide is addressed to WLCG/EGEE users and site administrators who would like to work with the gLite 3.1 middleware.

1.4. DOCUMENT EVOLUTION PROCEDURE

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

1.5. REFERENCE AND APPLICABLE DOCUMENTS

REFERENCES

- [1] Glossaries of Grid terms
<http://www.gridpp.ac.uk/gas/>
<http://egee-jra2.web.cern.ch/EGEE-JRA2/Glossary/Glossary.html>
<http://grid-it.cnaf.infn.it/fileadmin/users/dictionary/dictionary.html>

- [2] EGEE – Enabling Grids for E-science
<http://eu-egee.org/>
- [3] gLite – Lightweight Middleware for Grid Computing
<http://cern.ch/glite/>
- [4] Worldwide LHC Computing Grid
<http://cern.ch/LCG/>
- [5] The DataGrid Project
<http://www.edg.org/>
- [6] DataTAG – Research & technological development for a Data TransAtlantic Grid
<http://cern.ch/datatag/>
- [7] The Globus Alliance
<http://www.globus.org/>
- [8] GriPhyN – Grid Physics Network
<http://www.griphyn.org/>
- [9] iVDgL – International Virtual Data Grid Laboratory
<http://www.ivdgl.org/>
- [10] Open Science Grid
<http://www.opensciencegrid.org/>
- [11] The Virtual Data Toolkit
<http://vdt.cs.wisc.edu/>
- [12] NorduGrid
<http://www.nordugrid.org/>
- [13] Ian Foster, Carl Kesselman, Steven Tuecke,
The Anatomy of the Grid: Enabling Scalable Virtual Organizations
<http://www.globus.org/alliance/publications/papers/anatomy.pdf>
- [14] M. Dimou,
LCG User Registration and VO Management
<https://edms.cern.ch/document/428034/>
- [15] EGEE CIC Operations Portal
<http://cic.in2p3.fr/>
- [16] Global Grid User Support
<http://www.ggus.org/>
- [17] GOC Database 3.0
<https://goc.gridops.org/>

- [18] GOC Monitoring links
<http://goc.grid-support.ac.uk/gridsite/monitoring/>
Google map
<http://goc02.grid-support.ac.uk/googlemaps/sam.html>
- [19] Overview of the Grid Security Infrastructure
<http://www-unix.globus.org/security/overview.html>
- [20] The Storage Resource Manager
<http://sdm.lbl.gov/srm-wg/>
- [21] The GLUE schema
<http://glueschema.forge.cnaf.infn.it/>
- [22] The GLUE LDAP schema
http://forge.cnaf.infn.it/plugins/scmsvn/viewcvs.php/v_1_2/mapping/ldap/schema/openldap-2-1/?root=glueschema
- [23] MDS 2.2 Features in the Globus Toolkit 2.2 Release
http://www.globus.org/toolkit/mds/#mds_gt2
- [24] R-GMA: Relational Grid Monitoring Architecture
<http://www.r-gma.org/index.html>
- [25] B. Tierney *et al.*,
A Grid Monitoring Architecture,
GGF, 2001 (revised 2002)
<http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf>
- [26] S. Campana, M. Litmaath, A. Sciabà,
LCG-2 Middleware overview
<https://edms.cern.ch/document/498079/>
- [27] F. Pacini,
EGEE User's Guide – WMS Service
<https://edms.cern.ch/document/572489/>
- [28] F. Pacini,
EGEE User's Guide – WMPProxy service
<https://edms.cern.ch/document/674643/>
- [29] WP1 Workload Management Software – Administrator and User Guide
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf
- [30] CESNET,
EGEE User's Guide – Service Logging and Bookkeeping (L&B)
<https://edms.cern.ch/document/571273/>
- [31] Using LXPLUS as a gLite User Interface
<https://twiki.cern.ch/twiki/bin/view/LCG/AfsUiUserSetup> <https://twiki.cern.ch/twiki/bin/view/LCG/AfsUiUserSetup>

- [32] GridICE: a monitoring service for the Grid
<http://gridice.forge.cnaf.infn.it/>
- [33] Condor Classified Advertisements
<http://www.cs.wisc.edu/condor/classad>
- [34] The Condor Project
<http://www.cs.wisc.edu/condor/>
- [35] F. Pacini,
Job Description Language HowTo
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf
- [36] F. Pacini,
JDL Attributes
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf
- [37] F. Pacini,
Job Description Language Attributes Specification for the gLite middleware (submission through Network Server)
<https://edms.cern.ch/document/555796/1/>
- [38] F. Pacini,
Job Description Language Attributes Specification for the gLite middleware (submission through WMPProxy service)
<https://edms.cern.ch/document/590869/1/>
- [39] The EDG-Brokerinfo User Guide
<http://www.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2.2.pdf>
- [40] MPI Wiki page
<http://grid.ie/mpi/wiki/>
- [41] GSIFTP Tools for the Data Grid
<http://www.globus.org/toolkit/docs/2.4/datagrid/deliverables/gsiftp-tools.html>
- [42] RFIO: Remote File Input/Output
<http://doc.in2p3.fr/doc/public/products/rfio/rfio.html>
- [43] CASTOR
<http://cern.ch/castor/>
- [44] dCache
<http://www.dCache.org/>
- [45] Scientific Linux
<http://www.scientificlinux.org/>
- [46] User level tools documentation Wiki
http://goc.grid.sinica.edu.tw/gocwiki/User_tools

- [47] R. Santinelli, F. Donno,
Experiment Software Installation
<https://edms.cern.ch/document/498080/>
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/internal/chep04/SW_Installation.pdf
- [48] GOC Wiki
<http://goc.grid.sinica.edu.tw/gocwiki/FrontPage>
- [49] Freedom of Choice for Resources
<https://lcg-fcr.cern.ch:8443/fcr/fcr.cgi>
- [50] Service Availability Monitoring
http://goc.grid.sinica.edu.tw/gocwiki/Service_Availability_Monitoring
- [51] Service Availability Monitoring Web Interface
<https://lcg-sam.cern.ch:8443/sam/sam.py>
- [52] VO box How-to
http://goc.grid.sinica.edu.tw/gocwiki/VO-box_HowTo
- [53] EGEE User's Guide – Service Discovery
<http://hepunix.rl.ac.uk/egee/jra1-uk/sd/service-discovery.pdf>
- [54] Storage Classes in WLCG
<http://glueschema.forge.cnaf.infn.it/uploads/Spec/V13/SE-Model-3.5.pdf>
- [55] LCG-2 User Guide
<https://edms.cern.ch/document/454439/>
- [56] CREAM Home page
<http://grid.pd.infn.it/cream>
- [57] CREAM User's Guide
<https://edms.cern.ch/document/595770>
- [58] Specification of the JDL attributes supported by the CREAM CE service
<https://edms.cern.ch/document/592336>
- [59] AMGA Manual
http://amga.web.cern.ch/amga/downloads/amga-manual_1_30.pdf

1.6. TERMINOLOGY

The Grid world has a lot of specialised jargon. Acronyms used in this document are explained below; for more acronyms and definitions see [1].

1.6.1. Glossary

AFS:	Andrew File System
API:	Application Programming Interface
BDII:	Berkeley Database Information Index
CASTOR	CERN Advanced STORage manager
CE:	Computing Element
CERN:	European Laboratory for Particle Physics
ClassAd:	Classified advertisement (Condor)
CLI:	Command Line Interface
CNAF:	INFN's National Center for Telematics and Informatics
dcap:	dCache Access Protocol
DIT:	Directory Information Tree (LDAP)
DLI:	Data Location Interface
DN:	Distinguished Name
EDG:	European DataGrid
EDT:	European DataTAG
EGEE:	Enabling Grids for E-science
ESM:	Experiment Software Manager
FCR:	Freedom of Choice for Resources
FNAL:	Fermi National Accelerator Laboratory
FTS:	File Transfer Service
GFAL:	Grid File Access Library
GG:	Grid Gate (aka gatekeeper)
GGF:	Global Grid Forum (now called OGF)
GGUS:	Global Grid User Support
GIIS:	Grid Index Information Server
GLUE:	Grid Laboratory for a Uniform Environment
GMA:	Grid Monitoring Architecture
GOC:	Grid Operations Centre
GRAM:	Grid Resource Allocation Manager
GRIS:	Grid Resource Information Service
GSI:	Grid Security Infrastructure
gsidcap:	GSI-enabled version of the dCache Access Protocol
gsirfio:	GSI-enabled version of the Remote File Input/Output protocol
GUI:	Graphical User Interface
GUID:	Grid Unique ID
HSM:	Hierarchical Storage Manager
ID:	Identifier
INFN:	Istituto Nazionale di Fisica Nucleare
IS:	Information Service
JDL:	Job Description Language
kdcap:	Kerberos-enabled version of the dCache Access Protocol
LAN:	Local Area Network
LB:	Logging and Bookkeeping Service

LDAP:	Lightweight Directory Access Protocol
LFC:	LCG File Catalogue
LFN:	Logical File Name
LHC:	Large Hadron Collider
LCG:	LHC Computing Grid
LRC:	Local Replica Catalogue
LRMS:	Local Resource Management System
LSF:	Load Sharing Facility
MDS:	Monitoring and Discovery Service
MPI:	Message Passing Interface
MSS:	Mass Storage System
NS:	Network Server
OGF:	Open Grid Forum (formerly called GGF)
OS:	Operating System
PBS:	Portable Batch System
PFN:	Physical File name
PID:	Process IDentifier
POOL:	Pool of Persistent Objects for LHC
PPS:	Pre-Production Service
RAL:	Rutherford Appleton Laboratory
RB:	Resource Broker
RFIO:	Remote File Input/Output
R-GMA:	Relational Grid Monitoring Archicecture
RLI:	Replica Location Index
RLS:	Replica Location Service
RM:	Replica Manager
RMC:	Replica Metadata Catalogue
RMS:	Replica Management System
ROC:	Regional Operations Centre
ROS:	Replica Optimization Service
SAM:	Service Availability Monitoring
SASL:	Simple Authorization & Security Layer (LDAP)
SE:	Storage Element
SFN:	Site File Name
SMP:	Symmetric Multi Processor
SN:	Subject Name
SRM:	Storage Resource Manager
SURL:	Storage URL
TURL:	Transport URL
UI:	User Interface
URI:	Uniform Resource Identifier
URL:	Uniform Resource Locator
UUID:	Universal Unique ID
VDT:	Virtual Data Toolkit
VO:	Virtual Organization
WLCCG:	Worldwide LHC Computing Grid

WMS: Workload Management System
WN: Worker Node
WPr: Work Package #n

2. EXECUTIVE SUMMARY

This user guide is intended for users of the gLite 3.1 middleware. In these pages, the user will find an introduction to the services provided by the WLCG/EGEE Grid and a description of how to use them. Examples are given of the management of jobs and data, the retrieval of information about resources, and other functionality.

An introduction to the gLite 3.1 middleware is presented in Chapter 3. This chapter describes all the middleware components and provides most of the necessary terminology. It also presents the WLCG and the EGEE projects, which developed the gLite 3.1 middleware.

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

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

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

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

Finally, the appendices give information about the gLite 3.1 middleware components (Appendix A), the configuration files and environment variables for users (Appendix B), the possible states of a job during submission and execution (Appendix C), user tools for the Grid (Appendix ??), VO-wide utilities (Appendix D), APIs for data management and file access (Appendix E), and the GLUE Schema used to describe Grid resources (Appendix F).

3. OVERVIEW

The *EGEE project* [2] has a main goal of providing researchers with access to a geographically distributed computing Grid infrastructure, available 24 hours a day. It focuses on maintaining and developing the *gLite* middleware [3] and on operating a large computing infrastructure for the benefit of a vast and diverse research community.

The *Worldwide LHC Computing Grid Project (WLCG)* [4] was created to prepare the computing infrastructure for the simulation, processing and analysis of the data of the *Large Hadron Collider (LHC)* experiments. The LHC is the world's largest and most powerful particle accelerator.

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

The gLite 3.1 middleware comes from a number of current and past Grid projects, like DataGrid [5], DataTag [6], Globus [7], GriPhyN [8], iVDGL [9], EGEE and WLCG. This middleware is currently installed in sites participating in WLCG/EGEE.

In WLCG other Grid infrastructures exist, namely the *Open Science Grid (OSG)* [10], which uses the middleware distributed by VDT [11], and NorduGrid [12], which uses the ARC middleware. These are not covered by this guide.

The case of the LHC experiments illustrates well the motivation behind Grid technology. The LHC accelerator started operations in 2009 and the experiments that use it (*ALICE, ATLAS, CMS* and *LHCb*) will generate enormous amounts of data. The processing of this data requires huge computational and storage resources, and the associated human resources for operation and support. It was not considered feasible to concentrate all the resources at one site, and therefore it was agreed that the WLCG computing service would be implemented as a geographically distributed *Computational Data Grid*. This means that the service uses computing and storage resources installed at a large number of computing sites in many different countries, interconnected by fast networks. The gLite middleware hides much of the complexity of this environment from the user, giving the impression that all of these resources are available in a coherent virtual computer centre.

The users of a Grid infrastructure are divided into *Virtual Organisations (VOs)* [13], abstract entities grouping users, institutions and resources in the same administrative domain [14].

The WLCG/EGEE VOs correspond to real organisations or projects, such as the four LHC experiments, the community of biomedical researchers, etc. An updated list of all the EGEE VOs can be found at the CIC portal [15].

3.1. PRELIMINARY MATTERS

3.1.1. Code Development

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

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

3.1.2. Troubleshooting

This document will also explain the meaning of the most common error messages and give some advice on how to avoid some common errors. This guide cannot, however, include all the possible failures a user may encounter while using gLite 3.1. These errors may be produced due to user mistakes, to misconfiguration of the Grid components, to hardware or network failures, or even to bugs in the gLite middleware.

Subsequent sections of this guide provide references to documents which go into greater detail about the gLite 3.1 components.

The *Global Grid User Support (GGUS)* [16] service provides centralised support for WLCG/EGEE users, by answering questions, tracking known problems, maintaining lists of frequently asked questions, providing links to documentation, etc. The GGUS portal is the key entry point for Grid users looking for help.

Finally, a user who thinks that there is a security risk in the Grid may directly contact the relevant site administrator if the situation is urgent, as this may be faster than going through GGUS. Information on the local site contacts can be obtained from the Information Service or from the GOC database [17], which is described in Chapter 4.

3.1.3. User and VO utilities

This guide mainly covers information useful for the average user. Thus, only core gLite 3.1 middleware is described. Nevertheless, there are several tools which are not part of the middleware, but may be very useful to users. Some of these tools are summarised in Appendix ??.

Likewise, there are utilities that are only available to certain (authorised) users of the Grid. An example is the administration of the resources viewed by a VO or the installation of VO software on WLCG/EGEE nodes. Only authorised users can install software on the computing resources of WLCG/EGEE: the installed software is also published in the Information Service, so that users can select sites where the software they need is installed. Information on such topics is given in Appendix D.

3.2. THE WLCG/EGEE INFRASTRUCTURE

WLCG/EGEE operates a production Grid distributed over more than 200 sites around the world, with more than 30,000 CPUs and 20 PB of data storage. The status of the Grid can be seen from the various monitoring pages linked from the Grid Operations Centre (GOC) monitoring page [18]; in particular look at the Google map for a quick overview. Sites can choose which VOs to support and at what level, so users will generally not have access to every site; later chapters describe how to find out which resources are available to a specific user.

Sites vary widely in the size of their computing and storage resources; for WLCG the largest sites are designated as Tier-1 and play a key role in storing and processing data. In EGEE, sites are organised into geographical regions, co-ordinated by a Regional Operations Centre (ROC).

WLCG/EGEE also runs a smaller Pre-Production Service (PPS), a separate Grid where new versions of the middleware can be tested by both sites and users before being deployed on the main production Grid.

3.3. THE WLCG/EGEE ARCHITECTURE

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

3.3.1. Security

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

Once the user registration is complete, he can access WLCG/EGEE. The *Grid Security Infrastructure (GSI)* in WLCG/EGEE enables secure authentication and communication over an open network [19]. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol, with extensions for single sign-on and delegation.

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

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

The authorisation of a user on a specific Grid resource can be done in two different ways. The first is simpler, and relies on the *grid-mapfile* mechanism. The Grid resource has a local grid-mapfile

which maps user certificates to local accounts. When a user's request for a service reaches a host, the *Subject Name* of the user (contained in the proxy) is checked against what is in the local grid-mapfile to find out to which local account (if any) the user certificate is mapped, and this account is then used to perform the requested operation [19]. The second way relies on the *Virtual Organisation Membership Service (VOMS)* and the *LCAS/LCMAPS* mechanism, which allow for a more detailed definition of user privileges, and will be explained in more detail later. More recently the *Site Central Authorization Service (SCAS) [?]* has been introduced to allow LCMAPS to retrieve authorization information from a central service rather than from a local configuration file. Finally, *ARGUS [?]* will soon be available as an alternative to SCAS.

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

3.3.2. User Interface

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

- list all the resources suitable to execute a given job;
- submit jobs for execution;
- cancel jobs;
- query the status of jobs and retrieve their output;
- copy, replicate and delete files from the Grid;
- submit and manage file transfer jobs
- retrieve the status of different resources from the Information System.

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

3.3.3. Computing Element

A *Computing Element (CE)*, in Grid terminology, is some set of computing resources localized at a site (i.e. a cluster, a computing farm). A CE includes a *Grid Gate (GG)*¹, which acts as a generic interface to

¹For Globus-based CEs, it is called *Gatekeeper*.

the cluster; a *Local Resource Management System (LRMS)* (sometimes called *batch system*), and the cluster itself, a collection of *Worker Nodes (WNs)*, the nodes where the jobs are run.

There are two GG implementations in gLite 3.1: the *LCG CE*, initially developed by EDG and maintained by WLCG, and the *CREAM CE*, developed by EGEE. Sites can choose what to install, and some of them provide both types. The GG is responsible for accepting jobs and dispatching them for execution on the WNs via the LRMS.

In gLite 3.1 the supported LRMS types are OpenPBS/PBSPro, LSF, Maui/Torque, BQS and Condor, with work underway to support Sun GridEngine.

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

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

```
CEId = <gg_hostname>:<port>/<gg_type>-<LRMS_type>-<batch_queue_name>
```

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

```
ce101.cern.ch:2119/jobmanager-lcglsf-grid_alice
cmsrv25.fnal.gov:2119/condor-condor-cms
gridce0.pi.infn.it:8443/cream-lsf-cms4
```

3.3.4. Storage Element

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

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

Most storage resources are managed by a *Storage Resource Manager (SRM)* [20], a middleware service providing capabilities like transparent file migration from disk to tape, file pinning, space reservation, etc. However, SRM implementations from different storage system may differ and offer different capabilities.

There are different storage systems used in WLCG/EGEE. The *Disk Pool Manager (DPM)* is used for SEs with disk-based storage only, while *CASTOR* is designed to manage large-scale MSS, with front-end disks and back-end tape storage. *dCache* is targeted at both MSS and disk array storage systems. Other SRM implementations are available, like StoRM and BestMAN.

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

Type	Resources	File transfer	File I/O	SRM
CASTOR	MSS	GSIFTP	insecure RFIO	Yes
dCache	Disks/MSS	GSIFTP	gsidcap	Yes
DPM	Disks	GSIFTP	secure RFIO	Yes
StoRM	Disks/MSS	GSIFTP	direct file access	Yes

3.3.5. Information Service

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

Much of the data published to the IS conforms to the *GLUE Schema* [21], which defines a common conceptual data model to be used for Grid resource monitoring and discovery. More details about the GLUE schema can be found in Appendix F.

Two IS systems are used in gLite 3.1: the *Globus Monitoring and Discovery Service (MDS)* [23], used for resource discovery and to publish the resource status, and the *Relational Grid Monitoring Architecture (R-GMA)* [24], used for accounting, monitoring and publication of user-level information.

MDS

The MDS implements the GLUE Schema using OpenLDAP, an open source implementation of the *Lightweight Directory Access Protocol (LDAP)*, a specialised database optimised for reading, browsing and searching information. Access to MDS data is insecure, both for reading (clients and users) and for writing (services publishing information), i.e. no Grid credentials are required.

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

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

Figure 1 schematically depicts the Directory Information Tree (DIT) of a site: the root entry identifies the site, and entries for site information, CEs and SEs and other services appear at lower levels. Appendix F describes the GLUE schema entries in more detail.

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

Figure 2 shows the architecture of the information system in WLCG/EGEE. Computing and storage resources at a site run a piece of software called an *Information Provider*, which generates the relevant information about the resource (both static, like the type of SE, and dynamic, like the used space in an SE). This information is published via a server called a *resource-level BDII*, which normally runs on the resource itself. The *site-level BDII* is used to store and publish data from all the resource-level BDIIs at a site.

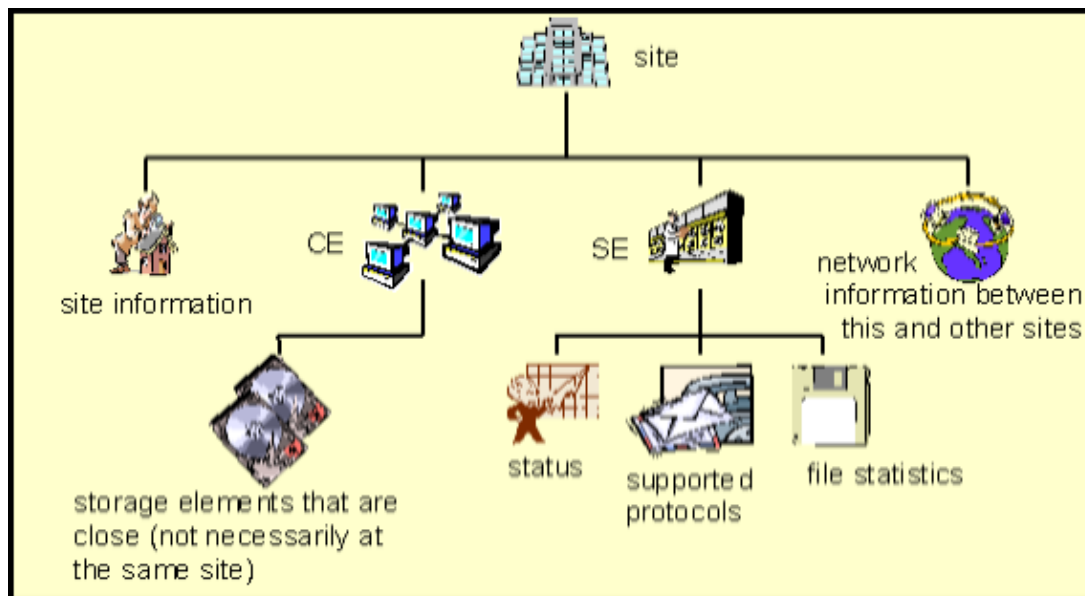


Figure 1: The Directory Information Tree (DIT)

Finally, a *top-level BDII* is used as the top of the hierarchy. BDIIs at this level are configured to read from a specific set of sites, which effectively defines a view of the overall Grid resources. These BDIIs act as a cache by storing information about the Grid status in their database. The BDIIs therefore contain all the available information about the Grid sites they look at. Nevertheless, it is always possible to get information about specific resources by directly contacting the site- or resource-level BDIIs..

The top-level BDIIs obtain information about the sites in the Grid from the Grid Operations Centre (GOCDB) database [17], where site managers can insert the contact address of their BDII as well as other useful information about the site.

R-GMA

R-GMA is an implementation of the *Grid Monitoring Architecture (GMA)* proposed by the *Global Grid Forum (GGF)* [25]. In R-GMA, information is in many ways presented as though it were in a global distributed relational database, although there are some differences (for example, a table may have multiple rows with the same primary key). This model is more powerful than the LDAP-based one, since relational databases support more advanced query operations. It is also much easier to modify the schema in R-GMA, making it more suitable for user information.

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

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

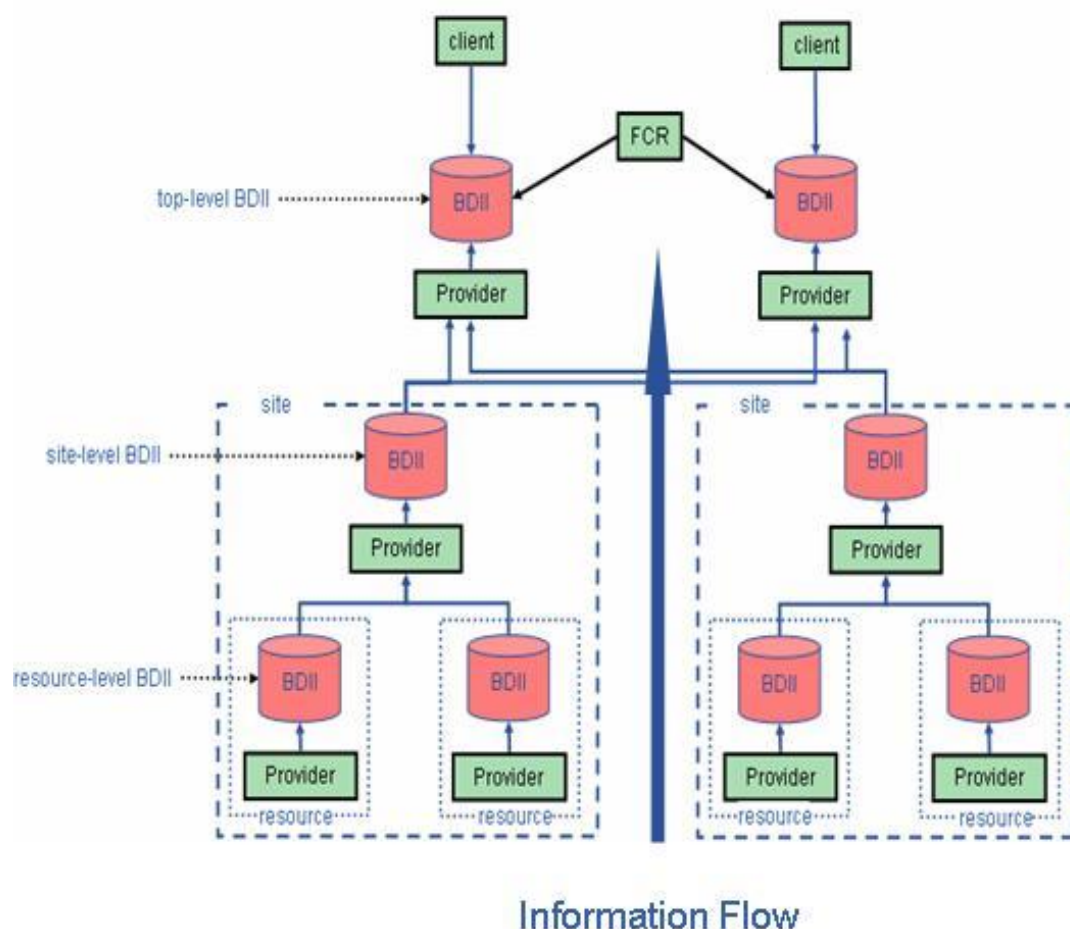


Figure 2: The Information Service in WLCG/EGEE.

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

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

R-GMA presents information as a single virtual database containing a set of virtual tables. The *Schema* contains the name and structure (column names, types and settings) of each virtual table in the system (Figure 4), and is normally co-located with the Registry. The Registry contains a list of Producers which publish information for each table. Producers may also register a *predicate*, i.e. a restriction on the tuples they produce, which enables more efficient selection of Producers which may be able to satisfy a query. A Consumer runs an SQL query on a table and the Registry selects the best Producers to answer the query through a process called *mediation*. The Consumer then contacts each Producer directly, combines the information and returns a set of tuples. The details

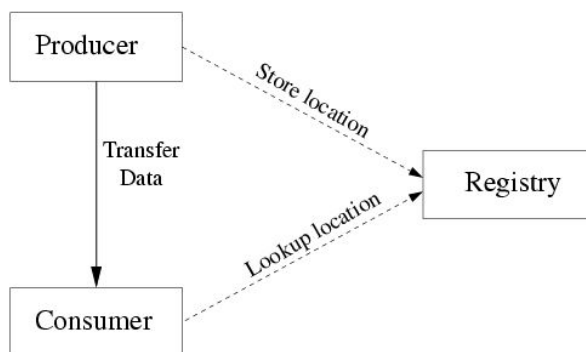


Figure 3: The R-GMA architecture.

of this process are hidden from the user, who just receives the tuples in response to a query.

An R-GMA system is defined by the Registry and the Schema: what information will be seen by a Consumer depends on what Producers are registered with the Registry. There is only one Registry and one Schema in the WLCG/EGEE production Grid (separate instances exist for the Pre-Production System).

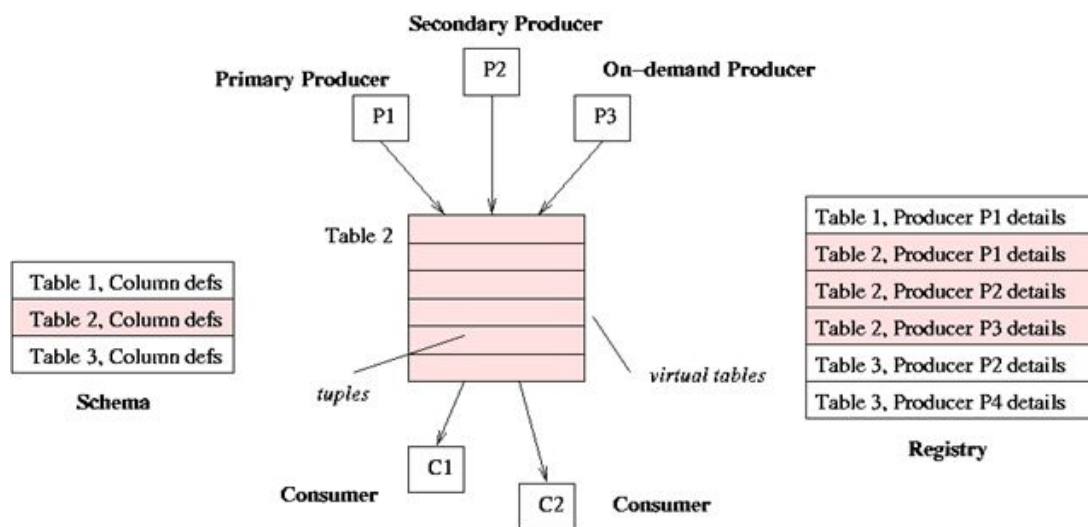


Figure 4: The virtual database of R-GMA

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

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

- **Continuous** (formerly known as stream): information is sent directly to Consumers as it is produced;
- **Latest**: only the latest information (the tuple with the most recent timestamp for a given value of the primary key) is sent to the Consumer;

- **History:** all tuples within a configurable retention period are stored to allow subsequent retrieval by Consumers.

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

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

3.3.6. Data Management

The primary unit for Grid data management, as in traditional computing, is the *file*. In a Grid environment, files can have *replicas* at many different sites. Because all replicas must be consistent, Grid files cannot be modified after creation, only read and deleted. Ideally, users do not need to know where a file is located, as they use logical names for the files that the Data Management services use to locate and access them.

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

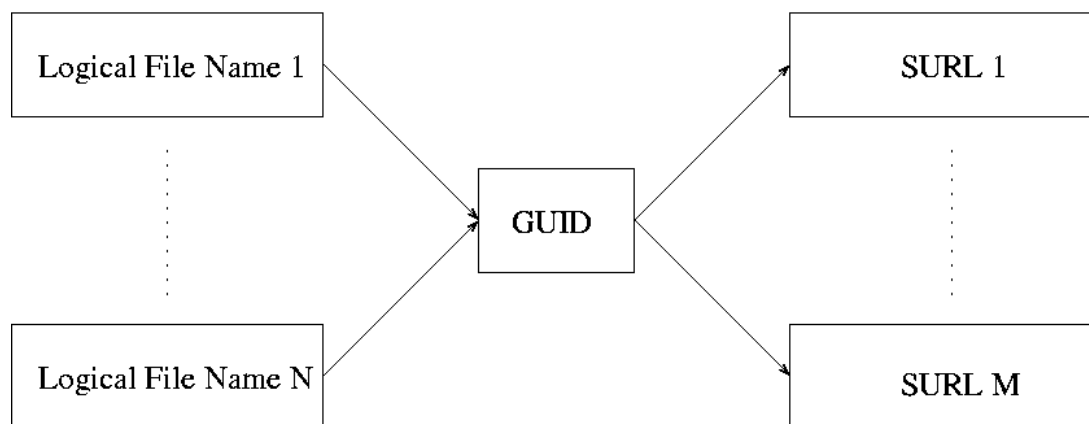


Figure 5: Different filenames in gLite 3.0.

A file can be unambiguously identified by its GUID; this is assigned the first time the file is registered in the Grid, and is based on the UUID standard to guarantee its uniqueness (UUIDs use a combination of a MAC address and a timestamp to ensure that all UUIDs are distinct). A GUID is of the form: `guid:<unique_string>` (e.g. `guid:93bd772a-b282-4332-a0c5-c79e99fc2e9c`).

In order to locate a file in the Grid, a user will normally use an LFN. LFNs are usually more intuitive, human-readable strings, since they are allocated by the user. Their form is: `lfn:<any_string>`, but the current WLCG/EGEE file catalogue uses strings which have the standard Unix hierarchical format, with elements separated by `/` characters. A Grid file can have many LFNs, in the same way that a file in a Unix file system can have many links.

The **SURL** provides information about the physical location of a file replica. Currently, **SURLs** have this format:

```
srm:<SE_hostname>/<path>
```

for files residing on an SRM-enabled SE.

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

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

The *File Transfer Service (FTS)* is the gLite service that takes care of accepting, scheduling and performing file transfers between SEs. It allows, among other things, to define partition file transfers among different VOs and set the priority of individual files. FTS is primarily used by the LHC VOs, as it is very convenient for large-scale massive data transfers.

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

3.3.7. Workload Management

The purpose of the *Workload Management System (WMS)* is to accept user jobs, to assign them to the most appropriate Computing Element, to record their status and retrieve their output [26] [27].

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

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

The glite WMS allows not only the submission of single jobs, but also collections of jobs (possibly with dependencies between them), making it much more efficient than the old LCG-2 Resource Broker.

Finally, the *Logging and Bookkeeping service (LB)* [30] tracks jobs managed by the WMS. It collects events from many WMS components and records the status and history of the job.

3.4. JOB FLOW

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

3.4.1. Job Submission

Figure 6 illustrates the process that takes place when a job is submitted to the Grid. The individual steps are as follows:

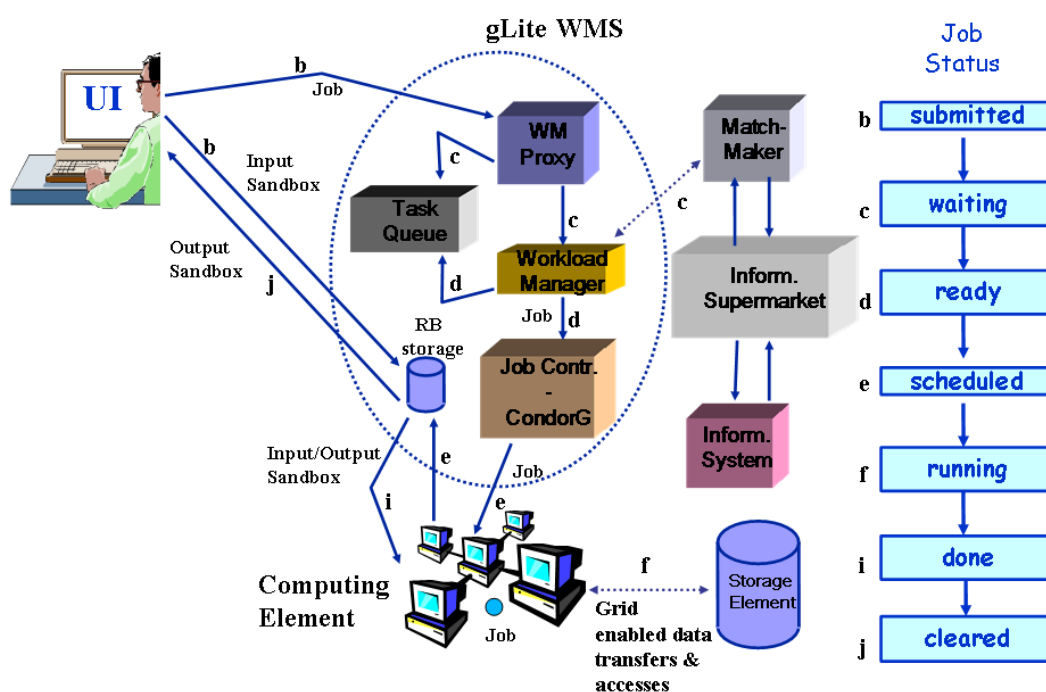


Figure 6: Job flow in the WLCG/EGEE Grid.

- After obtaining a digital certificate from a trusted Certification Authority, registering in a VO and obtaining an account on a User Interface, the user is ready to use the WLCG/EGEE Grid. He logs in to the UI and creates a proxy certificate to authenticate himself in subsequent secure interactions.
- The user submits a job from the UI to the gLite WMS. In the job description one or more files to be copied from the UI to the WN can be specified, and these are initially copied to the gLite WMS. This set of files is called the *Input Sandbox*. An event is logged in the LB and the status of the job is SUBMITTED.
- The WMS looks for the best available CE to execute the job. To do so, it interrogates the *Information Supermarket (ISM)*, an internal cache of information which in the current system is read from the BDII, to

- determine the status of computational and storage resources, and the File Catalogue to find the location of any required input files. Another event is logged in the LB and the status of the job is WAITING.
- d. The gLite WMS prepares the job for submission, creating a wrapper script that will be passed, together with other parameters, to the selected CE. An event is logged in the LB and the status of the job is READY.
 - e. The CE receives the request and sends the job for execution to the local LRMS. An event is logged in the LB and the status of the job is SCHEDULED.
 - f. The LRMS handles the execution of jobs on the local Worker Nodes. The Input Sandbox files are copied from the gLite WMS to an available WN where the job is executed. An event is logged in the LB and the status of the job is RUNNING.
 - g. While the job runs, Grid files can be directly accessed from a SE or after copying them to the local filesystem on the WN with the Data Management tools.
 - h. The job can produce new output files which can be uploaded to the Grid and made available for other Grid users to use. This can be achieved using the Data Management tools described later. Uploading a file to the Grid means copying it to a Storage Element and registering it in a file catalogue.
 - i. If the job ends without errors, the output (not large data files, but just small output files specified by the user in the so called *Output Sandbox*) is transferred back to the gLite WMS node. An event is logged in the LB and the status of the job is DONE.
 - j. At this point, the user can retrieve the output of his job to the UI. An event is logged in the LB and the status of the job is CLEARED.
 - k. Queries for the job status can be addressed to the LB from the UI. Also, from the UI it is possible to query the BDII for the status of the resources.
 - l. If the site to which the job is sent is unable to accept or run it, the job may be automatically resubmitted to another CE that satisfies the user requirements. After a maximum allowed number of resubmissions is reached, the job will be marked as aborted. Users can get information about the history of a job by querying the LB service.

3.4.2. Other Operations

While the Input and Output Sandboxes are a mechanism for transferring small data files needed to start a job or to check its results, large data files should be read and written from/to SEs and registered in a File Catalogue, and possibly replicated to other SEs. The LCG Data Management client tools are available for performing these tasks. In general, the user should not directly interact with the File Catalogue; instead, he should use the LCG tools.

The File Transfer Service (FTS) provides a managed way to move large numbers of files between SEs.

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

4. GRID SECURITY AND GETTING STARTED

This section gives an overview of the security aspects of the WLCG/EGEE Grid, and describes the preliminary steps to gain access to the Grid.

4.1. BASIC SECURITY CONCEPTS

Grid security is a very complex area, but it is useful for users to understand at least the basics of how the system works. The following sections give a brief explanation of the most important concepts.

4.1.1. Private and Public Keys

Grid security is based on the concept of *public key encryption*. Each user (or other entity like a server) has a *private key*, generated randomly. This is a number which can be used as a secret password to prove identity. The private key must therefore be kept totally secure; if someone can steal it they can impersonate the owner completely.

Each private key is mathematically related to another number called the *public key*. As the name suggests this can be known by anyone. Formally it is possible to calculate the private key from the public key, but in practice such a calculation is expected to take an unfeasibly long time (the time grows exponentially with the size of the keys). Conversely, calculating the public key from the private key is easy, hence these are sometimes referred to as *asymmetric keys*.

4.1.2. Encryption

The keys are used with an *encryption* algorithm, i.e. a mathematical function which can be applied to any data to produce a coded version of the data. The algorithm has the property that data encrypted using the private key can be decrypted with the public key, and vice versa. This mechanism can also be used to prove identity: if user X encodes some data with the public key of user Y, user Y decrypts it with his private key and sends the data back to user X, user X can be certain of Y's identity.

4.1.3. Signing

Private keys can also be used to *sign* a piece of data. This involves another mathematical function called a *hash function*, that is a function that, when applied to data of any length, produces a fixed-length number which is characteristic of the input data, like a digital fingerprint – even a tiny change to the input would produce a completely different hash.

To sign a piece of data a hash is calculated from it, and the hash is then encrypted with the private key and the result attached to the data. Anyone else can then decrypt the hash with the public key, and compare it with one they calculate themselves. If the two hashes match they know two things: that the data was signed by someone who had the private key corresponding to that public key, and that the data has not been modified since it was signed.

4.1.4. Certificates

To be useful, the public key has to be connected to some information about who the user (or server) is. This is stored in a specific format known as an *X.509 certificate* (X.509 being the name of the standard which specifies the format).

The most important thing in the certificate is the *Subject Name (SN)*, which is something which looks like:

```
/C=UK/O=eScience/OU=CLRC/L=RAL/CN=john smith
```

This is an example of a more general format called a *Distinguished Name (DN)*, which appears quite a lot in the Grid world. The idea is that a DN should uniquely identify the thing it names. The details of how to construct a DN have never been established as an international standard, but at least within the Grid it can be assumed that a DN is a unique name, and the SN in a certificate is the owner's name as far as the Grid is concerned.

A certificate also contains some other information, in particular an expiry date after which the certificate is no longer valid. User certificates are normally issued with an expiry date one year ahead, and have to be renewed before they expire. A renewed certificate will normally have new public and private keys, but will usually keep the same SN. In some circumstances, e.g. if the private key is stolen, a certificate may be *revoked*, i.e. added to a known list of certificates which should be considered invalid.

4.1.5. Certification Authorities

Certificates are issued by a *Certification Authority (CA)*. There are many commercial CAs, like Verisign, but for Grid usage there are special CAs run by academic organisations, generally serving users in a particular geographic region. The CA follows some defined procedures to make sure that it knows who their users are and that they are entitled to have a certificate.

To allow people to verify the information in the certificate, the CA signs it with its own private key. Anyone who wants to check the validity of a certificate needs to know the public key of the CA, and the CA therefore has a certificate of its own. Potentially this could create an infinite regression, but this is prevented by the fact that CA certificates, known as *root certificates*, are self-signed, i.e. the CA signs its own certificate. These root certificates are then distributed in some secure way, which in the Grid is typically as Linux RPMs from a trusted repository. (The root certificates of many commercial CAs are often pre-installed in web browsers.)

4.1.6. Proxies

To interact directly with a remote service a certificate can be used to prove identity. However, in the Grid world it is often necessary for a remote service to act on a user's behalf, e.g. a job running on a remote site needs to be able to talk to other servers to transfer files, and it therefore needs to prove that it is entitled to use the user's identity (this is known as *delegation*). On the other hand, since the private key is so vital it should not be sent to remote machines which might be insecure.

The solution is the use of a *proxy certificate*. A proxy certificate consists of a new public/private key pair, signed with the user's personal certificate with a SN like:

```
/C=UK/O=eScience/OU=CLRC/L=RAL/CN=john smith/CN=proxy
```

Proxies have normally a short lifetime (12 hours by default). A proxy files contain the proxy certificate, its private key and the user certificate (but not the user private key).

When verifying the user identity with a proxy, the mechanism is the same as for a personal certificate, but for the additional step of checking the proxy signature with the user certificate (included in the proxy file).

In security terms a proxy is a compromise. Since the private key is sent with it and is not protected by a passphrase, anyone who steals it can impersonate the owner, so proxies need to be treated carefully. Also there is no mechanism for revoking proxies, so in general even if someone knows that one has been stolen there is little they can do to stop it being used, apart from revoking the user certificate itself. On the other hand, proxies usually have a lifetime of only a few hours so the potential damage is fairly limited.

4.1.7. VOMS Proxies

A system called *VOMS (VO Management Service)* is used in WLCG/EGEE to manage information about the roles and privileges of users within a VO. This information is presented to services via an extension to the proxy. At proxy creation, one or more VOMS servers are contacted, and they return a “mini certificate” known as an *Attribute Certificate (AC)* which is signed by the VO and contains information about group membership and any associated roles within the VO.

To create a VOMS proxy the ACs are embedded in a standard proxy, and the whole thing is signed with the private key of the parent certificate. Services can then decode the VOMS information and use it as required, e.g. a user may only be allowed to do something if he has a particular role from a specific VO. One consequence of this method is that VOMS attributes can only be used with a proxy, they cannot be attached to a CA-issued certificate.

One other thing to be aware of is that each AC has its own lifetime. This is typically 12 hours as for the proxy, but it is possible for ACs to expire at different times than the proxy.

4.2. FIRST STEPS

Before using the WLCG/EGEE Grid, the user must do the following:

- a. Obtain an X.509 certificate from a (CA) recognized by WLCG/EGEE;
- b. Get registered with WLCG/EGEE by joining one or more Virtual Organisations;
- c. Obtain an account on a machine which has the WLCG/EGEE User Interface software installed, and copy the certificate to it;
- d. Create a proxy on the UI.

Steps a. to c. need to be executed only once to have access to the Grid, although the certificate will usually need to be renewed once a year, and VO membership may also need to be re-confirmed periodically.

Step d. needs to be executed only when a valid proxy is not already available. After the proxy expires a new proxy must be created before Grid services can be used again.

The following sections provide details on these prerequisites.

4.3. OBTAINING A CERTIFICATE

4.3.1. 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 organised geographically and by research institute. Each CA has its own procedure to release certificates.

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

<http://lcg.web.cern.ch/LCG/digital.htm>

For many purposes it may be useful to install the root certificates of Grid CAs in a web browser and/or email client, as this will enable the validation of Grid certificates used in web servers and to sign email. (The way to do this is specific to each piece of software and hence cannot be covered here.) In particular users should usually install the root certificate for their own CA. The root certificates can be obtained from this URL:

<https://www.tacar.org/repos/>

4.3.2. Requesting the Certificate

In order to obtain a certificate, a user must create 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 can be found on the CA website.

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

Note: The user must usually install the CA root certificate in his browser first. This is because the CA has to sign the user certificate using its private key, and the user's browser must be able to validate the signature.

For some CAs the certificate requests are generated using a command line interface. Again, details of the exact command and the requirements of each CA will vary and can be found on the CA's website.

4.3.3. Getting the Certificate

After a request is generated and sent to a CA, the CA will have to confirm that the user asking for a certificate is who he claims he is. This usually involves a physical meeting, or sometimes a phone call, with a **Registration Authority (RA)**, somebody delegated by the CA to verify the legitimacy of a request, and approve it if so. The RA is usually someone at the user's home institute, and will generally need some kind of ID card to prove the user's identity. Other CAs, like the CERN CA, simply ask the user to validate his request with the password of his computing account.

After approval, the certificate is generated and delivered to the user. This can be done via e-mail, or by giving instructions to the user to download it from a web page. If the certificate was directly installed in the user's browser then it must be exported (saved) to disk for Grid use. Details of how to do this will depend on the browser, and are usually described on the CA web site.

The received certificate will usually be in one of two formats: *PEM* (extension `.pem`) or *PKCS12* (extension `.p12` or `.pfx`). The latter is the most common for certificates exported from a browser, but the PEM format is currently needed on a WLCG/EGEE UI. The certificates can be converted from one format to the other using the `openssl` command.

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

```
$ 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 input PKCS12 format file;
<code>userkey.pem</code>	is the output private key file;
<code>usercert.pem</code>	is the output PEM certificate file.

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

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

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

4.3.4. Renewing the Certificate

CAs issue certificates with a limited duration (usually one year); this implies the need to renew them 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 the CA. The certificate usually needs to be renewed **before** the old certificate expires; CAs may send an email to remind users that renewal is necessary, but users should try to be aware of the expiration date.

Renewed certificates have the same SN as the old ones; failing to renew the certificate implies for some CAs the loss of the SN and the necessity to request a completely new certificate with a different SN, which is effectively a new Grid identity.

In the case of the CERN CA, the procedure to renew a certificate is the same as for requesting it the first time, and it can also be done after the expiration of the old one.

4.3.5. Taking Care of Private Keys

A private key is the heart of a Grid identity. Anyone who steals it and knows the passphrase can impersonate the owner, and if it is lost it is no longer possible to do anything in the Grid, so taking care of it is vital. Certificates are issued personally to individuals, and must **never** be shared with other users. To use the Grid a user must agree to an Acceptable Use Policy, which among other things requires him to keep his private key secure.

Proxies also contain private keys, and although these are less valuable, as the lifetime of a proxy is short, it is still important to look after them.

On a UNIX UI the certificate and private key are stored in two files. Typically they are in a directory called `$HOME/.globus` and are named `usercert.pem` and `userkey.pem`, although these can be changed. The certificate is public and can be world-readable, but the key must only be readable by the owner (Grid commands will check this and refuse to work if the permissions are wrong). Ideally the key should be stored on a disk local to the UI rather than e.g. an shared file system, although this is not always possible. If a certificate has been exported from a browser there may also be a PKCS12-format file (`.p12` or `.pfx`) which also contains the private key, and hence this must also be protected (or deleted).

Note: If a private key is stored under AFS, e.g. on LXPLUS at CERN, be aware that access is controlled by the AFS ACLs rather than the normal file permissions, so users must ensure that the key is not in a publicly-readable area.

Web browsers also store private keys internally, and these also need to be protected. The details vary depending on the browser, but password protection should be used if available – this may not be the default (it is not with Internet Explorer). The most secure mode is one in which every use of the private key needs the password to be entered, but this can cause problems as some web sites ask for the certificate many times. If the key is not password-protected it is particularly important to take care that no-one else can get access to a browser session.

A private key stored on a UI must be encrypted, meaning that a passphrase must be typed whenever it is used. A key must **never** be stored without a passphrase. The passphrase should follow similar rules to any computer passwords, but in general should be longer and harder to guess as it gives access to a much larger set of resources than a typical computer system. Users should be aware of the usual risks, like people watching them type or transmitting the passphrase over an insecure link.

A proxy, which includes its own private key, can be stored anywhere, but is typically under `/tmp` (note that this is usually a local area, so when using systems like LXPLUS with many front-end machines, sessions in different windows may not all see the same `/tmp`). The file name is e.g. `x509up_u1234` where 1234 is the uid, but again this can vary. A proxy must only be readable by the owner and there is no passphrase protection.

4.4. REGISTERING WITH WLCG/EGEE

4.4.1. The Registration Service

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

<http://cic.gridops.org/>

Note that some VOs are local and are not registered with WLCG/EGEE as a whole; in this case users should consult local documentation for information about registration procedures.

With your personal certificate loaded on your browser, open:

https://cic.gridops.org/index.php?section=vo&page=download_vodata

select your VO and find the VOMS server via which you can register in the “Enrollment URL” field of the VO identity card.

The registration procedure normally requires the use of a web browser with the user certificate loaded, to enable the request to be properly authenticated. Browsers normally use the PKCS12 certificate format: if the certificate was issued to a user in the 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 certificate available.

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

http://lcg.web.cern.ch/LCG/loading_certifs.htm

4.4.2. Virtual Organisations

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

VO names can be short strings like *cms* or *biomed*, but new VOs use a hierarchical name space, like *vo.pic.es*, to ensure that different VOs will always have distinct names.

Becoming a member of a VO usually requires membership of the corresponding experiment or community; in any case a user must comply with the rules of the VO to gain membership. A user may be expelled from a VO if he fails to comply with these rules.

It is possible to belong to more than one VO, although this is an unusual case. However, using a single certificate with more than one VO requires the recognition of VOMS proxies which is not yet the case for all WLCG/EGEE middleware and services, hence it is currently necessary to have a separate certificate for each VO.

4.5. SETTING UP THE USER ACCOUNT

4.5.1. The User Interface

Apart from registering with WLCG/EGEE, a user must also have an account on a WLCG/EGEE User Interface in order to access the Grid. To obtain such an account, a local system administrator must be contacted, either at the user's own site or at a central site like CERN.

As an example, the CERN LXPLUS service can be used as a UI as described in [31]. To load the proper shell environment, one has to execute the command

```
source /afs/cern.ch/project/gd/LCG-share/current/external/etc/profile.d/grid-env.sh
```

for a **bash** interactive shell.

It is also possible for a user to install the UI software on his own machine, but this is outside the scope of this document.

Once the account has been created, the user certificate must be installed. The usual procedure is to create a directory named `$HOME/.globus` and put the user certificate and key files there, naming them `usercert.pem` and `userkey.pem` respectively, with permissions 444 for the former, and 400 for the latter. A directory listing should give a result similar to this:

```
ls -l $HOME/.globus
total 13
-r--r--r--  1 doe      xy           4541 Aug 23  2009 usercert.pem
-r-----  1 doe      xy           963 Aug 23  2009 userkey.pem
```

4.5.2. Checking a Certificate

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

Example 4.5.2.1 (Retrieving information on a user certificate)

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

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

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

Certificate:

Data:

```

Version: 3 (0x2)
Serial Number: 5 (0x5)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=CH, O=CERN, OU=cern.ch, CN=CERN CA
Validity
  Not Before: Sep 11 11:37:57 2009 GMT
  Not After : Nov 30 12:00:00 2010 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` option for a full list. For example, the `-subject` option returns the Subject Name:

```

$ grid-cert-info -subject
/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe
  
```


or to check the certificate expiration date:

```
$ grid-cert-info -enddate  
Oct 15 05:37:09 2009 GMT
```

or to know which CA issued the certificate:

```
$ grid-cert-info -issuer  
/DC=ch/DC=cern/CN=CERN Trusted Certification Authority
```

Example 4.5.2.2 (Verifying a user certificate)

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

```
$ openssl verify -CApath $X509_CERT_DIR $HOME/.globus/usercert.pem
```

and if the certificate is valid and properly signed, 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: /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=does/CN=123456/CN=John Doe  
error 20 at 0 depth lookup:unable to get local issuer certificate
```

If the environment variable `X509_CERT_DIR` is defined, use `/etc/grid-security/certificates`.

Example 4.5.2.3 (Verifying the consistency between private key and certificate)

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

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

In case of mismatch, the output will be:

```
Your identity: /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=does/CN=123456/CN=John Doe  
Enter GRID pass phrase for this identity:  
Creating proxy ..... Done
```

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

4.6. PROXIES

4.6.1. Standard Proxies

At this point, the user is able to generate a proxy using the command `grid-proxy-init`, which prompts for the user passphrase, as in the next example.

Note: standard proxies are now considered obsolete and VOMS proxies should be used instead for most VOs.

Example 4.6.1.1 (Creating a proxy)

To create a proxy, issue the command:

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

If the command is successful, the output will be like

```
Your identity: /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Tue Jun 24 23:48:44 2009
```

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

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

```
ERROR: Couldn't read user key: Bad passphrase
key file location: /home/doe/.globus/userkey.pem
Use -debug for further information.
```

If the proxy 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). When a proxy has expired, it becomes useless and a new one has to be created with `grid-proxy-init`. However, longer lifetimes imply bigger security risks, and the Grid Acceptable Use Policy generally limits proxy lifetimes to 24 hours — some services may reject proxies with lifetimes which are too long.

Use the option `-help` for a full listing of options.

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

Example 4.6.1.2 (Printing information on a proxy)

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

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

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

```
subject : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe/CN=787058812
issuer  : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe
identity : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe
type    : Proxy draft (pre-RFC) compliant impersonation proxy
strength : 512 bits
path    : /tmp/x509up_u1234
timeleft : 11:59:56
```

If a proxy does not exist, the output is:

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

Example 4.6.1.3 (Destroying a proxy)

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

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

If no proxy exists, the result will be:

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

4.6.2. VOMS Proxies

The *Virtual Organisation Membership Service (VOMS)* is a system which allows a proxy to have *extensions* containing information about the VO, the groups the user belongs to in the VO, and any roles the user is entitled to have.

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

A *role* is an attribute which typically allows a user to acquire special privileges to perform specific tasks. In principle, groups are associated to privileges that the user always has, while roles are associated to privileges that a user needs to have only from time to time. Note that roles are attached to groups, i.e. roles in different groups with the same role name are distinct.

The groups and roles are defined by each VO; they may be assigned to a user at the initial registration, or added subsequently.

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

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

for example, /cms/HeavyIons/Role=production.

Example 4.6.2.1 (Creating a VOMS proxy)

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

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

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

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

```

Your identity: /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=doe/CN=123456/CN=John Doe
Enter GRID pass phrase:
Creating temporary proxy ..... Done
Contacting voms.cern.ch:15002 [/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch]
"cms" Done
Creating proxy ..... Done
Your proxy is valid until Thu Mar 30 06:17:27 2009
  
```

Note that there are two steps: a standard Grid proxy is created first and used to authenticate to the VOMS server,

and the full VOMS proxy is then created using information returned by it. If a valid proxy already exists the `-noregen` option can be used to avoid the first step, including typing the passphrase.

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

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

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

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

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

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

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

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

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

The configuration file must contain lines with the following syntax:

```
alias host port subject vo
```

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

```
"dteam" "voms.cern.ch" "15004"
"/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch" "dteam"
```

Example 4.6.2.2 (Printing information on a VOMS proxy)

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

```

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

Note that there are separate times to expiry for the proxy as a whole and the VOMS extension, which can potentially be different.

4.6.3. Proxy Renewal

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

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

To see if a WLCG/EGEE site has a MyProxy Server, the GOC database [17] may be consulted; MyProxy servers have a node type of PROX. A UI may have a default server defined in the MYPROXY_SERVER environment variable. The main MyProxy server for WLCG is `myproxy.cern.ch`.

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

```

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

The minimum recommended time for the initial proxy is 30 minutes, and in most circumstances it should be substantially longer. Job submission is forbidden for proxies with a remaining lifetime less than 20 minutes: an error message like the following will be produced:

```

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

Management of the proxy renewal functionality is available via the `myproxy` commands. The user must either specify the host name of a MyProxy server, or define it as the value of the `MYPROXY_SERVER` environment variable.

For the WMS to know which MyProxy server to use in the proxy 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, the name of the default MyProxy server is added automatically when the job is submitted. This default is defined in a VO-specific configuration file.

Note: the machine where the WMS runs must be authorized for proxy renewal in the MyProxy server configuration

Example 4.6.3.1 (Creating a long-term proxy and storing it in a MyProxy Server)

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

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

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

The output will be similar to:

```

Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Thu Jul 17 18:57:04 2009
A proxy valid for 168 hours (7.0 days) for user /O=Grid/O=CERN/OU=cern.ch/CN=John Doe
now exists on myproxy.cern.ch.
```

By default, the long-term proxy lasts for one week and the proxies created from it last 12 hours. These lifetimes can be changed using the `-c` and the `-t` option respectively, but cannot be longer than the lifetime of the user certificate.

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

Example 4.6.3.2 (Retrieving information about a long-term proxy)

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

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

where the `-s` and `-d` options have the same meaning as in the previous example. The output is similar to:

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

Note that there must be a valid proxy on the UI, created with `grid-proxy-init` or `voms-proxy-init`, to successfully interact with the long-term proxy on the MyProxy server.

Example 4.6.3.3 (Deleting a long-term proxy)

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

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

and the output is:

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

Again, a valid proxy must exist on the UI.

5. INFORMATION SERVICE

The architecture of the gLite 3.1 Information Service, based on MDS was described in Chapter 3. In this chapter, we have a closer look at the structure of the information published by this service, and we examine some tools that can be used to get information from it. Most middleware components (for Data and Workload Management) currently rely on information from MDS.

Some information about tools used for Grid monitoring in gLite 3.1 is also provided here.

5.1. THE MDS

In the following sections examples are given on how to interrogate the MDS Information Service in gLite 3.1. In particular, the different servers from which the information can be obtained are discussed. These are the local GRISes, the site BDII's and the global (or top-level) BDII's. Of these, the top-level BDII is usually the one queried, since it contains information about the whole Grid infrastructure.

Before the procedure to directly query the MDS is described, two higher level tools, `lcg-info` and `lcg-infosites`, are presented. These tools should be enough for most common user needs and will usually avoid the necessity of direct LDAP queries (although these are very useful for more complex queries).

As explained in Chapter 3, the data in the MDS in WLCG/EGEE conforms to the LDAP implementation of the GLUE Schema, although for historical reasons some extra attributes are also currently published and may be queried and used by clients of the IS. The current implementation relates to version 1.3 of the GLUE schema. For a list of the defined object classes and their attributes, as well as for a reference on the Directory Information Tree used to publish those attributes, please check Appendix F.

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 LDAP servers. 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 5.1.6.

5.1.1. `lcg-info`

The `lcg-info` command is a flexible tool to print the value of any desired CE, SE, service and site attribute and allows to select only instances that satisfy a given set of conditions on their attributes. The information is taken from the BDII specified by the `LCG_GFAL_INFOSYS` environment variable or in the command line; if a comma-separated list of BDII endpoints is provided, the first usable one is queried.

The general format of the command for listing CE or SE information is:

```
$ lcg-info [--list-ce | --list-se | --list-service | --list-site ]
          [--query <query>] [--attrs <attrs>]
```

where one and only one of the options `--list-ce`, `--list-se`, `--list-service` and `--list-site` must be used to indicate if CEs, SEs, services or sites should be listed. The `--query` option introduces a filter (conditions to be fulfilled) to the elements of the list, and the `--attrs` option may be used to specify which attributes to print. For example, if `--list-ce` is specified then only CE attributes are considered (others are just ignored).

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 related 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 5.1.1.1 (Get the list of supported attributes)

To have a list of the supported attributes, use:

```
$ 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 5.1.1.2 (List all the CEs satisfying given conditions and print the desired attributes)

Suppose one wants to know how many jobs are running and how many free CPUs there are on CEs that have an Athlon CPU and have Scientific Linux:

```
$ lcg-info --vo cms --list-ce --query 'Processor=*athlon*,OS=*Scientific*' \
  --attrs 'RunningJobs,FreeCPUs'
```

Note that the `--vo` option must be specified, as `RunningJobs` and `FreeCPUs` depend on the VO.

The output could be:

```
- CE: alice003.nipne.ro:2119/jobmanager-lcgpbs-alice
  - RunningJobs      0
  - FreeCPUs        2
```

```

- CE: alice003.nipne.ro:2119/jobmanager-lcgpbs-dteam
  - RunningJobs      0
  - FreeCPUs        2
[...]
```

It must be stressed that `lcg-info` only supports a logical AND of logical expressions, separated by commas, and the only allowed operator is `=`. In equality comparisons of strings the `*` wildcard matches any number of characters.

Another useful query is 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*'
```

Note that this lists all tags for all VOs for the matching CEs.

Example 5.1.1.3 (List the close CEs for all the SEs)

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

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

the output will be like:

```

- SE: SE.pakgrid.org.pk
  - CloseCE      CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-ops
                  CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-cms
                  CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-dteam

- SE: aliserv1.ct.infn.it
  - CloseCE      _UNDEF_

- SE: arxiloxos2.inp.demokritos.gr
  - CloseCE      arxiloxos1.inp.demokritos.gr:2119/jobmanager-lcgpbs-dteam
                  arxiloxos1.inp.demokritos.gr:2119/jobmanager-lcgpbs-cms
                  arxiloxos1.inp.demokritos.gr:2119/jobmanager-lcgpbs-ops
[...]
```

A value `_UNDEF_` means that the attribute is not defined for that SE or CE.

Example 5.1.1.4 (List all the WMSes which support a VO)

In this case one can execute a command like

```

$ lcg-info --list-service --vo cms --query ServiceType=org.glite.wms.WMPProxy
- Service: grid07.lal.in2p3.fr_org.glite.wms.WMPProxy

- Service: grid25.lal.in2p3.fr_org.glite.wms.WMPProxy

- Service: https://cms-wms.desy.de:7443/glite_wms_wmproxy_server

- Service: https://eu-india-02.pd.infn.it:7443/glite_wms_wmproxy_server
...

```

Example 5.1.1.5 (List all the sites in the information system)

This is easily done by issuing the command

```

$ lcg-info --list-site
- Site: AEGIS01-PHY-SCL

- Site: AEGIS07-PHY-ATLAS

- Site: AGLT2

- Site: ALBERTA-LCG2
...

```

Currently the command does not allow to list the services run at a particular site.

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

5.1.2. lcg-infosites

The `lcg-infosites` command can be used to print out preformatted information on some Grid service types. The syntax is the following:

```
lcg-infosites --vo <vo> <option> -v <verbosity> -f <site> --is <bdii>
```

This is the definition of the most important command options and arguments:

- `--vo <vo>`: the name of the VO to which the information to print is related (mandatory);
- `<option>`: specifies what information has to be printed. It can take the following values:
 - `ce`: the number of CPUs, running jobs, waiting jobs and CE names (global, no VO-specific information);
 - `v 1`: only the CE names;

- v 2: the cluster names, the amount of RAM, the operating system name and version and the processor model;
 - se: the names of the SEs supporting the VO, the type of storage system and the used and available space;
 - v 1: only the SE names;
 - all: the information given by ce and se; together;
 - closeSE: the names of the CEs supporting the VO and their close SEs;
 - tag: the software tags published by each CE supporting the VO;
 - lfc: the hostname of the LFC catalogues available to the VO;
 - lfcLocal: the hostname of the local LFC catalogues available to the VO;
 - rb: the hostname and port of the RBs available to the VO; to the VO;
 - vobox: the VO boxes available to the VO;
 - fts: the endpoints of the FTS servers available to the VO;
 - sitenames: the names of all WLCG/EGEE sites;
- --is <bdii>: the BDII to query. If not specified, the BDII defined in the environment variable LCG_GFAL_INFOSYS will be queried.
 - -f <site>: restricts the information printed to the specified site (it applies only to options rb, dli, vobox and fts).

Example 5.1.2.1 (Obtaining information about computing resources)

The way to get information relating to the computing resources for the *alice* VO is:

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

A typical output is as follows:

#CPU	Free	Total Jobs	Running	Waiting	ComputingElement
15	4	0	0	0	ce002.ipp.acad.bg:2119/jobmanager-lcgpbs-alice
15	4	0	0	0	ce001.ipp.acad.bg:2119/blah-pbs-alice
80	8	0	0	0	ce02.grid.acad.bg:2119/jobmanager-pbs-alice
10	10	0	0	0	ce.hpc.iit.bme.hu:2119/blah-pbs-alice
96	94	0	0	0	grid109.kfki.hu:2119/jobmanager-lcgpbs-alice
3409	6	493	493	0	ce101.cern.ch:2119/jobmanager-lcglsf-grid_alice
[...]					

Example 5.1.2.2 (Obtaining information about storage resources)

To get the status of the storage resources:

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

Avail Space(Kb)	Used Space(Kb)	Type	SEs
39657488	106362948	n.a	se.phy.bg.ac.yu
31400000	18580000	n.a	sel.egee.man.poznan.pl
569586792	47148288	n.a	clrauvergridse01.in2p3.fr
1200000000	410000000	n.a	koala.unimelb.edu.au
22903032	42994124	n.a	se-lcg.sdg.ac.cn
[...]			

Example 5.1.2.3 (Listing the close Storage Elements)

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

```

$ lcg-infosites --vo dteam closeSE

Name of the CE: g02.phy.bg.ac.yu:2119/blah-pbs-dteam
                se.phy.bg.ac.yu

Name of the CE: ce.phy.bg.ac.yu:2119/jobmanager-pbs-dteam
                se.phy.bg.ac.yu

Name of the CE: fangorn.man.poznan.pl:2119/jobmanager-lcgpbs-dteam
                sel.egee.man.poznan.pl
                sel.egee.man.poznan.pl

[...]
```

Example 5.1.2.4 (Listing local LFC servers)

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

```

$ lcg-infosites --vo atlas lfcLocal
lxb2038.cern.ch
pps-lfc.cnaf.infn.it
cclcglfcli03.in2p3.fr
[...]
```

5.1.3. The Local GRIS

The first level of MDS information publication is the GRIS, which provides specific information for a particular service. The GRIS normally runs on the same node as the CE, SE or other service for which it publishes, although it may be on a different node. There is usually no need to query a GRIS directly except for detailed debugging, and in some cases site firewalls may prevent access from external sites.

In order to interrogate the GRIS on a specific node, the hostname and the TCP port where the GRIS run must be specified. The port is normally either 2135 or 2170. 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. If the port is 2170, the `-b` option should be `mds-vo-name=resource, o=grid` (for port 2170).

5.1.4. Using the `ldapsearch` command to read the MDS

For the LDAP implementation of the GLUE schema, the root of the DIT is always `o=grid`. At the GRIS level the next entry is (for historical reasons) either `mds-vo-name=local` or `mds-vo-name=resource`, but at the site level this is replaced with `mds-vo-name=<sitename>`, and a top-level BDII has site entries under `mds-vo-name=<sitename>, mds-vo-name=local, o=grid`. The GLUE entries themselves are at lower levels and always have the same DN structure. For details, please refer to Appendix F.

The same effect as the command above 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 5.1.4.1 (Interrogating the GRIS on a Computing Element)

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

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

or:

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

In order to restrict the search, a filter of the form `attribute operator value` can be used. The operator is one of those defined in the following table (note that `<` and `>` are not included):

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 any 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 such expressions is the following:

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

Example of search filters are:

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

It is possible to construct complex queries, but the syntax is not very intuitive so some experimentation may be needed. Be aware that filters may need to be escaped to prevent special characters being interpreted by the shell.

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. Alternatively, **grep** or other Unix tools can be used to postprocess the output.

A description of all objectclasses and their attributes usable with the `ldapsearch` command can be found in Appendix F.

Example 5.1.4.2 *(Finding out to which site a CE belongs)*

This example illustrates a non trivial two-step query:

- find out the cluster connected to a given CE;
- find out in which site the cluster is located.

The first query is, for example:

```
$ ldapsearch -LLL -x -h cel12.cern.ch:2170 -b 'o=grid' \
  'GlueCEUniqueID=cel12.cern.ch:2119/jobmanager-lcglsf-grid_cms' GlueForeignKey

dn: GlueCEUniqueID=cel12.cern.ch:2119/jobmanager-lcglsf-grid_cms,Mds-Vo-name=resource,
o=grid
GlueForeignKey: GlueClusterUniqueID=cel12.cern.ch
```

This allows us to find that the cluster unique identifier is `cel12.cern.ch`.

Finally:


```

$ ldapsearch -LLL -x -h ce112.cern.ch:2170 -b 'o=grid' \
  'GlueClusterUniqueID=ce112.cern.ch' GlueForeignKey | grep GlueSiteUniqueID
  
```

GlueForeignKey: GlueSiteUniqueID=CERN-PROD

The site is CERN-PROD.

In this example the GRIS of the CE was queried; an identical result would have been produced by a query to a top BDII or to the site BDII.

The `-LLL` option was used to produce a more concise output.

5.1.5. The Site BDII

At each site, a site BDII collects information about all resources present at a site (i.e. data from all GRISes at the site). In this section we explain how to query a site BDII.

Often the site BDII runs on a Computing Element, although it may be on a separate node. The port used to interrogate a site BDII is usually 2170. The DIT base name is based on the site name. However, it is sufficient to use a base of `o=grid` in LDAP queries.

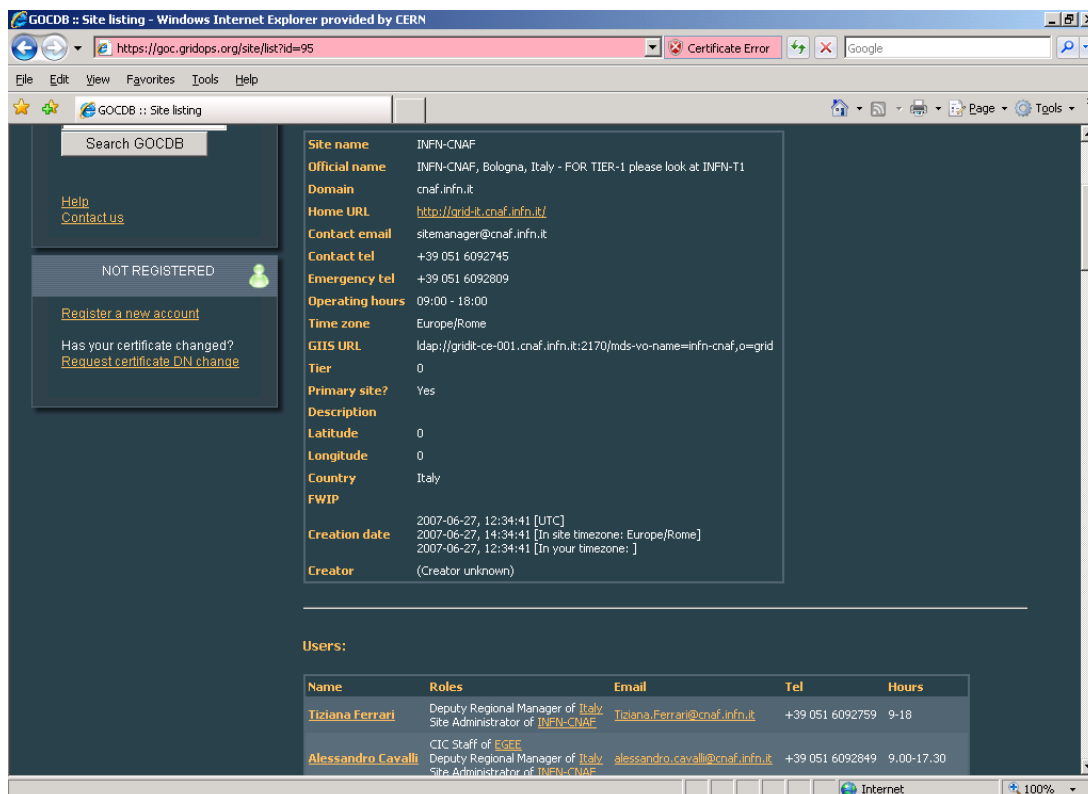


Figure 7: The GOCDB information page for the INFN-CNAF site

For a list of all sites and all resources present, refer to the GOC database [17].

This is the site information for the CNAF site (at Bologna in Italy), as shown in Figure 7. The BDII URL is:

```
ldap://gridit-ce-001.cnaf.infn.it:2170/mds-vo-name=infn-cnaf,o=grid
```

In order to interrogate it, use the ldapsearch command as follows.

Example 5.1.5.1 (Interrogating a site BDII)

```

$ ldapsearch -x -h gridit-ce-001.cnaf.infn.it -p 2170 \
  -b mds-vo-name=infn-cnaf,o=grid

version: 2

#
# filter: (objectclass=*)
# requesting: ALL
#
# INFN-CNAF, grid
dn: Mds-Vo-name=INFN-CNAF,o=grid
objectClass: GlueTop
objectClass: Mds
Mds-Vo-name: INFN-CNAF

# wms015.cnaf.infn.it_org.glite.wms.WMProxy_1944762328, INFN-CNAF, grid
dn: GlueServiceUniqueID=wms015.cnaf.infn.it_org.glite.wms.WMProxy_1944762328,M
  ds-Vo-name=INFN-CNAF,o=grid
GlueServiceAccessControlBaseRule: VO:cms
GlueServiceAccessControlBaseRule: VO:dteam
GlueServiceAccessControlBaseRule: VO:ops
GlueServiceStatus: OK
objectClass: GlueTop
objectClass: GlueService
objectClass: GlueKey
objectClass: GlueSchemaVersion
GlueServiceUniqueID: wms015.cnaf.infn.it_org.glite.wms.WMProxy_1944762328
GlueServiceAccessControlRule: cms
GlueServiceAccessControlRule: dteam
GlueServiceAccessControlRule: ops
GlueServiceEndpoint: https://wms015.cnaf.infn.it:7443/glite_wms_wmproxy_server
GlueServiceVersion: 3.2.1
GlueSchemaVersionMinor: 3
GlueServiceName: INFN-CNAF-WMProxy
GlueServiceType: org.glite.wms.WMProxy
GlueServiceWSDL: http://trinity.datamat.it/projects/EGEE/WMProxy/WMProxy.wsdl
GlueServiceSemantics: https://edms.cern.ch/file/674643/1/EGEE-JRA1-TEC-674643-
  WMProxy-guide-v0-3.pdf
GlueForeignKey: GlueSiteUniqueID=INFN-CNAF

```

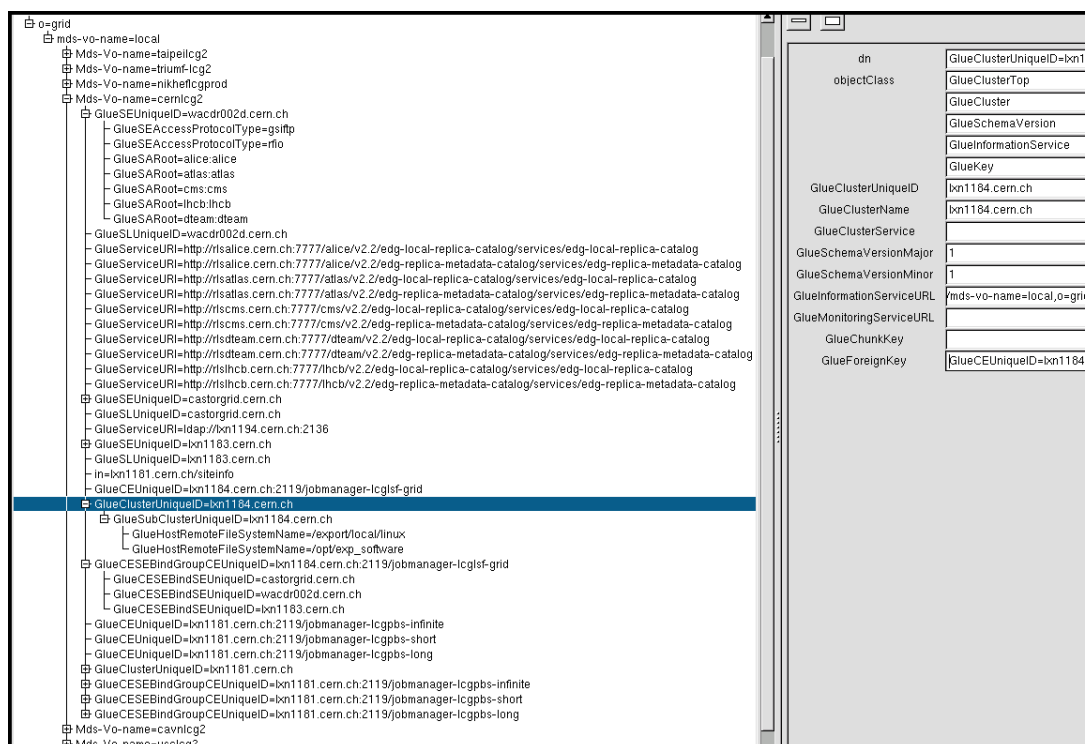
```

GlueServiceOwner: cms
GlueServiceOwner: dteam
GlueServiceOwner: ops
GlueSchemaVersionMajor: 1
GlueServiceStartTime: 2009-12-15T08:05:01+01:00
GlueServiceStatusInfo: WMPProxy httpd listening on port 7443
[...]
  
```

5.1.6. The top-level BDII

A top-level BDII collects all information coming from site BDII's and stores them in a cache. The top-level BDII can be configured to collect published information from resources in all sites in a Grid (usually derived from the GOC DB), or just from a subset of them. The site list is normally filtered to include only sites which are currently operational, and VOs can also apply their own filters to exclude sites which are currently failing certain critical tests, so the sites visible in a BDII may fluctuate.

In order to find the location of a top-level BDII at a site (if any), consult the GOCDB page for the site. The BDII will be listed with the rest of the nodes of the site (refer to Figure 7, node type BDII), and the entry may also include comments about the purpose and content of the BDII. The main top-level BDII for WLCG is `lcg-bdii.cern.ch`.



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=lxn1184.cern.ch`. On the right, a detailed view of this node is shown, listing various attributes and their values.

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

Figure 8: The LDAP directory of a gLite 3.1 BDII

A BDII can be interrogated using the base name `mids-vo-name=local,o=grid` (although it suffices to use `o=grid`) and port 2170. The subtree corresponding to a particular site appears under an entry with a DN like:

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

In Figure 8, a view of the DIT of a BDII in gLite 3.1 is shown. In the figure, only the subtree 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 and for various services, can be seen in the figure.

Each entry can contain attributes from different object classes. This can be seen in the entry with DN `GlueClusterUniqueID=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`. However, one of the object classes is the primary one for the object, in this case `GlueCluster`, and an attribute from it is used to form the DN. Since every object in the tree must have a unique DN the attribute used must be unique at least within its branch of the tree.

In the right-hand side of the window, the DN of the selected entry and the names and values (in the cases where they exist) 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 can be seen, a graphical tool can be quite useful to examine the structure (and indeed the details) of an MDS directory. In addition, the schema (object classes and attributes) can be also examined.

Example 5.1.6.1 (Interrogating a BDII)

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

```

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

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

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

[...]
```

Example 5.1.6.2 (Listing all the CEs which publish a given tag)

The attribute `GlueHostApplicationSoftwareRunTimeEnvironment` can be used to publish experiment-specific information (*tags*) for a CE, for example to indicate that a given set of experiment software is installed. To list

all the CEs which publish a given tag, a query to a BDII can be performed. In this example, the information is retrieved for all subclusters:

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

Example 5.1.6.3 (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 the *alice* VO, together with the storage space available in each of them, a query similar to this can be used:

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

where the `GlueSAAccessControlBaseRule` attribute contains the name of the supported VO. The obtained result will be something like the following:

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

dn: GlueSALocalID=alice,GlueSEUniqueID=grid08.ph.gla.ac.uk,mds-vo-name=UKI-Scot
  tGrid-Gla-PPS,mds-vo-name=local,o=grid
GlueSAStateAvailableSpace: 3840000000
GlueSAStateUsedSpace: 1360000000
GlueChunkKey: GlueSEUniqueID=grid08.ph.gla.ac.uk

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

5.2. SERVICE DISCOVERY

The gLite 3.1 Service Discovery (SD) API makes it possible to access service details published to the Information Systems. The main purpose is to answer questions like: *I am at CERN, in the ops VO. Where is a MyProxy server?* It therefore represents a simplified view of the Grid Information System to locate resources/services and query their properties. The SD interface supports several information systems, currently MDS, R-GMA, Globus MDS4,

and a local `service.xml` file. The client access mode to the underlying infrastructure that holds the information is set by environment variables.

- **BDII:** the environment variable `LCG_GFAL_INFOSYS` must contain the hostname and port number of the BDII service to query;
- **FILE:** the default configuration file is in `$GLITE_LOCATION/etc/services.xml` and can be overridden by `$HOME/.glite/etc/services.xml`.

The gLite 3.1 Service Discovery API provides interfaces for the Java and C/C++ programming languages, and a command line interface (`glite-sd-query`). The gLite 3.1 Service Discovery User Guide [53] offers a comprehensive documentation of the SD APIs.

5.2.1. Running a Service Discovery query

In order to use Service Discovery, the user has to set the `GLITE_SD_PLUGIN` variable to specify the Information System(s) to be queried. To use all of the R-GMA, BDII and XML file-based information systems set:

```
GLITE_SD_PLUGIN="bdii,file"
```

The API will then try each in turn until one of them returns something.

The `glite-sd-query` command allows the listing of basic information about known services. To list detailed information about all services at a site, use the `-s` option instead:

```
glite-sd-query -x -s CERN-PROD
```

If a user wants to know more details about a specific service of type *myproxy* located at CERN, the same command is used with the following options:

```
$ glite-sd-query -t MyProxy -s CERN-PROD
```

```
Name: myproxy.cern.ch:7512  
Type: MyProxy  
Endpoint: myproxy.cern.ch:7512  
Version: 1.1.0
```

Note: as an alternative to the `-s` option, the environment variable `GLITE_SD_SITE` can be used to restrict the search to a given site. The value of the variable can be either the name of the site in the GOC DB or the DNS domain of the site.

Note: the type of service is specified as part of the GLUE schema, and may have a more complex format: for example, the FTS service has a type of `org.glite.FileTransfer`.

The `-h` option shows all available options.

5.3. MONITORING

The ability to monitor resource related parameters is currently considered a necessary functionality in any network. In such a heterogeneous and complex system as the Grid, this necessity becomes fundamental. A monitoring system implies the existence of a central repository of operational information (in WLCG/EGEE, the GOCDB). The monitoring system should be able to collect data from the resources in the system, in order to analyze the usage, behavior and performance of the Grid, detect and notify fault conditions, contract violations and user-defined events.

The GOC web page contains a whole section concerning monitoring information for WLCG/EGEE. Several different monitoring tools are in use, including general-purpose monitoring tools and Grid specific systems.

Also important are the web pages publishing the results of functional tests applied periodically using the Service Availability Monitoring (SAM) to all the sites registered within WLCG/EGEE. The results of these 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 BDI 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 having failures in the standard test reports. If that is the case, the site will already have been notified of the problems by the grid operations staff. Also, the site details in the GOCDB will show if the site is currently in scheduled downtime. The results of some sets of functional sites can be checked in the following URLs:

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

<https://lcg-sam.cern.ch:8443/sam/sam.py>

6. WORKLOAD MANAGEMENT

6.1. INTRODUCTION

The *Workload Management System (WMS)* is the gLite 3.1 component that allows users to submit *jobs*, and performs all tasks required to execute them, without exposing the user to the complexity of the Grid. It is the responsibility of the user to describe his jobs and their requirements, and to retrieve the output when the jobs are finished.

In the following sections, we will describe the basic concepts of the language used to describe a job, the basic command line interface to submit and manage simple jobs, a description of more advanced job types, details on how to configure the command line interface, and some user tools related to job management.

6.2. THE JOB DESCRIPTION LANGUAGE

The *Job Description Language (JDL)* is a high-level language based on the *Classified Advertisement (ClassAd) language* [33], used to describe jobs and aggregates of jobs with arbitrary dependency relations. The JDL is used in WLCG/EGEE to specify the desired job characteristics and constraints, which are taken into account by the WMS to select the best resource to execute the job.

The fundamentals of the JDL are given in this section. A complete description of the JDL syntax is out of the scope of this guide, and can be found in [35]. The description of the JDL attributes is in [36][38].

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

attribute = expression;

Expressions can span several lines, but only the last one must be terminated by a semicolon. Literal strings are enclosed in double quotes. If a string itself contains double quotes, they must be escaped with a backslash (e.g.: Arguments = "\"hello\" 10"). The character “ ” cannot be used in the JDL.

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

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

Example 6.2.1 (Define a simple job)

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

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

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

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

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

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

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

The attributes `StdOutput` and `StdError` define the name of the files containing the standard output and standard error of the executable, once the job output is retrieved. For the standard input, an input file can be similarly specified:

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

but this attribute is rarely used.

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

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

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

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

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

where the file names are paths relative to the work directory of the job (the current directory when the executable starts).

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

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

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

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

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

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

Note: a common error is to write `VirtualOrganization`: it will not work.

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

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

where `test.sh` could be, for example:

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

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

Example 6.2.2 (Specifying requirements on the CE)

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

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 combined in a single `Requirements` attribute.

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

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

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

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

or to any queue in a CE:

```
Requirements = other.GlueCEInfoHostName == "lxshare0286.cern.ch";
```

Note: as explained in 6.5, normally the condition that a CE is in production state is automatically added to the `Requirements` attribute. Thus, CEs that do not correctly publish this will not match. This condition is, nevertheless, configurable.

If the job duration is significant, it is strongly advised to put a requirement on the maximum CPU time, or the wallclock time (expressed in minutes), needed for the job to complete. For example, to express the fact that the job could need up to eight CPU hours and 12 wallclock hours, this expression should be used:

```
Requirements = other.GlueCEPolicyMaxCPUTime > 480 &&
               other.GlueCEPolicyMaxWallClockTime > 720;
```

Note: if a job exceeds the time limits of the queue where it is running, it will be killed by the batch system. Currently, the WMS does not always report correctly to the user that the job failed due to exceeded time limits, if it cannot distinguish this case from an abrupt death of the job due to other causes: this can happen only if the batch system properly signals the job wrapper that the job is about to be killed.

Note: the CPU time needed by a job is inversely proportional to the “speed” of the CPU, which is expressed by the `GlueHostBenchmarkSI00` attribute. To take into account the differences in speed of the CPUs in different CEs, the CPU time should be rescaled to the speed. If, for example, the job needs 720 minutes on a CPU with a speed of 1000, the correct requirement should be

```
Requirements = other.GlueCEPolicyMaxCPUTime >
               (720 * 1000 / other.GlueHostBenchmarkSI00);
```

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("VO-cms-CMSSW_2_0_0",
                    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 fact, the `GlueHostApplicationSoftwareRunTimeEnvironment` attribute is a list of strings and is used to publish any VO-specific information relative to the CE (typically, information on the VO software available on that CE).

Example 6.2.3 *(Specifying requirements using wildcards)*

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

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

The opposite can be required by using:

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

Example 6.2.4 *(Specifying OS and architecture of the CE)*

The user might need to be able to specify which types of CE are to be used to process the job. Here are a couple of examples that provide JDL setups to allow this for many popular operating systems and machine architectures.

Red Hat Enterprise Linux 3, binary release

```
Requirements = (( other.GlueHostOperatingSystemName == "CentOS" ||
                  other.GlueHostOperatingSystemName == "RedHatEnterpriseAS"
                  ) &&
                ( other.GlueHostOperatingSystemRelease >= 3.0 &&
                  other.GlueHostOperatingSystemRelease < 4.0
                  )
                ) ||
                (( other.GlueHostOperatingSystemName == "Scientific Linux" ||
                  other.GlueHostOperatingSystemName == "Scientific Linux CERN"
                  ) &&
                ( RegExp("3\.[0-9]\.[0-9]", other.GlueHostOperatingSystemRelease)
                  )
                )
                ;
```

The wildcard at the end makes less assumptions about specific OS builds, so it is recommended to include it for compatibility and reliability reasons.

Red Hat Enterprise Linux 4, binary release

```

Requirements = ( other.GlueHostOperatingSystemName == "CentOS" ||
                 other.GlueHostOperatingSystemName == "RedHatEnterpriseAS" ||
                 other.GlueHostOperatingSystemName == "ScientificSL" ||
                 other.GlueHostOperatingSystemName == "ScientificCERNSLC"
               ) &&
               ( other.GlueHostOperatingSystemRelease >= 4.0 &&
                 other.GlueHostOperatingSystemRelease < 5.0
               ) ;
  
```

or use

```

SN    = other.GlueHostOperatingSystemName ;
SR    = other.GlueHostOperatingSystemRelease ;

RHEL4 = ( SN == "CentOS" || SN == "RedHatEnterpriseAS" ||
          SN == "ScientificSL" || SN == "ScientificCERNSLC"
        ) &&
        ( SR >= 4.0 && SR < 5.0
        ) ;

Requirements = RHEL4 ;
  
```

to make the requirements less contrived, as above.

The following examples lists some requirements to use in the user's JDL file for matching specific CE architectures:

Intel i386-i686

```

Requirements = other.GlueHostArchitecturePlatformType is UNDEFINED ||
               RegExp("i[3456]86", other.GlueHostArchitecturePlatformType) ;
  
```

This requirement also allows sites where this value is not defined.

64-bit Xeon and Opteron

```

Requirements = (other.GlueHostArchitecturePlatformType == "x86_64") ;
  
```

64-bit Itanium

```

Requirements = (other.GlueHostArchitecturePlatformType == "ia64") ;
  
```

Example 6.2.5 (Specifying requirements on a close SE)

In order to specify requirements on the SE “close” to the CE where the job should run, one can use an expression like:

```
Member("castorsrm.pic.es", other.GlueCESEBindGroupSEUniqueID);
```

It is not possible, though, to write down requirements on SE properties other than their name.

Example 6.2.6 (A complex requirement used in gLite 3.1)

The following example has been actually used by the *alice* VO 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 up to one day (i.e., so that the job is not aborted before it has time to finish).

```
Requirements = other.GlueHostNetworkAdapterOutboundIP==true &&
Member("VO-alice-AliEn", other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
Member("VO-alice-ALICE-v4-01", other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
(other.GlueCEPolicyMaxWallClockTime > 1440 );
```

Example 6.2.7 (Using the automatic resubmission)

It is possible to have the WMS automatically resubmitting jobs which, for some reason, are aborted by the Grid. Two kinds of resubmission are available for the gLite 3.1 WMS: the **deep resubmission** and the **shallow resubmission**. The resubmission is deep when the job fails after it has started running on the WN, and shallow otherwise.

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

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

It is advisable to disable the deep resubmission, as it may happen that a job fails after it has already done something (for example, creating a Grid file), or the WMS thinks that a still running job has failed; depending on the job, the resubmission of an identical job might generate inconsistencies. On the other hand, the shallow resubmission is extremely useful to improve the chances of a job being correctly executed, and it is strongly recommended to use it.

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

Example 6.2.8 (Using the automatic proxy renewal)

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

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

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

```
MyProxyServer = "";
```

Example 6.2.9 (Defining the “goodness” of a CE)

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

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

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

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

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

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

6.3. THE COMMAND LINE INTERFACE

In this section, all commands available for the user in the gLite CLI to manage jobs are described [28][27].

The gLite WMS implements a new service to manage jobs: the *WMProxy*. In previous gLite versions, it also supported the *Network Server*, which was eventually removed. The WMProxy implements several new functionalities, among which:

- submission of job collections;
- faster authentication;
- faster match-making;
- faster response time for users;

- higher job throughput.

The following table summarizes the main commands with their most commonly used options, and the use of these commands will be described in the following sections.

Function	gLite WMS
Submit a job	<code>glite-wms-job-submit [-d delegID] [-a] [-o joblist] jdlfile</code>
See job status	<code>glite-wms-job-status [-v verbosity] [-i joblist] jobIDs</code>
See job logging information	<code>glite-wms-job-logging-info [-v verbosity] [-i joblist] jobIDs</code>
Retrieve job output	<code>glite-wms-job-output [-dir outdir] [-i joblist] jobIDs</code>
Cancel a job	<code>glite-wms-job-cancel [-i joblist] jobID</code>
List available resources	<code>glite-wms-job-list-match [-d delegID] [-a] jdlfile</code>
Delegate proxy	<code>glite-wms-job-delegate-proxy -d delegID</code>

6.3.1. Single Job Submission

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

```
glite-wms-job-submit -a jdlfile
```

where `jdlfile` is a file containing the job description, usually with extension `.jdl`. The `-a` option for the `gLite` command is necessary to automatically delegate a user proxy to the `WMProxy` server (see later).

Example 6.3.1.1 (Submitting a simple job)

Create a file `test.jdl` with this content:

```
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 standard error are redirected to the files `std.out` and `std.err` respectively; the `OutputSandbox` attribute ensures that they are transferred back to the User Interface after the job is finished.

Now, submit the job via `gLite WMS` by doing:


```
$ glite-wms-job-submit -a test.jdl
```

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

```
Connecting to the service https://wms104.cern.ch:7443/glite_wms_wmproxy_server
```

```
===== glite-wms-job-submit Success =====
```

```
The job has been successfully submitted to the WMPProxy
Your job identifier is:
```

```
https://lb102.cern.ch:9000/vZKKk3gdBla6RySximq_vQ
```

```
=====
```

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

The command returns to the user the job identifier (*jobID*), which uniquely defines 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://<LB_hostname>[:<port>]/<unique_string>
```

where <unique_string> is guaranteed to be unique and <LB_hostname> is the host name of the Logging & Bookkeeping (LB) server for the job, which may sit on the WMS used to submit the job, or on a separate machine (to improve the scalability of the WMS).

Note: the jobID is **not** a web URL.

Note: to submit jobs via gLite WMS, it is required to have a VOMS proxy, as with a standard proxy the submission will fail with an error like:

```
Error - Operation failed
Unable to delegate the credential to the endpoint:
https://wms104.cern.ch:7443/glite_wms_wmproxy_server
User not authorized:
unable to check credential permission (/opt/glite/etc/glite_wms_wmproxy.gacl)
(credential entry not found)
credential type: person
input dn: /C=CH/O=CERN/OU=GRID/CN=John Doe
```

If the command returns the following error:

```
Error - WMPProxy Server Error
LCMAPS failed to map user credential

Method: getFreeQuota
Error code: 1208
```

it means that there are authentication problems between the UI and the WMPProxy server (you might not be authorized to use that WMPProxy server).

Many options are available to the job submission commands.

The user proxy must have VOMS extensions. If a submission with a standard proxy is attempted, at submission time this error will be returned:

```
Connecting to the service https://wms012.cnaf.infn.it:7443/glite_wms_wmproxy_server
```

```
Warning - Unable to delegate the credential to the endpoint:
https://wms012.cnaf.infn.it:7443/glite_wms_wmproxy_server
Stack dump
```

The `-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, and it is a convenient way to keep trace of one's jobs.

The `-r <CEId>` option is used to directly send a job to a particular CE. If used, the match making will not be carried out (see Section 6.3.5). The drawback is that the BrokerInfo file, which provides information about the evolution of the job, will not be created, and therefore the use of this option is discouraged.

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

```
<CE_hostname>:<port>/jobmanager-<service>-<queue>           (for a LCG CE)
```

where `<CE_hostname>` and `<port>` are the host name of the machine and the port where the Grid Gate is running (the Globus Gatekeeper for the LCG CE), `<queue>` is the name of one of the corresponding LRMS queue, and `<service>` is the LRMS type, such as `lsf`, `pbs`, `condor`. An example of CEId is:

```
adc0015.cern.ch:2119/jobmanager-lcgpbs-infinite           (LCG CE)
```

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

It is possible to see which CEs are eligible to run a job described by a given JDL using:

```
glite-wms-job-list-match -a <jdl_file>
```

The `--rank` option can be used to display the ranking value of each matching resource.

```
$ glite-wms-job-list-match -a --rank test.jdl
```

```
Connecting to the service https://wms104.cern.ch:7443/glite_wms_wmproxy_server
```

```
=====
```

COMPUTING ELEMENT IDs LIST

The following CE(s) matching your job requirements have been found:

CEId	*Rank*
- CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-cms	0
- grid-ce0.desy.de:2119/jobmanager-lcgpbs-cms	-10
- gw-2.ccc.ucl.ac.uk:2119/jobmanager-sge-default	-56
- grid-ce2.desy.de:2119/jobmanager-lcgpbs-cms	-107

=====
 The `-o <file path>` option can be used to store the CE list on a file.

Example 6.3.1.3 *(Delegating a proxy to WMProxy)*

Each job submitted to the gLite WMS must be associated to a proxy credential previously delegated by the owner of the job to the WMProxy server. This proxy is then used any time WMProxy needs to interact with other services for job related operations.

There are two ways to delegate one's credentials to WMProxy: either by having the delegation performed automatically each time a job is submitted, or by explicitly creating a named delegated credential on the WMProxy server, and subsequently referring to it each time a job is submitted. The advantage of the former method is simplicity, while that of the latter is better performance (as delegation may require a non-negligible amount of time).

We have already used the automatic delegation in the previous examples, where the `-a` was used with `glite-wms-job-submit` and `glite-wms-job-list-match`. To explicitly delegate a user proxy to WMProxy, the command to use is:

```
glite-wms-job-delegate-proxy -d <delegID>
```

where `<delegID>` is a string chosen by the user. Subsequent invocations of `glite-wms-job-submit` and `glite-wms-job-list-match` can bypass the delegation of a new proxy if the same `<delegID>` is given to the `-d` option. For example, to delegate a proxy:

```
$ glite-wms-job-delegate-proxy -d mydelegID
```

```
Connecting to the service https://wms104.cern.ch:7443/glite_wms_wmproxy_server
```

```
===== glite-wms-job-delegate-proxy Success =====
```

```
Your proxy has been successfully delegated to the WMProxy:
https://wms104.cern.ch:7443/glite_wms_wmproxy_server
```

```
with the delegation identifier: mydelegID
```

=====
 Alternatively, we can have the system to generate a random `<delegID>` by doing instead:

```

$ glite-wms-job-delegate-proxy -a

Connecting to the service https://wms104.cern.ch:7443/glite_wms_wmproxy_server

===== glite-wms-job-delegate-proxy Success =====

Your proxy has been successfully delegated to the WMPProxy:
https://wms104.cern.ch:7443/glite_wms_wmproxy_server

with the delegation identifier: 2cBscH0taSqCYcH8fNYncw

=====
  
```

Then, to submit a job:

```
$ glite-wms-job-submit -d mydelegID test.jdl
```

Note: due to a current bug, if many WMPProxy servers are indicated in the UI configuration, `glite-wms-job-delegate-proxy` will delegate a proxy to one of them chosen at random, not to all of them. This limits the usability of the explicit delegation.

6.3.2. Job Operations

After a job is submitted, it is possible to see its current status, to retrieve a complete log of the job history, to recover its output when it is finished, and if needed to cancel it if it has not yet finished. The following examples explain how.

Example 6.3.2.1 (Retrieving the status of a job)

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

```
glite-wms-job-status <jobID>
```

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

```

$ glite-wms-job-status https://wms104.cern.ch:9000/fNdD4FW_Xxkt2s2aZJeoeg

*****
BOOKKEEPING INFORMATION:

Status info for the Job : https://wms104.cern.ch:9000/fNdD4FW_Xxkt2s2aZJeoeg
Current Status:      Done (Success)
Exit code:           0
Status Reason:      Job terminated successfully
  
```

```

Destination:      cel.inrne.bas.bg:2119/jobmanager-lcgpbs-cms
Submitted:       Mon Dec  4 15:05:43 2007 CET
*****
  
```

which contains the time when the job was submitted, the current status of the job, and the reason for being in that status (which may be especially helpful for the ABORTED status). The possible states in which a job can be found were introduced in Section 3.4.1, and are summarized in Appendix C. Finally, the `Destination` field contains the CEId of the CE where the job has been submitted and the job exit code, if the job is finished.

The verbosity level controls the amount of information provided. The value of the `-v` option ranges from 0 to 3 (the default is configurable in the UI). See [29] for detailed information on each of the fields returned.

The commands to get the job status can have several jobIDs as arguments, i.e.:

```
glite-wms-job-status <jobID1> ... <jobIDN>
```

or, more conveniently, the `-i <file_path>` option can be used to specify a file with a list of jobIDs (possibly created by the `-o` option of a job submission command). In this case, the command asks the user interactively the status of which job(s) should be printed. The `--noint` option suppresses the interactivity and all the jobs are considered.

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 <status>` option makes the command retrieve only the jobs that are in the specified status, and the `--exclude <status>` option makes it retrieve jobs that are not in the specified status. This two last 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 status 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 status.

```

$ glite-wms-job-status --all -s DONE -s RUNNING
$ glite-wms-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 [29].

Finally, with the option `-o <file_path>` the command output can be written to a file.

Example 6.3.2.2 (Canceling a job)

A job can be canceled before it ends using the commands

```
glite-wms-job-cancel <jobID>
```

A job must be canceled using the command corresponding to the WMS flavour used to submit the job.

These commands require as arguments one or more JobIDs. For example:

```

$ glite-wms-job-cancel https://wms104.cern.ch:9000/P1c60RFsrIZ9mnBALa7yZA
Are you sure you want to remove specified job(s) [y/n]y : y
Connecting to the service https://128.142.160.93:7443/glite_wms_wmproxy_server

===== glite-wms-job-cancel Success =====
The cancellation request has been successfully submitted for the following job(s):
- https://wms104.cern.ch:9000/P1c60RFsrIZ9mnBALa7yZA
=====
  
```

If the cancellation is successful, the job will terminate in status CANCELLED.

Example 6.3.2.3 (Retrieving the output of a job)

If the job has successfully finished (it has reached the DONE status), its output can be copied to the UI with the command

```
glite-wms-job-output <jobID>
```

The job output must be retrieved using the command corresponding to the WMS flavour used to submit the job.

For example:

```

$ glite-wms-job-output https://wms104.cern.ch:9000/yabp72aERhofLA6W2-LrJw
Connecting to the service https://128.142.160.93:7443/glite_wms_wmproxy_server

=====

JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
https://wms104.cern.ch:9000/yabp72aERhofLA6W2-LrJw
have been successfully retrieved and stored in the directory:
/tmp/dae_yabp72aERhofLA6W2-LrJw
=====
  
```

The default location for storing the outputs (normally /tmp) is defined in the UI configuration, but it is possible to specify in which directory to save the output using the `--dir <path_name>` option.

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

Example 6.3.2.4 (Retrieving logging information about submitted jobs)

A complete history of a job is permanently stored in the Logging & Bookkeeping service and can be retrieved using the command:

```
glite-wms-job-logging-info <jobID>
```

This functionality is especially useful in the analysis of job failures, although the information provided is sometimes difficult to interpret.

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.

The following is the typical sequence of events for a successful job:

```
$ glite-wms-job-logging-info https://wms104.cern.ch:9000/hk0VSbNhe59j0Buo24G_qw
*****
LOGGING INFORMATION:

Printing info for the Job : https://wms104.cern.ch:9000/hk0VSbNhe59j0Buo24G_qw

    ---
Event: RegJob
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:03 2007 CET
    ---
Event: RegJob
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:04 2007 CET
    ---
Event: UserTag
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:04 2007 CET
    ---
Event: UserTag
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:04 2007 CET
    ---
Event: UserTag
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:04 2007 CET
    ---
```

```

Event: Accepted
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:08 2007 CET
  ---
Event: EnQueued
- result           = START
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:08 2007 CET
  ---
Event: EnQueued
- result           = OK
- source           = NetworkServer
- timestamp        = Thu Dec 14 14:35:08 2007 CET
  ---
Event: DeQueued
- source           = WorkloadManager
- timestamp        = Thu Dec 14 14:35:09 2007 CET
  ---
Event: Match
- dest_id          = fangorn.man.poznan.pl:2119/jobmanager-lcgpbs-cms
- source           = WorkloadManager
- timestamp        = Thu Dec 14 14:35:18 2007 CET
  ---
Event: EnQueued
- result           = START
- source           = WorkloadManager
- timestamp        = Thu Dec 14 14:35:18 2007 CET
  ---
Event: EnQueued
- result           = OK
- source           = WorkloadManager
- timestamp        = Thu Dec 14 14:35:18 2007 CET
  ---
Event: DeQueued
- source           = JobController
- timestamp        = Thu Dec 14 14:35:18 2007 CET
  ---
Event: Transfer
- destination      = LogMonitor
- result           = START
- source           = JobController
- timestamp        = Thu Dec 14 14:35:18 2007 CET
  ---
Event: Transfer
- destination      = LogMonitor
- result           = OK
- source           = JobController
- timestamp        = Thu Dec 14 14:35:19 2007 CET
  ---
Event: Accepted
- source           = LogMonitor
- timestamp        = Thu Dec 14 14:35:29 2007 CET
  
```



```

---
Event: Transfer
- destination      = LRMS
- result          = OK
- source          = LogMonitor
- timestamp       = Thu Dec 14 14:35:50 2007 CET
---
Event: Running
- source          = LogMonitor
- timestamp       = Thu Dec 14 14:38:09 2007 CET
---
Event: ReallyRunning
- source          = LogMonitor
- timestamp       = Thu Dec 14 14:41:26 2007 CET
---
Event: Done
- source          = LogMonitor
- timestamp       = Thu Dec 14 14:41:26 2007 CET

```

Note: in order to make easier to debug problems with the WMS, when asking for help for problems related to job submission and management, it is highly advisable to send the output of

```
glite-wms-job-logging-info -v 3 <jobID>
```

that is, using the highest level of verbosity.

6.3.3. Advanced Sandbox Management

A new feature introduced by the gLite WMS is the possibility to indicate input sandbox files stored not on the UI, but on a GridFTP server, and, similarly, to specify that output files should be transferred to a GridFTP server when the job finishes. This has several advantages:

- the input files do not have to be on the host from which the job is submitted;
- the output files are immediately available when the job ends, without having to issue a command to retrieve them;
- the sandbox files do not have to go through the WMS host, which otherwise can easily become a bottleneck.

The following examples show how to use this feature.

Example 6.3.3.1 (Using input files on a GridFTP server)

If the job input files are stored on a GridFTP server, it is possible to specify those files as GridFTP URI in the `InputSandbox` attribute:

```
InputSandbox = {"gsiftp://lxb0707.cern.ch/cms/doi/data/fileA",
               "fileB"};
```

where `fileA` is located on the GridFTP server and `fileB` in the current directory on the UI.

It is also possible to specify a base GridFTP URI with the attribute `InputSandboxBaseURI`: in this case, files expressed as simple file names or as relative paths will be looked for under that base URI. Local files can still be defined using the `file://<path>` URI format. For example:

```
InputSandbox = {"fileA", "data/fileB", "file:///home/doi/fileC"};
InputSandboxBaseURI = "gsiftp://lxb0707.cern.ch/cms/doi";
```

is equivalent to

```
InputSandbox = {"gsiftp://lxb0707.cern.ch/cms/doi/fileA",
               "gsiftp://lxb0707.cern.ch/cms/doi/data/fileB",
               "/home/doi/fileC"};
```

Example 6.3.3.2 (Storing output files in a GridFTP server)

In order to store the output sandbox files to a GridFTP server, the `OutputSandboxDestURI` attribute must be used together with the usual `OutputSandbox` attribute. The latter is used to list the output files created by the job in the WN to be transferred, and the former is used to express where the output files are to be transferred. For example:

```
OutputSandbox = {"fileA", "data/fileB", "fileC"};
OutputSandboxDestURI = {"gsiftp://lxb0707.cern.ch/cms/doi/fileA",
                       "gsiftp://lxb0707.cern.ch/cms/doi/fileB",
                       "fileC"};
```

where the first two files have to be copied to a GridFTP server, while the third file will be copied back to the WMS with the usual mechanism. Clearly, `glite-wms-job-output` will retrieve only the third file.

Another possibility is to use the `OutputSandboxBaseDestURI` attribute to specify a base URI on a GridFTP server where the files listed in `OutputSandbox` will be copied. For example:

```
OutputSandbox = {"fileA", "fileB"};
OutputSandboxBaseDestURI = "gsiftp://lxb0707.cern.ch/cms/doi/";
```

will copy both files under the specified GridFTP URI.

Note: the directory on the GridFTP where the files have to be copied must already exist.

6.3.4. Real Time Output Retrieval

It is possible to see the files produced by a job while it is still running by using the *Job Perusal* functionality, only available via gLite WMS.

Example 6.3.4.1 (Inspecting the job output in real time with the gLite WMS)

The user can enable the job perusal by setting the attribute `PerusalFileEnable` to `true` in the job JDL. This makes the WN to upload, at regular time intervals (defined by the `PerusalTimeInterval` attribute and expressed in seconds), a copy of the output files specified using the `glite-wms-job-perusal` command to the WMS machine (by default), or to a GridFTP server specified by the attribute `PerusalFilesDestURI`.

The following example shows how to use the job perusal. The JDL file should look like this:

```

Executable = "job.sh";
StdOutput = "stdout.log";
StdError = "stderr.log";
InputSandbox = {"job.sh"};
OutputSandbox = {"stdout.log", "stderr.log", "testfile.txt"};
PerusalFileEnable = true;
PerusalTimeInterval = 30;
RetryCount = 0;
  
```

After the job has been submitted with `glite-wms-job-submit`, the user can choose which output files should be inspected:

```

$ glite-wms-job-perusal --set -f stdout.log -f stderr.log -f testfile.txt \
https://wms104.cern.ch:9000/B02xR3EQg9ZHHRc-1nJkQ

Connecting to the service https://128.142.160.93:7443/glite_wms_wmproxy_server

Connecting to the service https://128.142.160.93:7443/glite_wms_wmproxy_server

===== glite-wms-job-perusal Success =====

Files perusal has been successfully enabled for the job:
https://wms104.cern.ch:9000/B02xR3EQg9ZHHRc-1nJkQ

=====
  
```

and, when the job starts, the user can see one output file:

```

$ glite-wms-job-perusal --get -f testfile.txt \
https://wms104.cern.ch:9000/B02xR3EQg9ZHHRc-1nJkQ
  
```

Connecting to the service https://137.138.45.79:7443/glite_wms_wmproxy_server

Connecting to the service https://137.138.45.79:7443/glite_wms_wmproxy_server

===== glite-wms-job-perusal Success =====

The retrieved files have been successfully stored in:
/tmp/does_OoDVmWCAnhx_HiSPvASGsg

=====

file 1/1: testfile.txt-20071220115405_1-20071220115405_1

This is a test file
Data : Wed Dec 20 11:53:37 CET 2007
Host : c01-017-103

Subsequent invocations of `glite-wms-job-perusal --get` will retrieve only the part of the file that was written after the previous invocation. To have the complete file, the `--all` option can be used. Only one file can be retrieved at a time. Finally, the job perusal can be disabled for all jobs using the `--unset` option.

6.3.5. The BrokerInfo

The *BrokerInfo file* is a mechanism to access, at job 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 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 using the `-r <CEId>` option of the job submission commands prevents the creation of the BrokerInfo file.

The command to print the information in the BrokerInfo file is `glite-brokerinfo`.

A detailed description of this command and of the corresponding API can be found in [39].

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

```
glite-brokerinfo [-v] [-f filename] function [parameter] [parameter] ...
```

where `function` is one of the following:

- `getCE`: returns the name of the CE where the job is running;

- `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 SEs with contain a copy of at least one file among those specified in `InputData`;
- `getCloseSEs`: returns a list of the SEs close to the CE where the job is running;
- `getSEMountPoint <SE>`: returns the access point for the specified close SE;
- `getSEFreeSpace <SE>`: returns the free space on the SE;
- `getLFN2SFN <LFN>`: returns the SURL of the specified LFN, listed in the `InputData` attribute;
- `getSEProtocols <SE>`: returns the list of the protocols available to transfer data in the specified SE;
- `getSEPort <SE> <Protocol>`: returns the port number used by the SE for the specified data transfer protocol;
- `getVirtualOrganization`: returns the name of the VO specified in the JDL.

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

6.3.6. Direct Submission to CREAM CE

The CREAM [56] (Computing Resource Execution And Management) Service is a simple, lightweight service for job management operations at the Computing Element level. CREAM accepts job submission requests and other job management requests both through the Workload Management System, via the ICE (Interface to Cream Environment) component, and through a generic client for directly submitting jobs to a CREAM CE. The submission to a CREAM based CE through the WMS is completely transparent to the user. In this subsection we give a brief review of the most relevant commands of the CLI for direct submission to CREAM.

Man pages are available for all the CREAM client commands. You can also access information about the usage of each command through the `--help` option. For a more comprehensive discussion, please refer to the CREAM User's Guide [57]. The most relevant commands to interact with CREAM based CEs are very similar to their LCG analogous.

Job Submission Being analogous to `glite-wms-job-submit`, the following command

```
glite-ce-job-submit -r <CEId> [options] <jdl_file>
```

submits a job to a CREAM based CE. It requires a JDL file and the `<CEId>` as input and returns a CREAM job identifier. Any gLite WMS JDL file is accepted (a complete reference for the specification of the JDL attributes supported by the CREAM CE service can be found in [58]). The `--resource (-r)` directive must be used to target the job submission to a specific CREAM CE, identified by its identifier `CEId`. The standard format for a CREAM `CEId` is:

```
<full-hostname>:<port-number>/cream-<service>--<queue-name>
```

where `<service>` identifies the underlying resource management system (e.g. `lsf`, `pbs`, etc.). An example of CREAM `CEId` is:

```
prod-ce-02.pd.infn.it:8443/cream-lsf-creamusr2
```

Upon successful submission, the command returns to the user the submitted CREAM job identifier (jobId) which is the analogous of the gLite WMS job identifier (see Section 6.3.1). Also the format of the CREAM jobId is similar and it is as follows:

```
https://<CREAM_full_hostname>:<port>/CREAM<unique_string>
```

Just like for the gLite WMS submission, it is possible to redirect the returned jobId to an output file using the `--output (-o)` option. Just like for the gLite WMS submission, it accepts the `-a` option for automatic delegation, or the `-d <DelegID>` option for named explicitly delegated credential (see next command right below).

Delegation Being analogous to `glite-wms-job-delegate-proxy`, the following command

```
glite-ce-delegate-proxy --endpoint <host>[:<port>] <DelegationId>
```

allows the user to delegate his proxy credential to the CREAM service. This delegated credential can then be used for job submissions. Mandatory inputs are the endpoint and the delegation ID string. The endpoint must be specified using the `--endpoint` or `-e` option followed by the full hostname and the port number of the considered CREAM CE service (not the full CREAM CEId). The delegation ID string, chosen by the user, will then have to be specified in the following submission operations (with the option `--delegationId` or `-d` of the `glite-ce-job-submit` command). An example of proxy delegation is:

```
glite-ce-delegate-proxy -e prod-ce-02.pd.infn.it:8443 myproxyid1
```

Job Status The command

```
glite-ce-job-status [options] <CREAMjobId1> <CREAMjobId2> ... <CREAMjobIdN>
```

displays information (in particular the states) of jobs previously submitted to CREAM based CEs. The verbosity level is selected through the `--level <verb_level> (-L <verb_level>)`; possible values for `verb_level` are 0, 1 and 2. The `--endpoint host[:port]` (or `-e host[:port]`) selects the endpoint of the CREAM CE to send the status request to. If this option is not specified, the endpoint will be extracted from the CREAMjobId strings. The endpoint must have the format `<host>:<port>` or `<host>`. The `--all (-a)` option displays information about all known jobs owned by the user issuing the command which are present in the specified CREAM CE. The CREAM CE must be specified through the `--endpoint host[:port]` (or `-e host[:port]`) option, which is mandatory in this case. Examples of job status query are the following:

```
glite-ce-job-status -L 1 https://cream-02.pd.infn.it:8443/CREAM955790315
```

```
glite-ce-job-status --endpoint grid005.pd.infn.it:8443 --all
```

Some other options of this command are just the same of the corresponding gLite WMS command, like the `--input (-i)` or the `--status (-s)` options.

Job Cancellation The command

```
glite-ce-job-cancel [options] <CREAMjobId1> <CREAMjobId2> ... <CREAMjobIdN>
```

cancels one or more jobs previously submitted to CREAM based CEs. The `--all`, `--endpoint`, `--input` options have just the same behavior they have for the `glite-ce-job-status` command. Examples of job cancel query are the following:

```
glite-ce-job-cancel --input joblist.txt

glite-ce-job-cancel --endpoint grid005.pd.infn.it:8443 --all
```

Job List The following command

```
glite-ce-job-list [options] <host>[:<port>]
```

lists the identifiers of jobs submitted by the user issuing the command which have been submitted to the CREAM based CE whose host and eventually port number are specified as input of the command. These returned CREAM job identifiers can be later used as a handle to perform monitor and control operations on the jobs (e.g. see `glite-ce-job-status`, described above). It is possible to redirect the returned CREAM jobIds into an output file using the `--output (-o)` option.

6.4. ADVANCED JOB TYPES

This section describes how to use more advanced job types.

6.4.1. Job Collections

One of the most useful functionalities of WMPProxy is the ability to submit job *collections*, defined as sets of independent of jobs. This greatly speeds up the job submission time, compared to the submission of individual jobs, and together with the proxy delegation mechanisms, it saves a lot of processing time by reusing the same authentication for all the jobs in the collection.

Example 6.4.1.1 (Submitting a job collection)

The simplest way to submit a collection is to put the JDL files of all the jobs in the collection in a single directory, and use the `--collection <dirname>`, where `<dirname>` is the name of that directory. For example, suppose that there are two JDL files in the `jdl` directory

```
$ ls -l jdl/
job1.jdl job2.jdl
```

We can submit both jobs at the same time by doing:

```
$ glite-wms-job-submit -a --collection jdl
```

Connecting to the service `https://wms104.cern.ch:7443/glite_wms_wmproxy_server`

```
===== glite-wms-job-submit Success =====
```

The job has been successfully submitted to the WMPProxy
Your job identifier is:

```
https://wms104.cern.ch:9000/n1JoZ8WbyJBrW3-pTU3f4A
```

```
=====
```

The jobID returned refers to the collection itself. To know the status of the collection and of all the jobs belonging to it, it is enough to use `glite-wms-job-status` as for any other kind of job:

```
$ glite-wms-job-status https://wms104.cern.ch:9000/n1JoZ8WbyJBrW3-pTU3f4A
```

```
*****
```

```
BOOKKEEPING INFORMATION:
```

```

Status info for the Job : https://wms104.cern.ch:9000/n1JoZ8WbyJBrW3-pTU3f4A
Current Status:      Done (Exit Code !=0)
Exit code:          1
Status Reason:      Warning: job exit code != 0
Destination:        dagman
Submitted:          Thu Dec 14 18:26:42 2007 CET

```

```
*****
```

```
- Nodes information:
```

```

Status info for the Job : https://wms104.cern.ch:9000/1SUgblV08Ge3OnIW07FAYw
Node Name:          job1_jdl
Current Status:     Done (Success)
Exit code:          0
Status Reason:      Job terminated successfully
Destination:        arxiloxos1.inp.demokritos.gr:2119/jobmanager-lcgpbs-cms
Submitted:          Thu Dec 14 18:26:42 2007 CET

```

```
*****
```

```

Status info for the Job : https://wms104.cern.ch:9000/_3Svxr0Lsg5L5QVZMmzTrg
Node Name:          job2_jdl
Current Status:     Aborted
Status Reason:      hit job shallow retry count (0)
Destination:        ce-lcg.sdg.ac.cn:2119/jobmanager-lcgpbs-cms
Submitted:          Thu Dec 14 18:26:42 2007 CET

```

```
*****
```

In this example, one job succeeded and one failed, which explains why the status of the collection itself reports and exit code different from zero.

Note: executing `glite-wms-job-status` for the collection is the only way to know the jobIDs of the jobs in the collection.

The behaviour of the other job management commands is as follows:

- `glite-wms-job-output <collID>` retrieves the output of all the jobs in the collection `<collID>` which finished correctly;
- `glite-wms-job-cancel <collID>` cancels all the jobs in the collection;
- `glite-wms-job-logging-info <collID>` returns the logging information for the collection, but not for the jobs which compose it.

Once the jobIDs of the single jobs are known, the job management commands can be used with them exactly as for any other job.

Example 6.4.1.2 (Advanced collections)

A more flexible way to define a job collection is illustrated in the following JDL file. Its structure includes a global set of attributes, which are inherited by all the sub-jobs, and a set of attributes for each sub-job, which supersede the global ones.

```
[
  Type = "Collection";
  VirtualOrganisation = "cms";
  MyProxyServer = "myproxy.cern.ch";
  InputSandbox = {"myjob.exe", "fileA"};
  OutputSandboxBaseDestURI = "gsiftp://lxb0707.cern.ch/data/does";
  DefaultNodeShallowRetryCount = 5;
  Requirements = Member("VO-cms-CMSSW_2_0_0",
    other.GlueHostApplicationSoftwareRunTimeEnvironment");
  Nodes = {
    [
      Executable = "myjob.exe";
      InputSandbox = {root.InputSandbox,
        "fileB"};
      OutputSandbox = {"myoutput1.txt"};
      Requirements = other.GlueCEPolicyMaxWallClockTime > 1440;
    ],
    [
      NodeName = "mysubjob";
      Executable = "myjob.exe";
      OutputSandbox = {"myoutput2.txt"};
      ShallowRetryCount = 3;
    ],
    [
      File = "/home/does/test.jdl";
    ]
  }
]
```

The interpretation of this JDL file is as follows:

- it describes a collection (Type = "Collection");
- the jobs belong to the *cms* VO;
- the Myproxy server to use for proxy renewal is `myproxy.cern.ch`;
- all the jobs in the collection have by default the executable `myjob.exe` and the file `fileA` in their sandbox (*shared input sandbox*);
- all the output files must be copied to a GridFTP server;
- the default maximum number of shallow resubmissions is 5;
- all the jobs must run on CEs with a given version of a software (CMSSW_2_0_0);
- the input sandbox of the first job (or node) has all the default files (`root.InputSandbox`), plus an additional file, `fileB`, while the second job has only the default files;
- the first job must run on a CE allowing at least one day of wallclock time;
- the second job has a limit of 3 for the number of shallow resubmissions;
- the third job is described by another JDL file, `/home/does/test.jdl`;
- the three jobs have names `node0`, `mysubjob` and `node2`.

The biggest advantage of this way to build a collection is the possibility to specify a shared input sandbox when the jobs have one or more in the input sandbox which are the same for each job.

A full description of the JDL syntax for collections is available at [38].

6.4.2. DAG jobs

The gLite WMS provides an implementation of for *direct acyclic graphs (DAG)*, which are sets of jobs linked by relative dependencies. A job A is said to depend on job B if A is not allowed to run before the job B is successfully completed. A complete description of the JDL syntax for DAGs will not be given here, but it is available elsewhere [38].

6.4.3. Parametric jobs

A *parametric job* is a job collection where the jobs are identical but for the value of a running parameter. it is described by a single JDL, where attribute values may contain the current value of the running parameter. An example of a JDL for a parametric job follows:

```
[
JobType = "Parametric";
Executable = "myjob.exe";
StdInput = "input_PARAM_.txt";
StdOutput = "output_PARAM_.txt";
StdError = "error_PARAM_.txt";
Parameters = 100;
ParameterStart = 1;
ParameterStep = 1;
```

```

InputSandbox = {"myjob.exe", "input_PARAM_.txt";
OutputSandbox = {"output_PARAM_.txt", "error_PARAM_.txt"};
]

```

The attribute `Parameters` can be either a number, or a list of items (typically strings, but not enclosed within double quotes): in the first case, the value represent the maximum value of the running parameter `_PARAM_`; in the second case, it is the list of the values the parameter must take.

The attribute `ParameterStart` is the initial number of the running parameter, and the attribute `ParameterStep` is the increment of the running parameter between consecutive jobs. Both attributes can be set only if `Parameters` is a number.

6.4.4. MPI Jobs

The *Message Passing Interface (MPI)* is a commonly used standard library for parallel programming. The gLite WMS natively supports the submission of MPI jobs, which are jobs composed of a number of processes running on different WNs in the same CE. However, this support is still experimental and this functionality will not be described in this User Guide. The most complete source of information regarding MPI jobs on the Grid is currently the MPI Wiki page [40].

6.5. COMMAND LINE INTERFACE CONFIGURATION

The command line interface of the WMS can be configured using appropriate configuration files. In this section it is explained how to use and customize these configuration files.

6.5.1. WMPProxy Configuration

The WMPProxy commands (`glite-wms-*`) look for configuration files in these locations, in order of precedence:

- a. the file specified by the `--config` option;
- b. the file pointed by the `$GLITE_WMS_CLIENT_CONFIG` environment variable;
- c. the file `$HOME/.glite/<vo>/glite_wms.conf`, where `<vo>` is the user's VO name in lowercase;
- d. the file `$GLITE_LOCATION/etc/<vo>/glite_wms.conf`;
- e. the file `$GLITE_LOCATION/etc/glite_wms.conf`.

The settings in files with higher precedence supersede settings in files with lower precedence.

A typical configuration file looks as follows:

```

[
  WmsClient = [
    VirtualOrganisation = "cms";

```

```

    Requirements = other.GlueCEStateStatus == "Production";
    Rank = - other.GlueCEStateEstimatedResponseTime;
    MyProxyServer = "myproxy.cern.ch";
    WMPProxyEndpoints = {
        "https://wms104.cern.ch:7443/glite_wms_wmproxy_server"
    };
    OutputStorage = "/tmp";
    ErrorStorage = "/tmp";
    ListenerStorage = "/tmp";
    AllowZippedISB = true;
    PerusalFileEnable = false;
    RetryCount = 0;
    ShallowRetryCount = 3;
  ];
]

```

In this example file, the following properties are configured:

- the default VO;
- the default requirements;
- the default rank;
- the default MyProxy server;
- the endpoints of the WMPProxy servers to be used to submit jobs;
- the path where to store the job output files;
- the path where to write log files;
- the path where to create listener input/output pipes for interactive jobs;
- whether the input sandbox should be zipped by default before being uploaded to the WMPProxy server;
- whether the job file perusal support should be enabled by default;
- the default maximum number of deep resubmissions;
- the default maximum number of shallow resubmissions.

Example 6.5.1.1 (Using several WMS)

It is possible to configure the WMS CLI to use several WMS instances for improved redundancy. A job submission command will pick a WMS at random from a list and, if it is unavailable, it will pick another until it succeeds submitting a job or exhausts the list of WMS.

A list of WMS can be specified by defining a list as value for `WMPProxyEndpoints`:

```

WMPProxyEndpoints = {
    "https://wms104.cern.ch:7443/glite_wms_wmproxy_server",
    "https://wms105.cern.ch:7443/glite_wms_wmproxy_server",
    "https://wms106.cern.ch:7443/glite_wms_wmproxy_server"
};

```

Example 6.5.1.2 (Using a separate LB server with the gLite WMS)

Normally, an LB server is installed on the same host as the WMS. However, when the WMS is very busy with managing a large number of jobs, the WMS services and the LB server might experience a performance degradation. In this case, it is advisable to configure the CLI in order to use an LB server on a separate machine. This is done using in the JDL the `LBAddress` attribute, whose value is a string representing the address (`<host>[:<port>]`) of the LB server.

If the user is submitting a job collection, the `LBAddress` attribute must be put in the common part of the collection JDL, because it must be the same for all the jobs in the collection. For example:

```
[
  Type = "Collection";
  LBAddress = "lxb7026.cern.ch:9000";
  ...
  Nodes = {
    ...
  }
]
```

7. DATA MANAGEMENT

7.1. INTRODUCTION

This chapter describes Data Management clients and services available in gLite 3.1. An overview of the available Data Management APIs is also given in Appendix E.

7.2. STORAGE ELEMENTS

The *Storage Element* is the service which allows a user or an application to store data for future retrieval. All data in a SE must be considered *read-only* and therefore can not be changed unless physically removed and replaced. Different VOs might enforce different policies for space quota management. Contact your VO Data Management Administrator for details.

7.2.1. Data Channel Protocols

The data transfer and access protocols supported in gLite 3.1 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
file	File I/O	No	Remote file access	Yes

The *GSIFTP*^[41]² protocol offers the functionalities of FTP, but with support for GSI. It is responsible for secure file transfers to/from Storage Elements. It provides third party control of data transfer as well as parallel stream data transfer. Every WLCG/EGEE SE runs at least one GridFTP server. For direct remote access of files stored in the SEs, the protocols currently supported by gLite 3.1 are the *Remote File Input/Output protocol (RFIO)* [42] and the *GSI dCache Access Protocol (gsidcap)*. RFIO was developed to access tape archiving systems, such as CASTOR (CERN Advanced STORAge manager)[43] and it comes in a secure and an insecure version. More information about RFIO can be found in Appendix E. The *gsidcap* protocol is the GSI enabled version of the dCache[44] native access protocol, *dcap*. The *file* protocol is used for local file access to network filesystems.

7.2.2. Types of Storage Elements

In WLCG/EGEE, different types of Storage Elements are in use:

- **CASTOR**: it consists of a disk buffer frontend to a tape mass storage system. A virtual filesystem (namespace) shields the user from the complexities of the disk and tape underlying setup. File migration between

²In the literature, the terms *GridFTP* and *GSIFTP* are sometimes used interchangeably. Strictly speaking, *GSIFTP* is a subset of *GridFTP*.

disk and tape is managed by a service called “stager”. The native storage protocol, the insecure RFIO, allows access of files in the SE. Since the protocol is not GSI-enabled, only RFIO access from a location in the same LAN of the SE is allowed. With the proper modifications, the CASTOR disk buffer can be used also as disk-only storage system;

- **dCache**: it consists of a 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. The native gsidcap protocol allows POSIX-like data access. dCache is widely employed as disk buffer frontend to many mass storage systems, like HPSS and Enstore, as well as a disk-only storage system.
- **LCG Disk pool manager (DPM)**: is a lightweight disk pool manager. Disks can be added dynamically to the pool. Like in dCache and CASTOR, a virtual filesystem hides the complexity of the disk pool architecture. The secure RFIO protocol allows file access from the WAN;
- **StoRM**: a SE implementation providing a Grid interface to a storage system based on GPFS, a proprietary distributed file system from IBM;
- **BestMAN**: like StoRM, it provides a Grid interface to any disk-based file system;
- **Classic SE**: it consists of a GridFTP server and an insecure RFIO daemon in front of a physical single disk or disk array. This type of SE is not supported anymore.

7.2.3. The Storage Resource Manager interface

The Storage Resource Manager (SRM) has been designed to be the single interface (through the corresponding SRM protocol) for the management of disk and tape storage resources. Any type of Storage Element in WLCG/EGEE offers an SRM interface except for the Classic SE, which was phased out. SRM hides the complexity of the resources setup behind it and allows the user to request files, keep them on a disk buffer for a specified lifetime, reserve in advance space for new files, and so on. SRM offers also a third party transfer protocol between different endpoints, not supported however by all SE implementations. It is important to notice that the SRM protocol is a storage management protocol and not a file access protocol.

7.3. FILE NAMES IN GLITE 3.1

As an extension of what was introduced in Chapter 3, the different types of file names that can be used within the gLite 3.1 file catalogue are summarized below:

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

```

guid:<36_bytes_unique_string>
guid:38ed3f60-c402-11d7-a6b0-f53ee5a37e1d
  
```

- the **Logical File Name (LFN)**, which is a human-readable name for a file, has this format:

```

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

In the case of the LCG File Catalogue (see Section 7.4), the LFNs are organized in a hierarchical directory-like structure and have the following format:

```

lfn:/grid/<vo>/<directory>/<file>
  
```

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

```
srm://<SE_hostname>:<port>/<some_string>
```

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 host name. 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://srm.cern.ch:8443/srm/managerv2?SFN=castor/cern.ch/dteam/file1
```

- 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/data/dteam/does/file1
```

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 immutable, TURLs are obtained dynamically from the SURL through the SRM interface. 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 CATALOGUE IN GLITE 3.1

Users and applications need to locate files (or replicas) on the Grid. The *File Catalogue* is the service which maintains mappings between LFN(s), GUID and SURL(s). The *LCG File Catalogue (LFC)* is the File Catalogue adopted by gLite 3.1, but other file catalogues are in use.

The catalogue publishes its endpoint (service URL) in the Information Service so that it can be discovered by Data Management tools and other services. LFC could either be used as a Local File Catalogue, holding only replicas stored at a given group of site, or as a Global File Catalogue, containing information about all files in the Grid. This last one can have multiple read-only instances de-localized at main computing centres all holding the same information.

LFC was developed to overcome serious performance and security issues of the old EDG-RLS catalogue [55]; it also adds some new functionalities such as transactions, sessions, bulk queries and a hierarchical namespace for LFNs. It consists of a unique catalogue, where the LFN is the main key (Figure 9). Further LFNs can be added as symlinks to the main LFN. System metadata are supported, while for user metadata only a single string entry is available (rather a description field than real metadata).

Note: a file is considered to be a *Grid file* if it is **both** physically present in a SE **and** registered in a file catalogue. In this chapter several tools will be described. In general high level tools like *lcg_util* (see Sec. 7.5.1) will ensure consistency between files in the SEs and entries in the file catalogue. However, usage of low level Data Management tools, loss of files in a SE, data corruption in a file catalogue, could create inconsistencies between SEs physical files and catalogue entries. This is why the usage of low level tools is strongly discouraged unless really necessary.

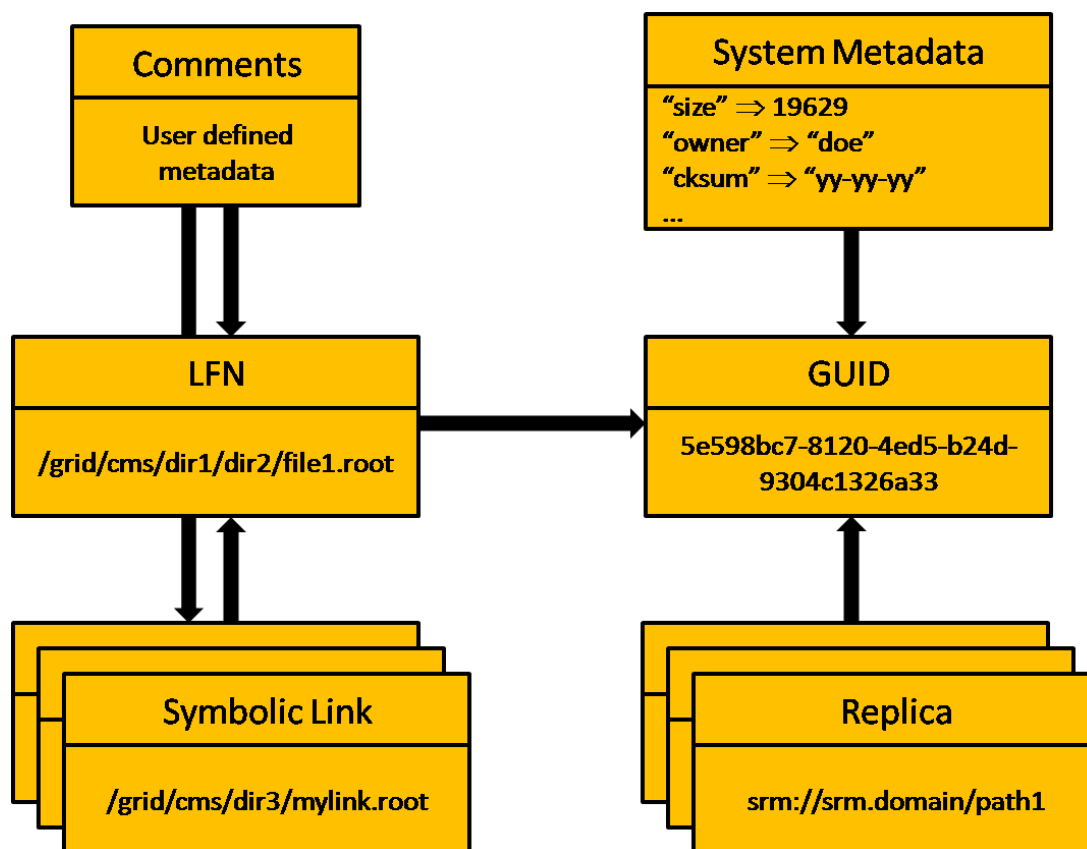


Figure 9: Architecture of the LFC

7.4.1. LFC Commands

In general the user should interact with the file catalogue through high level utilities (*lcg_util*, see Section 7.5.1). The CLIs and APIs that are available for catalogue interaction provide further functionality and more fine-grained control for catalogue operations; in some situations, they represent the only possible way to achieve the desired result.

In gLite 3.1 the environment variable `LFC_HOST` can be set to hold the host name of the LFC server. This is mandatory for the LFC CLIs and APIs; for GFAL and *lcg_util* (see later) such variable, if set, supersedes the endpoint definition published in the Information System.

The directory structure of the LFC namespace has the form:

```
/grid/<VO>/<filepath>
```

Users of a given VO will have read and write permissions only under the corresponding <VO> subdirectory. More restrictive access patterns on deeper subpaths of the directory tree can be enforced by the 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-chgrp</code>	Set group ownership of a LFC file/directory
<code>lfc-setcomment</code>	Add/replace a comment
<code>lfc-delcomment</code>	Delete the comment associated with a file/directory
<code>lfc-setacl</code>	Set file/directory access control lists
<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-rm</code>	Remove a file/directory
<code>lfc-mkdir</code>	Create directory
<code>lfc-rename</code>	Rename a file/directory
<code>lfc-enterusrmap</code>	Defines a new user entry in the Virtual ID table
<code>lfc-entergrpmap</code>	Defines a new group entry in the Virtual ID table
<code>lfc-listusrmap</code>	List users in the Virtual ID table
<code>lfc-listgrpmap</code>	List groups in the Virtual ID table
<code>lfc-modifyusrmap</code>	Modify a user entry corresponding to a given virtual uid
<code>lfc-modifygrpmap</code>	Modify a group entry corresponding to a given virtual gid
<code>lfc-rmusrmap</code>	Suppress user entry corresponding to a given virtual uid or user name
<code>lfc-rmgrpmap</code>	Suppress group entry corresponding to a given virtual gid or group name
<code>lfc-ping</code>	Check if the name server is alive.

Man pages are available for all the commands. Most of them work in a very similar way to their Unix equivalents, but operating on directories and files of the catalogue 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.

Users should use these commands carefully, keeping in mind that the operations they are performing affect the catalogue, but not the physical files that the entries represent.

Example 7.4.1.1 (*Listing the entries of a LFC directory*)

The `lfc-ls` command lists the LFNs in a given directory.

Attention! The `-R` option, for recursive listing, is available for the command, but **it should not be used extensively**. It is a very expensive operation on the catalogue and should be avoided as much as possible.

In the following example content of the directory `/grid/dteam/MyExample` is listed:

```

$ lfc-ls /grid/dteam/MyExample

/grid/dteam/MyExample:
day1
day2
day3
day4

```

Example 7.4.1.2 (*Creating directories in the LFC*)

The `lfc-mkdir` creates a directory in the LFN namespace:

```

$ lfc-mkdir /grid/lhcb/test_doe/MyTest
$ lfc-ls -l /grid/lhcb/test_doe
drwxrwxr-x  0 doe z5                0 Feb 21 16:50 MyTest
  
```

Example 7.4.1.3 (Creation of symbolic links)

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

In the following example, we create a symbolic link `/grid/lhcb/test_doe/MyTest/newname` to the original file `/grid/lhcb/test_doe/testfile`:

```

$ lfc-ln -s /grid/lhcb/test_doe/testfile /grid/lhcb/test_doe/MyTest/newname
  
```

And check that the new alias exists:

```

$ lfc-ls -l /grid/lhcb/test_doe/MyTest/newname
lrwxrwxrwx  1 doe  z5    0 Feb 21 16:54 /grid/lhcb/test_doe/MyTest/newname
-> /grid/lhcb/test_doe/testfile
  
```

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 will still exist as broken links.

Example 7.4.1.4 (Adding metadata information to LFC entries)

The `lfc-setcomment` and `lfc-delcomment` commands allow the user to associate a comment with a catalogue entry and delete such comment. This is the only user-defined metadata that can be associated with catalogue 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/cms/MyFiles/file1 "Most promising measure"

$ lfc-ls --comment /grid/cms/MyFiles/file1
/grid/dteam/MyFiles/file1 Most promising measure
  
```

Example 7.4.1.5 (Removing LFNs from the LFC)

The `lfc-rm` command can be used to remove files and directories from the LFN namespace, but with two basic limitations:

- a file can be removed only if there are no SURLS associated to it. If SURLS exist, the `lcg_util` commands should be used instead (Section 7.5.1);
- a directory can be removed (`-r` option) only if it is empty.

In the next example, the directory `trash` is removed:

```

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

$ lfc-rm -r /grid/dteam/MyExample/trash

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

```

7.4.2. Access Control Lists

LFC allows to attach to a file or directory an *access control list (ACL)*, a list of permissions which specify who is allowed to access or modify it. The permissions are very much like those of a UNIX file system: read (`r`), write (`w`) and execute (`x`). A combination of these permissions can be associated to these entities:

- a user (`user`);
- a group of users (`group`);
- any other user (`other`);
- the maximum permissions granted to specific users or groups (`mask`).

Permissions for multiple users and groups can be defined. If this is the case, a mask must be defined and the “effective” permissions are the logical AND of the user or group permissions and the mask.

In LFC, users and groups are internally identified as numerical *virtual uids* and *virtual gids*, which are virtual in the sense that they exist only in the LFC namespace.

A user can be specified as a name, as a virtual uid or as a Distinguished Name. A group can be specified as a name, as a virtual gid or as a VOMS FQAN.

In addition, a directory in LFC has also a *default ACL*, which is the ACL associated to any file or directory being created under that directory. After creation, the ACLs can be freely changed. When creating a sub-directory, its default ACL is inherited from the parent directory’s default ACL. A user can be in more than one group. For example, it might be possible for a user to be allowed to delete an LFC file when his VOMS proxy has a FQAN `/atlas` and also when he has a FQAN `/atlas/Role=production`, as expected.

Example 7.4.2.1 (Print the ACL of a directory)

In the following example, the ACL for a given directory is displayed:

```

$ lfc-getacl /grid/atlas/UserGuide

# file: /grid/atlas/UserGuide
# owner: /C=CH/O=CERN/OU=GRID/CN=John Doe
# group: atlas
user::rwx
group::rwx          #effective:rwx
other::r-x
default:user::rwx
default:group::rwx
default:other::r-x
  
```

The output prints the DN and the group of the owner of the directory, followed by the ACL and the default ACL. In this example, the owner and all users in the `atlas` group have full privileges to the directory, while other users cannot write into it.

Example 7.4.2.2 (Modify the ACL of a directory)

Suppose that we want to associate a set of permissions to a given FQAN for the LFC directory from the previous example. This could be done by doing:

```

$ lfc-setacl -m g:/atlas/Role=production:rwx,m:rwx,d:g:/atlas/Role=production:rwx,d:m:rwx \
/grid/atlas/UserGuide
  
```

The `-m` option means that we are modifying the existing ACL.

The added ACL is specified as a comma-separated list of entries, where each entry is a colon-separated list of fields: an ACL type (`user`, `group`, `other`, `mask`, or these preceded by `default`), a user or group, and a permission set. Notice that ACL types can be abbreviated using their first letter.

In this example, we have set a permission set for a group (the `/atlas/Role=production` FQAN), the mask and the same for the default ACL.

If now we print again the ACL for the directory:

```

$ lfc-getacl /grid/atlas/UserGuide

# file: /grid/atlas/UserGuide
# owner: /C=CH/O=CERN/OU=GRID/CN=John Doe
# group: atlas
user::rwx
group::rwx          #effective:rwx
group:/atlas/Role=production:rwx      #effective:rwx
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:/atlas/Role=production:rwx
  
```

```

default:mask::rwx
default:other::r-x
  
```

we see now permissions for both the owner's group (atlas) and the /atlas/Role=production FQAN, and the same for the default ACL. The effective permissions for the owner's group and the VOMS FQAN take into account the mask.

Other options of `lfc-setacl` are `-d` to remove ACL entries, and `-s` to replace the complete set of ACL entries.

7.5. FILE AND REPLICA MANAGEMENT CLIENT TOOLS

The gLite 3.1 middleware offers a variety of data management client tools to upload/download files to/from the Grid, replicate data and interact with the file catalogues. Every user should deal with data management through the gLite Data Management tools (usually referred to as *lcg_util* or `lcg-*` commands). They provide a high level interface (both command line and APIs) to the basic DM functionality, hiding the complexities of catalogue and SEs interaction. Furthermore, such high level tools minimize the risk of grid files corruption.

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

7.5.1. LCG Data Management Client Tools

The LCG Data Management tools (*lcg_util*) allow users to copy files between UI, CE, WN and a SE, to register entries in the file catalogue and replicate files between SEs. The name and functionality overview of the available commands is shown in the following table.

Replica Management

<code>lcg-cp</code>	Copies a file to/from a SE
<code>lcg-cr</code>	Copies a file to a SE and registers the file in the catalogue
<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 catalogue (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 Catalogue Interaction

<code>lcg-aa</code>	Adds an alias in the catalogue for a given GUID
<code>lcg-ra</code>	Removes an alias in the catalogue for a given GUID
<code>lcg-rf</code>	Registers in the catalogue a file residing on an SE
<code>lcg-uf</code>	Unregisters in the the catalogue 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
<code>lcg-ls</code>	Lists file information for given SURLs or LFNs

The `--vo <vo_name>` option, to specify the virtual organisation of the user, is present in all commands, except for `lcg-gt` and `lcg-sd`. Its usage is mandatory unless the variable `LCG_GFAL_VO` is set, see below. The `--config <file>` option (to specify a configuration file) and the `-i` option (to connect insecurely to the file catalogue) 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, that is no timeout. If a tool times out in the middle of an operation, all actions performed till that moment are rolled back, so no broken files are left on a SE and no existing files are not registered in the catalogues.

Environment variables

- For all `lcg-*` commands to work, the environment variable `LCG_GFAL_INFOSYS` must be set to point to a top BDII in the format `<hostname>:<port>`, or to a comma-separated list of top BDIIs in such format, so that the commands can retrieve the necessary information. Remember that the default BDII read port is 2170;
- the endpoint(s) for the catalogues can also be specified (taking precedence over that published in the IS) through the environment variable `LFC_HOST`. If no endpoints are specified, the ones published in the Information System are taken;
- if the variable `LCG_GFAL_VO` is set to indicate the user VO, the `--vo` option is not required. However, if the VO name is specified in the command option, the `LCG_GFAL_VO` variable is ignored;
- The `VO_<VO>_DEFAULT_SE` variable specifies the default SE for the VO `<VO>`.

The user must hold a valid proxy and be authorized on the SE in order to use `lcg-cr`, `lcg-cp`, `lcg-rep` and `lcg-del`. While access to resources (SEs and LFCs) is authenticated, the data channel is not crypted.

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, which are described in Chapter 5.

In what follows, some examples are given. Most commands can run in verbose mode (`-v` or `--verbose` option). For details on the options of each command, refer to the man pages of the commands.

Example 7.5.1.1 (Uploading a file to the Grid)

In order to upload a file to the Grid, that is to transfer it from the local machine to a Storage Element and register it in the catalogue, the `lcg-cr` command (which stands for *copy®ister*) can be used:

```
$ lcg-cr --vo dteam -d lxb0710.cern.ch file:/home/does/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 file GUID. If no destination is given, the SE specified by the `VO_<VO>_DEFAULT_SE` environmental variable is taken. Such variable is 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 published in the Information System. If no `-P` option is given, the relative path is automatically generated.

It is also possible 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/data/dteam/my_file
-d lxb0710.cern.ch -P my_dir/my_file
  
```

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

```

$ lcg-cr --vo dteam -d lxb0710.cern.ch -l lfn:my_alias1 file:/home/does/file1

guid:db7ddbc5-613e-423f-9501-3c0c00a0ae24
  
```

Note: LFNs in LFC are organized in a hierarchical namespace (like UNIX directory trees). So the LFN will take the form `lfn:/grid/<vo>/<dir1>/...`. Subdirectories in the namespace are **not** created automatically by `lcg-cr` and the user should manage himself their creation 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).

Attention! 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 7.5.1.2 (Replicating a file)

Once a file is stored on an SE and registered in the catalogue, 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/data/dteam/does/file1
File size: 30
Destination specified: lxb0707.cern.ch
Source URL for copy: gsiftp://lxb0710.cern.ch/data/dteam/does/file1
Destination URL for copy: gsiftp://lxb0707.cern.ch/data/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/data/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 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 7.5.1.3 (Listing replicas, GUIDs and aliases)

The `lcg-lr` (*list replicas*) command allows users to list all the replicas of a file registered in the file catalogue:

```

$ lcg-lr --vo dteam lfn:/grid/dteam/does/my_alias1

sfn://lxb0707.cern.ch/data/dteam/generated/2004-07-09/file79aee616-6cd7-4b75-8848-f091
sfn://lxb0710.cern.ch/data/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930
  
```

Again, a LFN, the GUID or a SURL can be used to specify the file. The SURLs of all 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/data/dteam/does/file1

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
  
```

Example 7.5.1.4 (Listing files and directories)

The `lcg-ls` command allows to list a file or a directory and its contents in the LFC namespace (if the argument is a LFN), or in the local storage namespace (if the argument is a SURL), very much like the `ls` command in UNIX.

For example, if the argument is a LFN:

```
$ lcg-ls -l lfn:/grid/dteam/does/myfile
-rw-rw-r-- 1 19692 1399 192 lfn:/grid/dteam/does/myfile
```

Another example involving the same file, but by SURL:

```
$ lcg-ls -l srm://cmsdcache.pi.infn.it/pnfs/pi.infn.it/data/cms/generated/2009-01-07/
fileacf27b40-ec72-424a-9ddc-4dcf757efb47
-rw-r--r-- 1 2 2 192 ONLINE /pnfs/pi.infn.it/data/cms/generated/
2009-01-07/fileacf27b40-ec72-424a-9ddc-4dcf757efb47
```

The options `-l` and `-d` work like for `ls`.

The type of SE is taken from the BDII by default, but the option `-b` can be used to avoid a call to the BDII; in this case, the SE type (`se` for the classic SE, `srmv1` or `srmv2`) must be specified using the `-T setype` option.

Example 7.5.1.5 (Copying files out of the Grid)

The `lcg-cp` command can be used to copy a Grid file to a local destination, or a local file to a SE without registering it in a file catalogue. The first argument is the source file and can be a LFN, GUID, SURL, a GSIFTP URL or a local file, and the second argument (destination file) must be a SURL, a GSIFTP URL or a local file. 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/does/myfile file:/tmp/myfile

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

Although `lcg-cp` is primarily intended to copy files from the Grid, it can be used as well to copy files to a SE, when registration in a file catalogue is not necessary.

Be aware that in the case of a MSS, the file may be not present on disk but only stored on tape. For this reason, `lcg-cp` on such a file could time out, waiting for the file stage-in on a disk buffer.

Example 7.5.1.6 (Obtaining a TURL for a replica)

The `lcg-gt` allows to get a TURL from a SURL and a supported protocol. The command behaves very differently if the Storage Element exposes an SRM interface or not. The command always returns three lines of output: the first is always the TURL of the file, the last two are meaningful only in case of SRM interface.

- For a classic SE (no SRM interface), the command obtains the TURL by simple string manipulation of the SURL 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/data/dteam/generated/2004-07-08/file0dcabb4
6-2214-4db8-9ee8-2930de1a6bef gsiftp

gsiftp://lxb0710.cern.ch/data/dteam/generated/2004-07-08/file0dcabb46-22
14-4db8-9ee8-2930de1a6bef
0
0
  
```

- In the case of a SRM interface, the TURL is returned to `lcg-gt` by the SRM itself. For a 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 start copying the file immediately. 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 with SRM 1). It is important to know that some SRM SEs 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://srm.cern.ch/castor/cern.ch/grid/dteam/generated/2005-04-12/file
fad1e7fb-9d83-4050-af51-4c9af7bb095c gsiftp

gsiftp://srm.cern.ch:2811//shift/lxfsrk4705/data02/cg/stage/filefad1e7fb-9d
83-4050-af51-4c9af7bb095c.43309
-337722383
0

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

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

Example 7.5.1.7 (Deleting replicas)

A file stored on a SE and registered in LFC 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/data/dteam/generated/2004-07-12/file78ef5a13-166f-4701-
8059-e70e397dd2ca
sfn://lxb0710.cern.ch/data/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/data/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 catalogue, as the last replica was deleted.

Example 7.5.1.8 (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 catalogue. 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/data/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1
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 catalogue:

```

$ lcg-uf --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b \
sfn://lxb0710.cern.ch/data/dteam/generated/2004-07-08/file0dcabb46-2214-4db8-9ee8-2930de1
  
```

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

Attention! `lcg-uf` just removes entries from the catalogue, it does not remove any physical replica from the SE. Watch out for consistency.

Example 7.5.1.9 (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:/grid/dteam/does/my_alias1

$ lcg-aa --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b lfn:/grid/dteam/does/new_alias

$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
lfn:/grid/dteam/does/my_alias1
lfn:/grid/dteam/does/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:/grid/dteam/does/my_alias1

$ lcg-la --vo dteam guid:baddb707-0cb5-4d9a-8141-a046659d243b
lfn:/grid/dteam/does/new_alias
  
```

7.6. FILE TRANSFER SERVICE

The *File Transfer Service (FTS)* is the gLite 3.1 low level data movement service. The user can schedule asynchronous and reliable file replication from source to destination (point-to-point, i.e. no file routing via intermediate storage) while participant sites can control the network usage. The FTS handles internally the SRM negotiation between the source and destination SEs and the management of the underlying GridFTP transfers.

7.6.1. Basic Concepts

- **Transfer Job:** a set of files to be transferred in a source/destination pair format. A job may contain optional parameters for the underlying transport layer (GridFTP). Finally, the job carries along a cyphered pass phrase to decrypt user credentials from the MyProxy server;
- **File:** a source/destination SURL pair to be transferred;
- **Job State:** a function of the individual file states constituting the Job;
- **File State:** the state of an individual file transfer;
- **Channel:** a specific network pipe used for file transfers. *Production channels* are high bandwidth, dedicated network pipe between Tier-0, Tier-1's and other major Tier-2's centers. *Non-production channels* are assigned typically to open networks and do not guarantee a minimum throughput as production channels do.

The transfer jobs are processed asynchronously (batch mode). Upon submission, a job identifier is returned to the user. This identifier can be used to query the status of the job as it progresses through the system or cancel the job. Once a job has been submitted to the system it is assigned to a transfer channel based on the SEs containing the source and the destination. Finally, FTS accepts only SURLS as source and destination. Logical entries like LFNs or GUIDs are at the moment not supported.

7.6.2. Transfer job states

The possible states a job can assume are:

- **Submitted:** the job has been submitted to FTS but not yet assigned to a channel
- **Pending:** the job has been assigned to a channel and files are waiting for being transferred
- **Active:** the transfer for some of the job's files is ongoing
- **Canceling:** the job is being canceled
- **Done:** all files in a job were successfully transferred
- **Failed:** some file transfers in a job have failed
- **Canceled:** the job has been canceled
- **Hold:** the job has aborted and requires manual interventions (moving it to **Pending** or **Failed**)

The final states for jobs are **Done**, **Canceled** and **Failed**. The possible job status transitions are depicted in Figure 10.

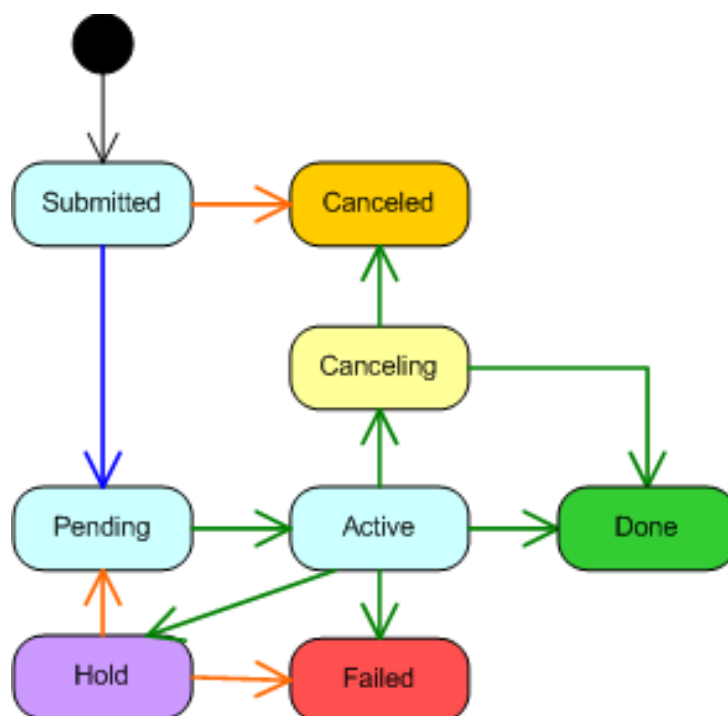


Figure 10: FTS transfer job states.

7.6.3. Individual file states

The possible states for individual files are the following:

- **Submitted**: the status of all files in a job which is in **Submitted** status
- **Pending**: the status of all files in a job which is in **Pending** status
- **Active**: the transfer of the file is ongoing
- **Canceling**: the transfer of the file is being canceled
- **Waiting**: the transfer of the file has failed; depending on the VO policies, it will then go to **Pending**, **Failed** or **Hold** status
- **Done**: the transfer of the file has finished correctly
- **Failed**: the transfer of the file has failed
- **Canceled**: the transfer of the file has been canceled
- **Hold**: the transfer of the file has failed and requires manual interventions (moving it to **Pending** or **Failed**)

The final states for files are **Done**, **Canceled** and **Failed**. The possible file status transitions are depicted in Figure 11.

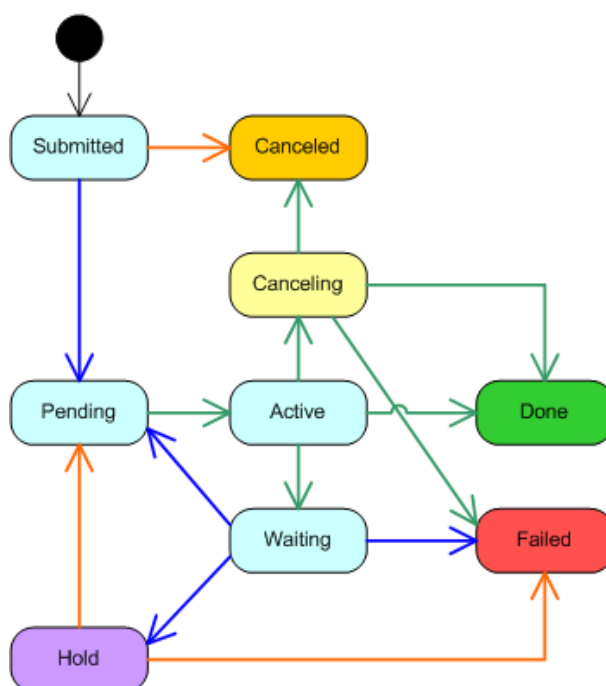


Figure 11: FTS individual file states.

7.6.4. FTS Commands

Before submitting a job, the user is expected to upload an appropriate password-protected long-term proxy to the MyProxy server used by FTS.

```
$ myproxy-init -s myproxy-fts.cern.ch -d
```

Attention! This is a different usage of MyProxy with respect to the WMS Proxy Renewal. In the latest case the `-n` option must be used while here it must be omitted. In addition, the same MyProxy server can not be *simultaneously* used for WMS Proxy Renewal and FTS authentication, because they require a different configuration of the MyProxy server.

The same password is passed to FTS at job submission time. The following user-level commands for submitting, querying and canceling jobs are described here:

<code>glite-transfer-submit</code>	Submits a transfer job
<code>glite-transfer-status</code>	Displays the status of an ongoing transfer job
<code>glite-transfer-list</code>	Lists all submitted transfer jobs owned by the user
<code>glite-transfer-cancel</code>	Cancels a transfer job

For completeness the following administrative commands are also briefly described. Only FTS service administrators are allowed to use them.

<code>glite-transfer-channel-add</code>	Creates a new channel with defined parameters on FTS
<code>glite-transfer-channel-list</code>	Displays details of a given channel defined on FTS
<code>glite-transfer-channel-set</code>	Allows administrators to set a channel Active or Inactive
<code>glite-transfer-channel-signal</code>	Changes status of all transfers in a given job or channel

Furthermore, here are the commands that have been added with FTS 2.0:

<code>glite-transfer-channel-audit</code>	Shows the audit log of changes made to a channel
<code>glite-transfer-channel-list -x</code>	Shows extra information about the channel
<code>glite-transfer-channel-set -m "Reason"</code>	Specify why a relevant channel change has been made
<code>glite-transfer-channel-setvolimit</code>	Allows setting of transfer caps for each VO
<code>glite-transfer-getroles -u</code>	Allows explicit asking for the privileges given to a specified user

Example 7.6.4.1 (Submitting a job to FTS)

Once a user has successfully registered a long-term proxy to a MyProxy server, he can submit a transfer job. He can do it either by specifying the source-destination pair in the command line:

```
$ glite-transfer-submit -m myproxy-fts.cern.ch \
-s https://fts.cnaf.infn.it:8443/sc3/glite-data-transfer-fts/services/FileTransfer \
srm://srm.sara.nl/pnfs/srm.sara.nl/data/lhcb/doi/zz_zz.f \
srm://srm.cnaf.infn.it/castor/cnaf.infn.it/grid/lcg/lhcb/test/SARA_1.25354
```

Enter MyProxy password:

Enter MyProxy password again:

```
c2e2cdb1-a145-11da-954d-944f2354a08b
```

or by specifying all source-destination pairs in an input file (bulk submission). The `-m` option specifies the MyProxy server to use; the `-s` option specifies the FTS service endpoint to be contacted. If the service starts with **http://**,

https:// or **http://** it is taken as a direct service endpoint URL; otherwise is taken as a service instance name and Service Discovery is invoked to look up the endpoints. If not specified the first available transfer service from the Service Discovery will be used. This is true for all subsequent examples.

```
$ glite-transfer-submit -m "myproxy-fts.cern.ch" \
-s https://fts.cr.cnaf.infn.it:8443/sc3/glite-data-transfer-fts/services/FileTransfer \
SARA-CNAF.in -p $passwd
```

where the input file `SARA-CNAF.in` looks like:

```
$ cat SARA-CNAF.in

srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb/test/doi/zz_zz.f \
srm://sc.cr.cnaf.infn.it/castor/cnaf.infn.it/grid/lcg/lhcb/test/doi/SARA_1.25354
srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb/test/doi/zz_zz.f \
srm://sc.cr.cnaf.infn.it/castor/cnaf.infn.it/grid/lcg/lhcb/test/doi/SARA_2.25354
srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb/test/doi/zz_zz.f \
srm://sc.cr.cnaf.infn.it/castor/cnaf.infn.it/grid/lcg/lhcb/test/doi/SARA_3.25354
.....
```

The `$passwd` in the example is an environment variable set to the value of the password to be passed to FTS.

Attention! The transfers handled by FTS within a single job bulk submission must be all assigned to the same channel, otherwise FTS will not process such transfers and will return the message: *Inconsistent channel*.

Example 7.6.4.2 *(Querying the status of a job)*

The following example shows a query to FTS to infer information about the state of a transfer job:

```
$ glite-transfer-status \
-s https://fts.cnaf.infn.it:8443/sc3/glite-data-transfer-fts/services/FileTransfer \
-l c2e2cdb1-a145-11da-954d-944f2354a08b
```

```
Pending
  Source:      srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lhcb/test/doi/zz_zz.f
  Destination: srm://sc.cr.cnaf.infn.it/castor/cnaf.infn.it/grid/lcg/lhcb/test/doi/
SARA_1.25354
  State:      Pending
  Retries:    0
  Reason:     (null)
  Duration:   0
```

Attention! The verbosity level of the status of a given job is set with the `-v` option; the status of individual files is however available through the option `-l`.

Example 7.6.4.3 (Listing ongoing data transfers)

The following example allows to query all ongoing data transfers in the specified (intermediate) state in a defined FTS service. In order to list only the transfer jobs relative to a channel, this must be specified with the `-c` option.

```

$ glite-transfer-list \
-s https://fts.cnaf.infn.it:8443/sc3/glite-data-transfer-fts/services/FileTransfer Pending
...
c2e2cdb1-a145-11da-954d-944f2354a08b Pending
...

```

Example 7.6.4.4 (Canceling a job)

An example of cancellation of a previously submitted data transfer job is shown here:

```

$ glite-transfer-cancel \
-s https://fts.cnaf.infn.it:8443/sc3/glite-data-transfer-fts/services/FileTransfer \
c2e2cdb1-a145-11da-954d-944f2354a08b

```

7.7. LOW LEVEL DATA MANAGEMENT TOOLS

In this section some details on lower level data management tools are given.

7.7.1. GSIFTP

The following low level tools can be used to interact with GSIFTP servers on SEs:

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

Attention! The commands `edg-gridftp-rename`, `edg-gridftp-rm`, and `edg-gridftp-rmdir` should be used with extreme care. In fact, these commands do not interact with any of the catalogues and therefore they can compromise the consistency/coherence of the information contained in the Grid.

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

Some examples are shown. To obtain help on these commands use the option `--usage` or `--help`. More information on the GSIFTP protocol is available in [41].

Example 7.7.1.1 (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 catalogues.

```

$ lcg-lr --vo dteam guid:27523374-6f60-44af-b311-baa3d29f841a

sfn://lxb0710.cern.ch/data/dteam/generated/2004-07-13/file42ff7086-8063-414d-9000-75c459b71296

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

$ edg-gridftp-exists \
gsiftp://lxb0710.cern.ch/data/dteam/generated/2004-07-13/my_fake_file

error gsiftp://lxb0710.cern.ch/data/dteam/generated/2004-07-13/my_fake_file
does not exist

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

/data/dteam/generated/2004-07-13/file42ff7086-8063-414d-9000-75c459b71296
  
```

Example 7.7.1.2 (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/data/dteam/generated/2004-07-13/file42ff7086-8063-414d-9000-75c459b71296 file://`pwd`/my_file
  
```

7.7.2. CASTOR and RFIO

Direct access to the CASTOR Mass Storage System (not via SRM) can be obtained through its native CLI. The clients are available in every gLite 3.1 UI or WN and described below, divided into two logical subgroups. Remember however that CASTOR supports insecure RFIO access only and therefore such commands must be used from a UI or WN in the same LAN of the CASTOR SE.

- Clients interacting with the CASTOR namespace:

- | | |
|-----------------------------|--|
| <code>ns ls</code> | list directories/files in CASTOR |
| <code>ns find</code> | search for files in CASTOR |
| <code>ns mkdir</code> | create a directory in CASTOR |
| <code>ns rm</code> | remove directories/files from CASTOR |
| <code>ns chmod</code> | change access mode of a directory/file in CASTOR |
| <code>ns chown</code> | change owner and group of a directory/file in CASTOR |
| <code>ns rename</code> | rename a directory/file in CASTOR |
| <code>ns ln</code> | create a link to a file in CASTOR |
| <code>ns touch</code> | change the filestamp of a file in CASTOR |
| <code>ns setcomment</code> | add/replace a comment associated with directory/file in CASTOR |
| <code>ns delcomment</code> | delete the comment associated with a directory/file in CASTOR |
| <code>ns setchecksum</code> | set or reset the checksum for a tape segment |
| <code>ns setacl</code> | set the ACL for a directory/file in CASTOR |
| <code>rfcp</code> | Remote file copy |
| <code>rfstat</code> | Display remote file or filesystem status |
| <code>rfcat</code> | Remote file concatenation to standard output |
- Clients managing CASTOR file classes:

<code>nschclass</code>	change the class of a CASTOR directory in the name server
<code>nsdeleteclass</code>	delete a file class definition
<code>nsenterclass</code>	define a new file class
<code>nslistclass</code>	query the CASTOR Name Server about a given class or list all existing classes
<code>nsmodifyclass</code>	modify an existing file class

File classes reflect into different service classes exposed to the user. For example, the CASTOR file class attribute defines whether a file is permanent or not: all permanent files belong to a class with an associated tape pool; all scratch files belong to a class with no associated tape pool. Setting the file class attributes requires administrator privileges.

7.7.3. dCache and DCAP

Analogously to CASTOR and the RFIO protocol, CLIs for direct access to dCache storage systems are available:

`dccp` allows to copy files from/to dCache SEs via the native dcap protocol

7.8. JOB SERVICES AND DATA MANAGEMENT

With both the LCG-2 and gLite 3.1 WMS, some specific JDL attributes allow the user to specify requirements on the input data (see Chapter 6 for information on how to define and manage Grid jobs).

Example 7.8.1 (Specifying input data in a job)

If a job requires one or more input files stored in a SE, the `InputData` JDL attribute can be used. Files can be specified by both by LFN and GUID.

An example of JDL specifying input data looks like:

```
Executable = "/bin/hostname";
```

```

StdOutput = "sim.out";
StdError = "sim.err";
DataCatalog = "http://lfc.cern.ch:8085";
InputData = {"lfn:/grid/dteam/does/fileA"};
DataAccessProtocol = {"rfio", "gsiftp", "gsidcap"};
OutputSandbox = {"sim.err", "sim.out"};
  
```

The `InputData` field may also be specified through GUIDs. This attribute is used only during the match-making process, to find an appropriate CE to run the job. It has nothing to do with the real access to files that the job can do while running. However, it is reasonable to list in `InputData` the files that will be accessed by the job and vice-versa.

The `DataAccessProtocol` attribute is used to specify the protocols that the application can use to access the file and is mandatory if `InputData` is present. Only data in SEs which support one or more of the listed protocols are considered. The WMS will schedule the job to a CE *close* 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.

Note: a SE or a list of SEs is published as “close” to a given CE via the `GlueCESEBindGroupSEUniqueID` attribute of the `GlueCESEBindGroup` object class. For more information see Appendix F.

Example 7.8.2 (Specifying a Storage Element)

With the LCG-2 WMS the user can ask the job to run close a specific SE using the attribute `OutputSE`. For example:

```
OutputSE = "srm.cern.ch";
```

The WMS will not submit the job if there is no CE close to the `OutputSE` specified by the user.

Note: the same is not true with the gLite 3.1 WMS. Even if there are CEs close to a given `OutputSE` specified by the user, no resources get matched when this field is defined on the JDL.

Example 7.8.3 (Automatic upload and registration of output files)

In the LCG-2 WMS (but not in the gLite 3.1 WMS), the `OutputData` attribute 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 not specified, 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 of JDL requiring explicitly an `OutputData` attribute:

```

Executable   = "test.sh";
StdOutput    = "std.out";
StdError     = "std.err";
InputSandbox = {"test.sh"};
OutputSandbox = {"std.out", "std.err"};
OutputData = {
  [
    OutputFile="my_file";
    LogicalFileName="lfn:/grid/dteam/doe/my_file";
    StorageElement = "castorsrm.pic.es";
  ]
};

```

Once the job is terminated and the user retrieves its output, the Output Sandbox downloaded will contain a further file, automatically generated by the JobWrapper, containing the logs of the output upload.

```

$ cat DSUpload_ZboHMYWoBsLVax-nUCmtaA.out
#
# Autogenerated by JobWrapper!
#
# The file contains the results of the upload and registration
# process in the following format:
# <outputfile> <lfn|guid|Error>

```

```
my_file    guid:2a14e544-1800-4257-afdd-7031a6892ef7
```

Example 7.8.4 (Selecting the file catalogue to use for match making)

For the WMS 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. To do this, the WMS uses the Data Location Interface service, which acts as interface to a file catalogue. Since it is possible that several different DLI services exist on the Grid, the user has the possibility to select which one he wants to talk to by using the JDL attribute `DataCatalog`. The user will specify the attribute and the endpoint of the DLI as value, as in this example:

```
DataCatalog = "http://lfc-lhcb-ro.cern.ch:8085/";
```

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

7.9. THE AMGA METADATA CATALOG

7.9.1. Introduction

AMGA is a gLite 3.1 metadata service which provides database access for Grid applications. AMGA is implemented as a thin layer between a Grid application and the underlying database and acts as an interface between them, providing a Grid style authentication mechanism. Another interesting feature of AMGA is the protocol used to transmit information, which reduces latency in WAN networks as much as possible and implements a data streaming mechanism which allows to retrieve information at high speed. AMGA also provides functionality to make Grid applications interoperable with different database backends. Finally, AMGA's support for replication of relational data can be used to build scalable and reliable Grid applications.

The AMGA project has been developed by the ARDA group at CERN and is now a collaborative effort of CERN, INFN Catania and Kisti in Korea.

This documentation is based on the document [59] provided by the developers and available on the AMGA web site: <http://amga.web.cern.ch/amga/>. Since AMGA is now delivered with the gLite 3.1 middleware, in the User Guide we will not give instructions about the installation. We will just mention that the AMGA server can be installed on the same server as the database or on another host, and has to be configured to point to the desired database schema, which can be located on another host anywhere in the network.

7.9.2. Configuration of the client

On the client side, the user can set the desired configuration in the `$HOME/.mdclient.config` file. If this file doesn't exist, the client will read the default configuration from the file `/opt/glite/etc/mdclient.config`.

An example configuration file is shown below:

```
# Connection options
Host = localhost
Port = 8822

# User settings
Login = root
Home = /

# Security options
UseSSL = no # Values: require, try, no. If off, all options below are ignored

AuthenticateWithCertificate = 0 # Use certificate to authenticate
# Certificates used for authentication: ... either normal certs
#CertFile=/home/does/.globus/usercert.pem
#KeyFile=/home/does/.globus/userkey.pem
# ... or a grid proxy certificate
UseGridProxy = 1
# Use password instead of certificate to authenticate
# Password = secret
#VerifyServerCert = 1
```

```

#IgnoreCertificateNameMismatch = 0
# If server certificates are verified, CA certificates need to be loaded:
TrustedCertDir = /etc/grid-security/certificates
  
```

The following parameters can be configured:

- **Host:** the name of the host where the AMGA server is running. Default is `localhost`
- **Port:** the port of the host to connect to. Default is `8822`.
- **Login:** the login name used for the AMGA server. This will be the owner of the new entries created in the catalog.
- **Home:** the home-directory. The default is `"/`.
- **UseSSL:** possible values are `no`, `try`, `require` (or `yes`). The default value is `no`. It has to be set to `yes` if the user wants to make the authentication via certificates or proxies. If `UseSSL` is on, all the session is encrypted. If it is `off`, all the parameters below will be ignored.
- **AuthenticateWithCertificate:** set to `1` to enable certificate based authentication. If this option is on, then you should provide either a key and certificate pair, or a proxy. If both options are provided, then the proxy has the priority.
- **CertFile:** the path to the certificate file. It has to be in `.pem` format. Note that the `UseGridProxy` option should be disabled, otherwise the grid proxy has the priority.
- **KeyFile:** the path to the key file. Like for the `CertFile` option, the `UseGridProxy` option should be disabled.
- **UseGridProxy:** the path to the X509 user grid proxy. The default value is `/tmp/x509up_u[userId]`.
- **Password:** you can also use a password to authenticate with your login. Password is sent encrypted if SSL is used.
- **VerifyServerCert:** if set to `1` (default value), verifies the server certificate against the CA certificate located in `TrustedCertDir` (see below for this option).
- **IgnoreCertificateNameMismatch:** if set to `yes`, do not try to match the DN in the server certificate with the host name of the server. Useful for services that are multi-homed (for example that have an alias for the host name)
- **TrustedCertDir:** the directory where the CA certificate is stored.

7.9.3. Metadata access from the shell

AMGA provides two ways to access metadata from the shell: via the `cli` command line and via the `mdclient` metadata terminal interface.

To start the `mdclient` interface, just enter:

```
mdclient [-p port] [hostname]
```


where the `-p` option and the `hostname` argument can override the corresponding values set in the configuration file. This will establish a connection to the database the AMGA server points to.

AMGA has its own language, quite similar to SQL, with the main difference that it deals with tables as if they were directories. For example, the content of the database schema you are connected to can be listed with the `ls` command:

```
$ mdclient
Connecting to lxn1187.cern.ch:8822...
ARDA Metadata Server 1.3.0
Query> ls
>> /files
>> /jobs
>> /evtTypes
```

or, in an equivalent way, using the `mdcli` command line client:

```
$ mdcli ls
/files
/jobs
/evtTypes
```

To list the columns (or attributes) of a table:

```
$ mdcli listattr /evtTypes
Description
varchar(256)
EventTypeId
int
Primary
varchar(256)
```

the command result is the sequence of attribute names and types, for all the columns of the table.

7.9.4. Some commands to manipulate entries and attributes of a table

AMGA supports all the main commands to manage data in the database. To list all the possible commands available in the `mdclient` interface you can type `help`. A complete documentation of the available commands is given in [59], here we will report some of them, just to show the way AMGA works.

Example 7.9.4.1 (Adding new entries to a table)

First we check the attributes name and type:

```
Query> listattr test
>> i
>> int
```

which means that the table `test` has one column of type `int`. In AMGA an entry in a table is handled like a file in a directory. The syntax to add a new entry is: `addentry file column_name value`, where 'file' is the full path name of the new entry, in analogy with the file name in a file system. If the path name is not provided, then the new entry is created in the current directory (or table). For example, to add an entry into the `test` table, assigning the value 10 to the column `i`, we should type:

```

Query> addentry /test/a i 10
Query> listentries /test/
>> /test/a
  
```

Example 7.9.4.2 (Removing an entry from a table)

The syntax to remove an entry from a table is:

```
rm [-options] pattern [conditions]
```

where `pattern` is the path to the file to be removed. It also can contain wild characters to match more entries. In this example we first list the content of the directory `test` and then we delete some entries:

```

Query> listentries /test/
>> /test/a
>> /test/b
>> /test/t1
>> /test/t2
Query> rm /test/a
Query> rm /test/t*
Query> listentries /test/
>> /test/b
  
```

Example 7.9.4.3 (Adding and removing an attribute from a table)

The syntax to add a new column to a table is: `adattr table_name column_name data_type`. For example, to add the column `column2` to the `test` table:

```
Query> addattr test column2 int
```

and to remove it:

```
Query> removeattr test column2
```

AMGA provides numeric, character and time stamp data type which correspond to the data type of the different database backends. A table showing the correspondance is reported in the manual in [59].

In addition to these basic commands, AMGA can manage more complex operations like creating indexes, table constraints, creation of views and sequences. For a more complete documentation you can refer again to the user manual [59].

7.9.5. Using the API's

AMGA provides API's in several programming languages such as: Java, C++, Python and Perl. Developing code using the API has the big advantage to be totally decoupled of the database backend. The functionality provided by the API's is the same described for the client (see 7.9.3).

Here we will provide some examples about the usage of the Python API. The API can be used adding to the `$PYTHONPATH` variable the directory where the `mdclient` package is located and then importing the `mdclient` module. From the Python interactive prompt:

```

>>> import mdclient
>>> client = mdclient.MDClient('localhost', 8822, 'root')
>>> dir(client)
['_MDClient__dataArrived', '_MDClient__fetchData', '_MDClient__fetchRow',
'_MDClient__quoteValue', '_MDClient__sendCommand', '__doc__', '__init__',
'__module__', 'abort', 'addAttr', 'addEntries', 'addEntry', 'buffer',
'cd', 'clearAttr', 'commit', 'connect', 'connected', 'createDir',
'disconnect', 'eot', 'execute', 'executeNoWait', 'getEntry',
'getSelectAttrEntry', 'getattr', 'greetings', 'host', 'keepalive',
'listAttr', 'listEntries', 'login', 'password', 'port', 'protocolVersion',
'put', 'pwd', 'removeAttr', 'removeDir', 'reqSSL', 'requireSSL',
'retrieveResult', 'rm', 'selectAttr', 'sequenceCreate', 'sequenceNext',
'sequenceRemove', 'session', 'sessionID', 'setAttr', 'sslOptions',
'sslSock', 'updateAttr', 'upload']
>>> client.createDir("/pytest")
>>> client.cd("/pytest")
>>> client.addAttr(".", "events", "int")
>>> client.addAttr("/pytest", "eventGen", "varchar(20)")
>>> client.listAttr('pytest')
(['events', 'eventGen'], ['int', 'varchar(20)'])

```

In the sequence of commands above reported, first of all we instantiate an object, `client`, which provides the connection to the AMGA server running on the localhost, on the port 8822. Then we create a directory and therein we add some attributes. The location where to create the attributes can be specified, exactly as in a file system, with the `'.'` symbol or with the absolute path. This is equivalent to creating a table and adding some columns. Then, we can add entries to the table:

```

>>> for i in range(0,3):
...     client.addEntry("/pytest/t"+str(i), ['events', 'eventGen'], [ i*100,
'LHCs Gen'])

```

after, we can see the values of the entries for the desired attributes using the `getattr` method. The syntax is: `getattr(pattern, AttributeList)`, where `'pattern'` is the path to the attributes, including also wild characters, and the `'AttributeList'` is a Python list containing the attributes to display. In this case, to display the values of all the entries for the `eventGen` and `events` attributes of the `pytest` table:

```

>>> client getattr('/pytest/*', ['eventGen', 'events'])
>>> while not client.eot():
...     file, values=client.getEntry()

```

```
...     print "-->",file, values
...
-> t2 ['200', 'LHCs Gen']
-> t0 ['0', 'LHCs Gen']
-> t1 ['100', 'LHCs Gen']
```

A select query can be done using the `selectAttr` method of the client object, with the following syntax: `selectAttr(AttributeList, Condition)`. Where the 'AttributeList' is a python list containing the attributes to display as the result of the query and the 'Condition' is a string containing the clause of the query. As an example:

```
>>> client.selectAttr(['/pytest:eventGen', '/pytest:events'], '/pytest:events = 100')
>>> while not client.eot():
...     eventGen,events = client.getSelectAttrEntry()
...     print 'eventGen = ', eventGen, ' events = ', events
eventGen = LHCs Gen events = 100
```

the `selectAttr` method executes the query on the database backend and stores the result in a buffer, which can be read later using the `getSelectAttrEntry` method, as shown above.

Component	EGEE	EDG	EDT	INFN Grid	Globus	Condor	Other
Basic middleware							
Globus Toolkit 2.4.3 ClassAds 0.9.7					✓	✓	
Authentication and Authorisation							
MyProxy 0.6.1 VOMS VOMRS LCAS/LCMAPS	✓	✓					✓ ✓
Workload management							
Condor-G 6.6.7 EDG WMS gLite WMS	✓	✓				✓	
Data management							
LFC DPM FTS GFAL LCG DM tools	✓ ✓ ✓ ✓ ✓						
Fabric management							
Quattor YAIM	✓						✓
Monitoring							
GridICE				✓			
Information system							
MDS Glue Schema BDII R-GMA LCG Information tools	✓ ✓ ✓	✓	✓		✓		

Table 1: Software components of gLite 3.1 and projects that contributed to them.

APPENDIX A THE GRID MIDDLEWARE

The only operating system currently supported by gLite 3.1 is Scientific Linux 3[45] and the supported architecture is IA32. It is foreseen to have soon support for Scientific Linux 4 and the x86_64 and IA64 architectures.

The gLite 3.1 middleware layer uses components from several Grid projects, including EGEE, Datagrid (EDG), DataTag (EDT), DataGrid (EDG), INFN-GRID, Globus and Condor. In some cases, patches have been applied to some 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 gLite 3.1 are listed in table 1.

APPENDIX B ENVIRONMENT VARIABLES AND CONFIGURATION FILES

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.

Environment variables

Variable	Definition	UI	WN
EDG_LOCATION	EDG middleware installation directory	✓	✓
EDG_WL_JOBID	JobID (defined for a running job)		✓
EDG_WL_LOCATION	LCG-2 WMS UI installation directory	✓	
EDG_WL_RB_BROKERINFO	Location of the .BrokerInfo file		✓
EDG_WL_UI_CONFIG_VAR	Non-standard location of the LCG-2 WMS UI configuration file	✓	
EDG_WL_UI_CONFIG_VO	Non-standard location of the VO-specific LCG-2 WMS UI configuration file	✓	
GLITE_LOCATION	gLite middleware installation directory	✓	✓
GLITE_SD_PLUGIN	Sets the type of service discovery implementation to be used (file, bdi, rgma)	✓	✓
GLITE_SD_SITE	Sets the local site where to find services	✓	✓
GLITE_SD_VO	Sets the default VO for which to find services	✓	✓
GLITE_WMS_CLIENT_CONFIG	Non-standard location of the gLite WMPProxy UI configuration file	✓	
GLITE_WMSUI_CONFIG_VAR	Non-standard location of the gLite WMS UI configuration file	✓	
GLITE_WMSUI_CONFIG_VO	Non-standard location of the VO-specific gLite WMS UI configuration file	✓	
GLOBUS_LOCATION	Globus middleware installation directory	✓	✓
LCG_CATALOG_TYPE	Type of file catalogue used by lcg_util and GFAL (it should be lfc)	✓	✓
LCG_GFAL_INFOSYS	comma-separated list of BDII contact strings for lcg_utils and GFAL (<hostname>:<port>)	✓	✓
LCG_GFAL_VO	User's VO for lcg_utils and GFAL	✓	✓
LFC_HOST	Location of the LFC catalogue	✓	✓
LCG_LOCATION	LCG middleware installation directory	✓	✓
LCG_RFIO_TYPE	Type of RFIO for GFAL (dpm or castor)	✓	✓
VO_<VO>_DEFAULT_SE	Default SE for the VO <VO>	✓	✓
VO_<VO>_SW_DIR	<VO>'s software installation directory		✓
X509_CERT_DIR	Directory containing the CA certificates	✓	✓
X509_USER_CERT	User's certificate file	✓	
X509_USER_KEY	User's private key file	✓	
X509_USER_PROXY	User's proxy certificate file	✓	✓
X509_VOMS_DIR	Directory containing the certificates of the VOMS servers	✓	✓

Configuration files

Configuration File	Notes
<code>\$GLITE_LOCATION/etc/vomses</code>	System-level configuration of the VOMS CLI
<code>\$HOME/.glite/vomses</code>	User-level configuration of the VOMS CLI
<code>\$GLITE_LOCATION/etc/<vo>/glite_wms.conf</code>	Configuration file for the WMProxy CLI for the VO <VO>
<code>\$GLITE_LOCATION/etc/glite_wmsui_cmd_var.conf</code>	Generic configuration file for the gLite WMS CLI via NS
<code>\$GLITE_LOCATION/etc/<vo>/glite_wmsui.conf</code>	VO-specific configuration file for the gLite WMS CLI via NS for the VO <VO>
<code>\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf</code>	Generic configuration file for the LCG-2 WMS
<code>\$EDG_WL_LOCATION/etc/<vo>/edg_wl_ui.conf</code>	VO-specific configuration file for the LCG-2 WMS

APPENDIX C JOB STATUS DEFINITION

As it was already mentioned in Chapter 6, a job can find itself in one of several possible states. Also, only some transitions between states are allowed. These transitions are depicted in Figure 12. For completeness, also the DAG states are described in 13.

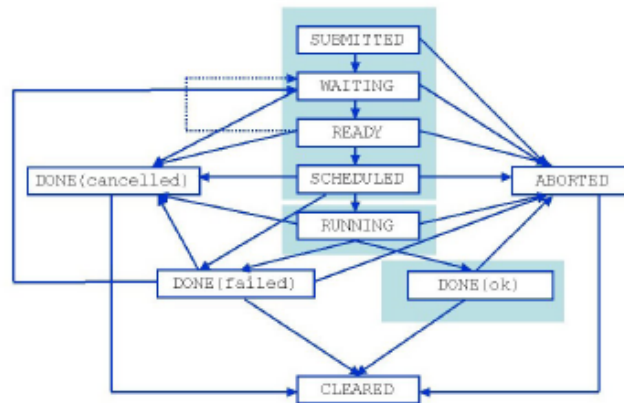


Figure 12: Possible job states in gLite 3.1

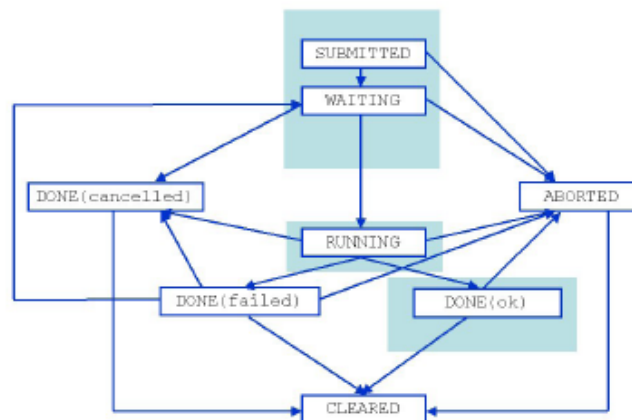


Figure 13: Possible DAG states in gLite 3.1

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

Status	Definition
SUBMITTED	The job has been submitted by the user but not yet processed by the Network Server or WMProxy
WAITING	The job has been accepted by the Network Server or WMProxy 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 VO-WIDE UTILITIES

D.1. INTRODUCTION

This section describes some administrative tools that are only relevant to selected VO members (VO managers, VO software managers, etc.). Links to other sources of documentation are provided when available.

D.2. FREEDOM OF CHOICE FOR RESOURCES

The *Freedom of Choice for Resources (FCR)* is a tool for VO Software Managers to set up selection rules for Computing and Storage Elements, which will determine whether a particular resource will be available or not to the VO.

The FCR interface[49] allows the VO manager to decide if a resource supporting his VO should be visible at any time, invisible at any time or visible only if it passes periodic tests run by the WLCG/EGEE operations team, or by the VO itself (see next section). The VO manager can also decide what tests are to be considered critical for its VO: only the failure of critical tests determines the exclusion of a resource.

D.3. SERVICE AVAILABILITY MONITORING

The *Service Availability Monitoring (SAM)*[50] is a framework to provide a centralized and uniform monitoring tool for all Grid services.

It is based on the concept of running periodic tests on all known Grid services to determine whether they are working properly. These tests can both be directly run from a User Interface, or sent out as Grid jobs (for example, to test a Computing Element). It provides detailed information about the overall status of the service, and about the outcome of the individual tests, and keeps a historical record of the information.

The SAM is a very useful tool both for the WLCG/EGEE operations team and for the VO members. It also allows Virtual Organisations to complement the standard tests with custom tests covering VO-specific functionalities.

The user can view the results of the SAM tests on the production WLCG/EGEE infrastructure from the SAM Web interface [51]. From this page, the user can choose the service type (CE, LFC, WMS, etc.), a VO, a region and the specific tests he is interested in.

D.4. THE VO BOX

The *VO box*[52] is a type of node, which is deployed at many sites, where the VO can run specific agents and services. The access to the VO box is restricted to the VO software manager of the VO.

The VO box offers basically two main features to the users:

- VOs can run their own services from this node;

- it provides direct access to the software area of each VO, also accessible from all WNs of the site.

Each VO should negotiate with the site the setup of the VO box depending on the services which are run inside that node.

D.5. VO SOFTWARE INSTALLATION

Authorized users can install VO-specific software on the computing resources of WLCG/EGEE. The availability of such software can be advertised in the Information System[47].

The *VO Software Manager* is the member of the VO with the privileges to install VO-specific software on the different sites. The software manager can install, validate or remove VO-specific software on a site at any time through a normal Grid job. Normally, the software manager privileges are expressed by a special VOMS role, which must be taken when creating the VOMS proxy used to submit the software installation job.

The VO software manager can also modify the VO-specific information for the CEs using either the command `lcg-tags` or the command `lcg-ManageVOTag`, available from the UI and the WN.

Each site should provide 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, as well as any special priority for software installation jobs.

D.6. USING LCG-TAGS

This command allows to list, add and remove tags to computing elements, which are normally used to advertise the availability of some piece of software. A full man page is available, so we will give just a few examples.

Example D.6.1 (List all the tags for a CE and a given VO)

If you want to see all the tags that have been defined by a VO on a CE, you can run something like

```

$ lcg-tags --ce cel110.cern.ch --vo cms --list
VO-cms-CMSSW_3_0_0_pre2
VO-cms-dummyarch
VO-cms-slc4_ia32_gcc345

```

Please note that the command works by locally copying via GridFTP the file containing the tags to be published via the CE GRIS; it may happen that, due to problems with the information system, the tags are visible via `lcg-tags` but not in the BDII.

Example D.6.2 (Modify the tags of a CE)

If you are entitled to the `lcgadmin` VOMS role for a VO, you can also modify the tags attached by that VO to a CE. For example, to remove a tag, you can do

```
$ lcg-tags --ce cell10.cern.ch --vo cms --remove --tags VO-cms-dummyarch
```

and, to add a new tag:

```
$ lcg-tags --ce cell10.cern.ch --vo cms --add --tags VO-cms-mytag
```

The option `--tags` can contain a comma-separated list of tags.

More tags can be added or removed at the same time also using the option `--tagfile tagfile` instead of `--tags`, where `tagfile` is a file containing a list of tags separated by any number of spaces, tabs or newlines.

Finally, the options `--clean` and `--replace` allow respectively to remove all tags and to replace the current tags with those specified by `--tags` or `--tagfile`.

D.7. USING LCG-MANAGEVOTAG

The command `lcg-ManageVOTag` has a similar syntax and functionality. For details, use

```
$ lcg-ManageVOTag -help
```

APPENDIX E DATA MANAGEMENT AND FILE ACCESS THROUGH AN APPLICATION PROGRAMMING INTERFACE

In this section, an overview of the available Data management API will be given, and some details on the most high-level API will be provided.

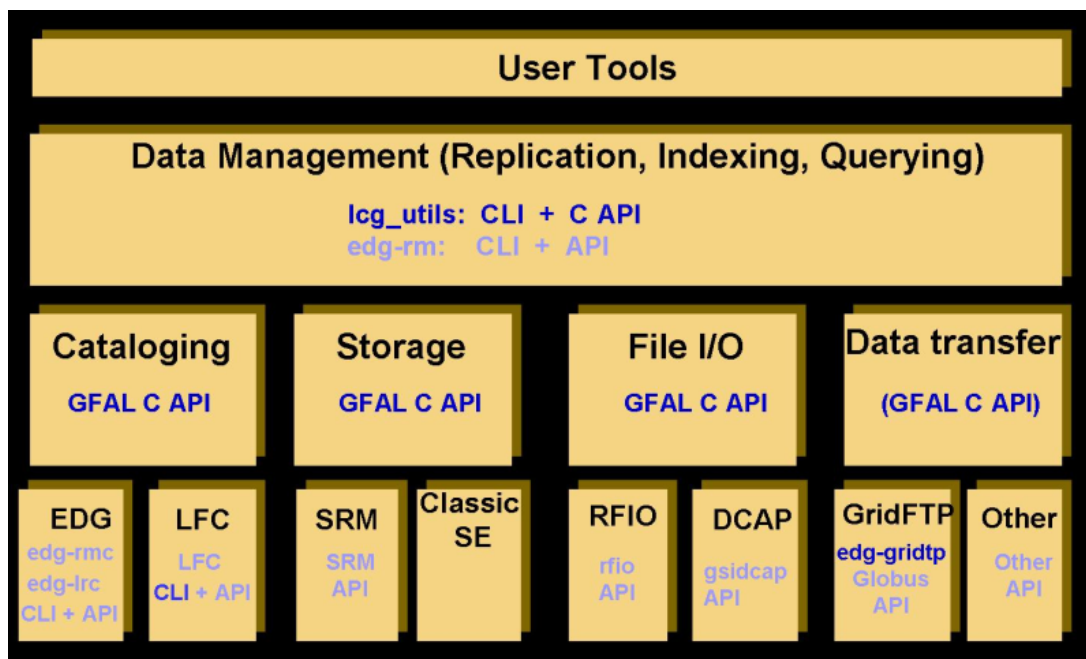


Figure 14: Layered view of the Data Management APIs and CLIs

Figure 14 shows a layered view of the different gLite Data Management API and of the CLI which were described earlier. The CLI and API whose use is discouraged are shadowed.

Just below the user tools, we find the `lcg_util` API. This is a C API that provides the same functionality as the `lcg-*` commands for Data Management we have already seen: in fact, the commands are just wrappers around the C calls. This layer should cover most of the basic needs of user applications. It is independent from the underlying technology, since it will transparently interact with the LFC catalogue and will use the correct protocol (GSIFTP, RFIO or `gsidcap`) for file transfer.

In the following table, all the available methods and a short description are listed.

Method name	Description
lcg_aa	add an alias in the LFC for a given GUID
lcg_cp	copy a Grid file to a local destination
lcg_cr	copy and register a file
lcg_del	delete one file (either one replica or all replicas)
lcg_gt	get the TURL given the SURL and the transfer protocol
lcg_la	get the list of aliases for a given LFN, GUID or SURL
lcg_lg	get the GUID for a given LFN or SURL
lcg_lr	get the list of replicas for a given LFN, GUID or SURL
lcg_ra	remove an alias in LFC for a given GUID
lcg_rep	copy a file from one SE to another and register it in LFC
lcg_rf	register in LFC a file residing on a SE
lcg_sd	set file status to "Done" for a given SURL in a specified request
lcg_stmd	get space tokens associated to a space token descriptor
lcg_uf	unregister in LFC a file residing on a SE

Apart from the basic calls `lcg_cp`, `lcg_cr`, etc., there are other calls that enhance them with a buffer for complete error messages (`lcg_cpx`, `lcg_crx`, ...), that include timeouts (`lcg_cpt`, `lcg_crt`, ...), and both (`lcg_cpxt`, `lcg_crx`). 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 a POSIX-like interface for I/O operations on Grid files, effectively hiding the interactions with the LFC, the SEs and SRM. 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. It will automatically select the most appropriate transfer protocol, depending on the kind of SE the file is located on (if a TURL is used, the protocol is already implicitly specified).

Note: In the case where LFNs or GUIDs are used, GFAL (and, as a consequence, `lcg_util`) needs to contact the LFC to obtain the corresponding TURL. For GFAL to be able to discover the LFC endpoints and to find out information about the Storage Elements, the user must set the environment variables `LCG_GFAL_VO` and `LCG_GFAL_INFOSYS` to the VO name and a comma-separated list of BDII hostnames and ports. For example:

```
export LCG_GFAL_VO=cms
export LCG_GFAL_INFOSYS=exp-bdii.cern.ch:2170,grid01.lal.in2p3.fr:2170
```

The endpoint of the catalogue may also be directly specified by setting the environment variable `LFC_HOST`. For example:

```
export LFC_HOST=prod-lfc-cms-central.cern.ch
```

In Figure 15, it is shown 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 catalogue 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 filehandle to the caller.

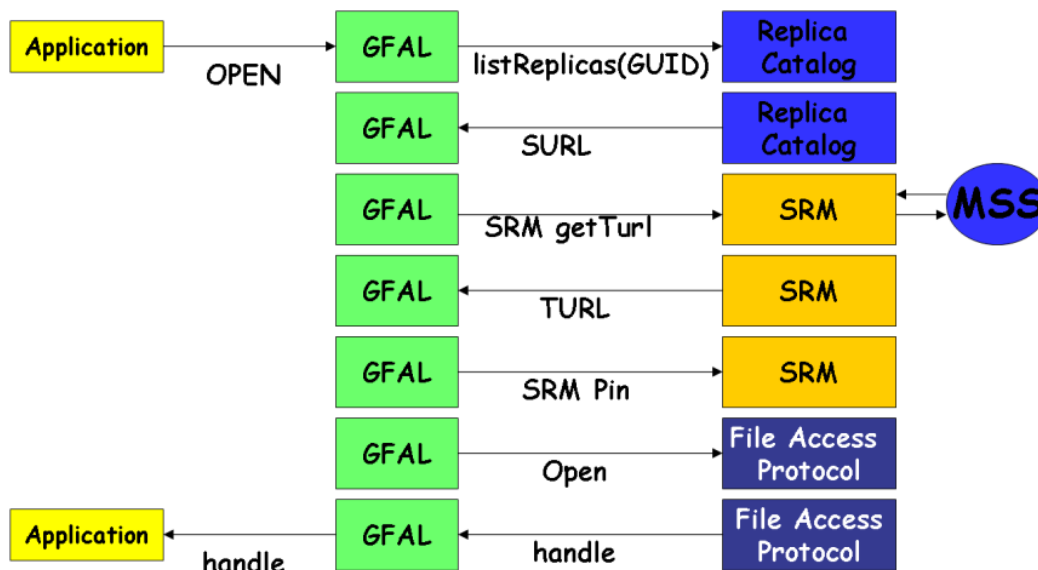


Figure 15: Flow diagram of a GFAL call

It is important to notice that if a file is created with GFAL naming it by SURL, for it to be used as a Grid file, it should be manually registered with a LFN using `lcg-rf`.

In addition, GFAL may expose functionality applicable only to a specific 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.

For more information on GFAL, refer to the manpages of the library (`gfal`) and of the different calls (`gfal_open`, `gfal_write...`).

Finally, below GFAL, we find some other CLI and API 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 E.0.1 (Using `lcg_util` API to transfer a file)

The following example copies a file from a SE to the local host. The file can be then accessed locally with normal file I/O calls.

The source code follows (`lcg_cp_example.cpp`):

```
#include <iostream>

extern "C"{
    #include "lcg_util.h"
}
```

```

    using namespace std;

int main(int argc, char **argv){

/* Check syntax (there must be 2 arguments) */
  if (argc != 3) {
    cerr << "Usage: " << argv[0]<< " source destination\n";
    exit (1);
  }
  char * src_file=argv[1];
  char * my_file=argv[2];
  char * dest_file=new char[200];
  char * vo=getenv("LCG_GFAL_VO");
  int nbstreams=1;
  int verbose=1;
  int timeout=180;

/* Form the name of the destination 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_cpt(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!");
  }

/* That was it */
  cout << endl;
  return 0;

} //end of main

```

The code can be compiled with the following command:

```

$ c++ -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 link 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 link as shown in the example was necessary for the program to compile in the moment that this guide was written.

The resulting executable will take two arguments: the name of a Grid file and a local name for the file in the current directory. For example:


```

$ ./lcg_cp_example lfn:/grid/cms/does/gridfile localfile
Using grid catalog type: lfc
Using grid catalog : prod-lfc-cms-central.cern.ch
Source URL: lfn:/grid/cms/does/gridfile
File size: 7840
VO name: cms
Source URL for copy: gsiftp://lxfsr4601.cern.ch//castor/cern.ch/grid/cms/generated/
2006-11-14/fileee903ced-b61a-4443-b9d2-a4b0758721a8
Destination URL: file:/home/does/localfile
# streams: 1
# set timeout to 180 (seconds)
          0 bytes      0.00 KB/sec avg      0.00 KB/sec inst
Transfer took 6080 ms
File correctly copied to local filesystem
  
```

Example E.0.2 (Using GFAL to access a file)

The following C++ code uses GFAL to access a Grid file. The program opens the file, writes a set of numbers into it, and closes it. Afterwards, the file 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:

```
$ c++ -I$LCG_LOCATION/include -L$LCG_LOCATION/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/does/gfal/test.txt
... Open successful ... Write successful ... Close successful

Reading back file:///afs/cern.ch/user/d/does/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

```

This example will not work in all cases from a UI, though. Due to the limitations of the insecure RFIO protocol, GFAL can work with a classic SE or a CASTOR SE only from a worker node at the same site. The reason is that insecure RFIO does not handle Grid certificates, and while the local UNIX user ID to which a user job is mapped on the WN will be allowed to access a file in the local SE, the UNIX user ID of the user on the UI will be normally 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 UNIX user IDs. For this reason, it can be used with no problem

to access files from UIs or in remote SEs, just as `gsidcap` can. As a consequence, the example will work without any problem from the UI with DPM and dCache.

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 Information System. The result is that currently GFAL has to figure out which one of the two RFIO versions it uses based on the environment variable `LCG_RFIO_TYPE`. If its value is `dpm`, the secure version of RFIO will be used; otherwise insecure RFIO will be the used. Therefore, the user must correctly define the indicated variable depending on the SE he wants to talk to.

Another important issue is that of the names used to access files. For classic SEs, `SURLs` and `TURLs` must include a double slash between the hostname of the SE and the path of the file. This is a known limitation in GFAL and insecure RFIO. For example:

```
sfn://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
rfio://lxb0710.cern.ch//flatfiles/SE00/dteam/my_file
```

As seen in previous examples, the `lcg-*` commands will work with `SURLs` and `TURLs` registered in the catalogues, even if they do not follow this rules. 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.

In addition to GFAL, there is also the possibility to use the RFIO C and C++ API, which also allows to remotely open and read a file. Nevertheless, this is not recommended, as it can work only with classic SEs and CASTOR SEs located in the same local area network, and RFIO does not understand `LFNs`, `GUIDs` or `SURLs`. More information on RFIO and its API can be found in [42].

Example E.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 after the file has been migrated.

The program uses both the `lcg_util` and the GFAL API. 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 forever, 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.

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 lies 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 were run a second time, 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
(rfiio://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 F THE GLUE SCHEMA

The GLUE information schema [21] provides a standardised description of a Grid computing system, to enable resources and services to be presented to users and external services in a uniform way. The schema has been defined as a joint project between a number of Grids. It does not attempt to model the real systems in any detail, but rather to provide a set of attributes to facilitate key use cases. The intended uses are resource discovery (“what is out there?”), selection (“what are the properties?”) and monitoring (“what is the state of the system?”). Inevitably, when real systems are mapped on to the standard schema some details are lost and some features may not entirely match the assumptions in the schema.

The schema has evolved as experience has been gained and new features have required schema support. The current schema version deployed with gLite 3.1 is 1.2, but version 1.3 has now been defined and is expected to be deployed during 2007. A working group is also being created in the context of the Open Grid Forum to define a major revision (2.0) to be deployed in 2008.

F.1. BASIC CONCEPTS

The schema itself is defined in an abstract way, using a simplified UML description, in terms of *objects* which have *attributes* and *relations* to other objects. Some attributes may be *multivalued*, i.e. there may be many instances of the same attribute for one object instance, and most attributes are *optional*. Most objects have an *identifier*, which is either globally unique (*UniqueID*) or unique in its local context (*LocalID*). Many objects also have a human-readable *Name* to provide a more user-friendly description (i.e. the Name is an attribute of an object instance, as opposed to the name of the abstract object itself). Neither the UniqueID nor the LocalID should have any semantics, i.e. they should not be interpreted or decoded by the middleware.

Attributes have *types*. In many cases these are simply `int32` or `string`. However, where possible more restrictive types are used, notably for URLs (or the more general URIs) and for enumerated lists, where the allowed values must be taken from a defined set. Other restrictions on values may be implicit, e.g. for latitude and longitude. In most cases, if an attribute is not available or is irrelevant it is simply omitted, rather than taking a special value.

The schema is split into a number of high-level pieces. *Site* defines some information about an entire Grid site. *Service* provides a general abstraction for any Grid service. *CE* and *SE* provide detailed information about Computing and Storage Elements, as these are the most important components of the Grid. In GLUE terms a CE represents a queue in a batch system; the schema also describes a *Cluster* which represents the hardware (Worker Nodes) which can be accessed via a CE.

Finally, *CESEBind* allows relationships between “close” CEs and SEs to be defined. Historically there was also a *Host* concept to allow individual Worker Nodes to be represented in detail, but in a Grid context this has not proved to be useful.

These high-level abstractions are in most cases broken down further into a number of objects representing those details of the system which need to be exposed to the outside world.

F.2. MAPPINGS

The abstract schema has a set of *mappings* to specific Information System technologies, currently LDAP, R-GMA (relational) and XML, of which the first two are currently in use in WLCG/EGEE. These technologies are

quite different and therefore the mappings have significantly different structures, although the basic objects and attributes are preserved. For example, relational tables cannot support multivalued attributes directly (a table cell can only hold a single value), hence these are split into separate tables with an extra key attribute. Similarly the LDAP mapping introduces extra attributes (*ForeignKey* and *ChunkKey*) to allow relations between objects to be expressed in a way which is usable in LDAP queries.

These mappings are to some extent a matter of judgement, hence they are defined by hand rather than being automated. Details can be found on the GLUE web site [21], or simply by exploring the LDAP or R-GMA structures with a browser.

A further mapping is to the Classad language used in JDL files, which is the principle way in which most users interact with the schema. The WMS is able to convert from both LDAP and R-GMA to this form. In most respects this “flattens” the schema and simply provides a list of attributes which can be used to construct the `Rank` and `Requirements` expressions.

F.3. INFORMATION PROVIDERS

The information presented in the schema is produced by programs known as *Information Providers*. These are structured as a framework *Generic Information Provider* with a set of plugins for specific parts of the schema, which may vary depending on the circumstances, e.g. to cope with different batch systems or SE technologies. The same providers are used for both LDAP and R-GMA.

Broadly speaking these divide into *static* and *dynamic* providers. Dynamic information (e.g. the number of running jobs) changes on a short timescale and therefore has to be collected from some other system (e.g. a batch system) every time the provider is run. By contrast, static information (e.g. the maximum CPU time for a queue) changes infrequently, and the providers therefore read it from a configuration file which is updated only when the system is reconfigured.

To reduce the load on the underlying systems the dynamic information is normally cached for a short time. In addition there are usually further caches and propagation delays in the wider Information Systems. It should therefore be assumed that dynamic information is always somewhat out of date, typically by a few minutes.

F.4. GLUE ATTRIBUTES

The rest of this appendix provides some information about those schema attributes which are the most important in WLCG/EGEE. For a full list of attributes see the GLUE documentation. Attributes not mentioned here are generally either not defined in WLCG/EGEE, not usable in practice, or are standard elements like `UniqueID` or `Name`.

It should be emphasised that the Information Providers in WLCG/EGEE do not provide everything defined in the schema (most schema attributes are optional), and in addition WLCG/EGEE imposes some extra constraints on the attributes which are not part of the schema itself, hence some of the comments here do not apply to the schema in general. Attributes defined in the 1.3 schema are included here for completeness, but it should be borne in mind that this schema is not expected to be fully deployed for some time (the LDAP mapping includes the schema version as attributes, `GlueSchemaVersionMajor` and `GlueSchemaVersionMinor`). Some attributes are deprecated; these are mentioned here if they are still in common use.

F.4.1. Site information

This provides information about a grid site as a whole. Most of the information is intended to be human-readable, as opposed to being used by the middleware. Most entries are set by hand by the system managers and hence may vary from site to site, although some are configured in a standard way by the WLCG/EGEE tools.

- **GlueSite object**

- **GlueSiteUniqueID:** This is the unique name for the site, as defined in the GOC database. This is more or less human-readable, and the Name is currently the same string in most cases.
- **GlueSiteDescription:** A general description of the site.
- **GlueSiteEmailContact:** A `mailto:` URL defining a general email contact address for the site; however, note that in WLCG/EGEE users should normally contact sites via GGUS. Separate attributes may define specific contacts for user support, security and system management.
- **GlueSiteLocation:** The geographical location of the site as a string, normally in the form City, State, Country.
- **GlueSiteLatitude, GlueSiteLongitude:** The map reference for the site, in degrees. The resolution usually locates the site to within 100m.
- **GlueSiteWeb:** A URL pointing to a web page relating to the site.
- **GlueSiteSponsor:** The organisation(s) providing funding for the site.
- **GlueSiteOtherInfo:** A multivalued string which may contain any further information the site considers useful; in WLCG/EGEE this generally includes the Tier affiliation, in the form TIER-n.

F.4.2. Service information

This provides a general abstraction of a Grid service (not necessarily a web service, and perhaps not even something with an externally visible endpoint). This information is also available via the *Service Discovery* API (see Section 5.2). At present the Service information is not directly related to the CE and SE information described below, but it is likely that the proposed major revision of the GLUE schema will describe everything as a specialisation of a service concept.

- **GlueService object**

- **GlueServiceType:** The service type, taken from a defined list which can be found on the Glue web site [21].
- **GlueServiceVersion:** The version of the service, in the form major.minor.patch.
- **GlueServiceEndpoint:** The network endpoint for the service.
- **GlueServiceStatus:** The status of the service (one of OK, Warning, Critical, Unknown, Other).
- **GlueServiceStatusInfo:** A textual explanation of the Status.
- **GlueServiceWSDL:** For web services this is a URL pointing to the WSDL definition of the service.
- **GlueServiceSemantics:** This is a URL which would typically point to a web page explaining how to use the service.
- **GlueServiceOwner:** The service owner, if any; typically a VO name.
- **AccessControlBaseRule:** A set of ACLs defining who is allowed access to the service.

F.4.3. Attributes for the Computing Element

These are attributes that give information about the computing system (batch queues and Worker Nodes). These are mostly available for use in the JDL, and consequently are the most important for most users.

Note that the term CE is overloaded; in different contexts it may refer to the front-end machine through which jobs are submitted, or to the entire set of computing hardware at a site. However, in the GLUE schema a CE is a single queue in a batch system, and there are typically many CEs at a site submitting jobs to the same set of WNs. This means that attributes published per-CE, e.g. the total number of available CPUs (or job slots), cannot be summed in a simple way.

The original schema concept was to represent the computing hardware as a Cluster consisting of one or more SubClusters, where each SubCluster would describe a set of identical WNs. However, limitations in the WMS mean that currently only a single SubCluster per Cluster (and hence per CE) is supported. Most Grid sites have WNs which are heterogeneous in various ways (processor speed, memory size etc), hence they have to publish the attributes of a representative WN which may not always match the hardware on which a job actually runs. However, the variations are generally not too large, and nodes should not differ in more vital attributes like the OS version. In some cases sites may publish more than one Cluster, e.g. to make a set of nodes with very large memory available via a separate set of queues.

Many sites use scheduling policies which give jobs priority according to who submits them, often to give specific VOs a higher priority. This was not representable in the original schema, e.g. a queue shown with a large number of queued jobs might in fact execute a job from a particular VO immediately. As a result most sites have configured separate queues for each VO. However, this increases the management effort, and can also result in a very large number of CEs being published. The 1.2 schema revision therefore introduced the concept of a VOView, which allows a subset of the CE information to be published separately for each VO (or subgroup) for which scheduling policies are defined. This is supported by the latest version of the WMS, so it is likely that the Grid sites will gradually move back to a single set of generic queues.

- **GlueCE object**

- **GlueCEUniqueID:** The unique identifier for the CE. This is constructed from various information including a host name, batch system type and queue name, but for most purposes it should simply be treated as an identifier. The constituent information is available in other attributes if needed.
- **GlueCECapability:** Introduced in version 1.3 of the schema, this will enable sites to advertise any features not represented by specific attributes.
- **GlueCEInfoTotalCPUs:** The total number of CPUs on all WNs available via the CE, which is usually the maximum number of jobs which can run. This attribute is deprecated in favour of **MaxRunningJobs**.
- **GlueCEInfoApplicationDir:** The path of a directory in which application software is installed; normally each VO has a subdirectory within this directory.
- **GlueCEInfoDefaultSE:** The unique identifier of an SE which should be used by default to store data.
- **GlueCEStateStatus:** The queue status: one of **Queueing** (jobs are accepted but not run), **Production** (jobs are accepted and run), **Closed** (jobs are neither accepted nor run), or **Draining** (jobs are not accepted but those already in the queue are run). The JDL normally has a default Requirement for the Status to be **Production**.
- **GlueCEStateTotalJobs:** The total number of jobs in this queue (running + waiting).
- **GlueCEStateRunningJobs:** The number of running jobs in this queue.
- **GlueCEStateWaitingJobs:** The number of jobs in this queue waiting for execution.

- `GlueCEStateWorstResponseTime`: The worst-case time between the submission of a new job and the start of its execution, in seconds.
- `GlueCEStateEstimatedResponseTime`: An estimate of the likely time between the submission of a new job and the start of its execution, in seconds. This is usually the default `Rank` in the JDL, i.e. jobs will be submitted to the queue with the shortest estimated time to execution. However, note that the estimate may not always be very accurate, and that all queues which currently have free execution slots will have an `EstimatedResponseTime` of 0 (or close to 0).
- `GlueCEStateFreeCPUs`: The number of CPUs not currently running a job. This is deprecated in favour of `FreeJobSlots`, since the relationship between CPUs and jobs is not always one-to-one.
- `GlueCEStateFreeJobSlots`: The number of jobs that could start immediately if submitted to this queue.
- `GlueCEPolicyMaxWallClockTime`: The maximum wall clock time (i.e. real time as opposed to CPU time) allowed for jobs submitted to this queue, in minutes. Jobs will usually be killed automatically after this time. Specify a `Requirement` on this attribute for jobs which are expected to spend a significant time waiting for I/O.
- `GlueCEPolicyMaxCPUtime`: The maximum CPU time available to jobs submitted to this queue, in minutes. Jobs will usually be killed after this time. Note that this value should be scaled according to the `SI00 (SpecInt)` rating, published as a `SubCluster` attribute (see below). All jobs should have a suitable `Requirement` on this value, otherwise they may be killed before they finish.
- `GlueCEPolicyMaxTotalJobs`: The maximum allowed total number of jobs in this queue. Jobs which exceed this limit will be rejected if the WMS attempts to submit them.
- `GlueCEPolicyMaxRunningJobs`: The maximum number of jobs in this queue allowed to execute simultaneously.
- `GlueCEPolicyMaxWaitingJobs`: The maximum allowed number of waiting jobs in this queue. Jobs which exceed this limit will be rejected if the WMS attempts to submit them. This attribute is new in version 1.3 of the schema.
- `GlueCEPolicyAssignedJobSlots`: The number of job execution slots assigned to this queue. This will normally be the same as `MaxRunningJobs`.
- `GlueCEPolicyMaxSlotsPerJob`: The maximum number of job slots which can be occupied by a multi-processor job. A value of 1 means that the CE does not accept multi-processor jobs. This attribute is new in version 1.3 of the schema.
- `GlueCEPolicyPreemption`: This flag is `TRUE` if jobs may be pre-empted, i.e. suspended after they start executing. This attribute is new in version 1.3 of the schema.
- `GlueCEAccessControlBaseRule`: This defines a set of rules which specify who can submit a job to this queue. This is usually of the form `VO:<vo>`, but may also specify `VOMS` roles or groups. This is taken into account automatically by the WMS.

- **GlueVOView object**

- The `VOView` object overloads a subset of the CE attributes for users defined by the `AccessControlBaseRule`. Some attributes are only defined in the 1.3 schema version.
- `GlueCECapability`: As for CE. New in 1.3.
- `GlueCEInfoTotalCPUs`: As for CE. Deprecated.
- `GlueCEInfoApplicationDir`: As for CE, but points to a VO-specific location.
- `GlueCEInfoDefaultSE`: As for CE.
- `GlueCEStateRunningJobs`: As for CE.
- `GlueCEStateWaitingJobs`: As for CE.

- GlueCEStateTotalJobs: As for CE.
- GlueCEStateEstimatedResponseTime: As for CE.
- GlueCEStateWorstResponseTime: As for CE.
- GlueCEStateFreeJobSlots: As for CE.
- GlueCEStateFreeCPUs: As for CE. Deprecated.
- GlueCEPolicyMaxWallClockTime: As for CE.
- GlueCEPolicyMaxCPUtime: As for CE.
- GlueCEPolicyMaxTotalJobs: As for CE.
- GlueCEPolicyMaxRunningJobs: As for CE.
- GlueCEPolicyMaxWaitingJobs: As for CE. New in 1.3.
- GlueCEPolicyAssignedJobSlots: As for CE.
- GlueCEPolicyMaxSlotsPerJobs: As for CE. New in 1.3.
- GlueCEPolicyPreemption: As for CE. New in 1.3.
- GlueCEAccessControlBaseRule: As for CE. This defines the set of users for which this VOView is valid.

- **GlueSubCluster object**

- GlueSubClusterTmpDir: This should be the name of a scratch directory which is shared across all WNs, e.g. via NFS. However, in practice this is not currently reliable at most sites.
- GlueSubClusterWNTmpDir: This should similarly be a scratch directory on a disk local to the WN. However, again this is not currently set reliably.
- GlueSubClusterPhysicalCPUs: The total number of real CPUs on all nodes in the subcluster. Currently this value is often not set.
- GlueSubClusterLogicalCPUs: The total number of logical CPUs on all nodes in the subcluster (e.g. including the effect of hyperthreading). Currently this value is often not set.
- GlueHostOperatingSystemName: This is the name of the OS installed on the WNs. The convention for the OS Name, Release and Version in WLCG/EGEE can be found at: http://goc.grid.sinica.edu.tw/gocwiki/How_to_publish_the_OS_name.
- GlueHostOperatingSystemRelease: The name of the OS release installed on the WNs.
- GlueHostOperatingSystemVersion: The version of the OS installed on the WNs.
- GlueHostProcessorModel: The CPU model name as defined by the vendor.
- GlueHostProcessorVendor: The name of the CPU vendor.
- GlueHostProcessorClockSpeed: The CPU clock speed in MHz.
- GlueHostMainMemoryRAMSize: The amount of physical memory in the WNs, in MB.
- GlueHostMainMemoryVirtualSize: The total amount of memory (RAM + swap space) on the WNs, in MB.
- GlueHostNetworkAdapterOutboundIP: TRUE if outbound network connections are allowed from a WN. This is normally the case in WLCG/EGEE.
- GlueHostNetworkAdapterInboundIP: TRUE if inbound network connections are allowed to a WN. This is not normally the case in WLCG/EGEE.
- GlueHostArchitectureSMPSize: The number of CPUs per WN.
- GlueHostBenchmarkSI00: The nominal SpecInt2000 speed rating for the CPU on a WN. This should be used to scale any requested time limit.

- `GlueHostApplicationSoftwareRunTimeEnvironment`: This is a multivalued string which allows the presence of specialised installed software to be advertised. VO-specific software uses the format `VO-<vo>-<sw_name_version>`.

- **GlueLocation object**

- The Location object was defined to advertise the location of installed software. However, in version 1.3 of the schema it is replaced by a new Software object.
- `GlueLocationName`: The name of the software.
- `GlueLocationPath`: The name of the directory where the software is installed.
- `GlueLocationVersion`: The software version number.

- **GlueSoftware object**

- This object is new in version 1.3 of the schema.
- `GlueSoftwareName`: The name of the software.
- `GlueSoftwareVersion`: The software version number.
- `GlueSoftwareInstalledRoot`: The name of the directory where the software is installed.
- `GlueSoftwareEnvironmentSetup`: The fully-qualified path name of a script with which to set up the application environment.
- `GlueSoftwareModuleName`: The name of the module with which to set up the application environment using a module management tool.
- `GlueSoftwareDataKey`: The name of any additional information item.
- `GlueSoftwareDataValue`: The value associated with the Key.

F.4.4. Attributes for the Storage Element

The part of the schema relating to the Storage Element has been evolving rapidly in line with the development of the SRM protocol, hence many of the attributes are new in the 1.3 schema version. Also, even for the current (1.2) schema the attributes are not always filled correctly by the information providers, or supported correctly by the middleware. This is expected to improve during 2007 as the data management software matures.

In addition to overall SE information, the schema introduces the concept of a *Storage Area* (SA). Originally this referred to a specific area of disk space in which files could be stored, but the SRM has a somewhat more abstract view of an SA as something which collects files which share some attributes. There is also a *Storage Library* (SL) which was introduced to represent the physical storage hardware, but this has been found not to be useful and is now deprecated, and hence not described here.

Auxiliary concepts are the *Control Protocol* and *Access Protocol*. The former relates to the protocol used to manage the SE; in WLCG/EGEE this currently means some version of the SRM protocol. The latter specifies the protocols used for data transfer; a typical SE will support several of these.

Storage systems have many variations, and are evolving rapidly. To allow some flexibility to publish information not otherwise represented in the schema, *Capability* attributes can be used to publish extra information, either as a simple identifier or as `key=value` pairs.

- **GlueSE object**

- **GlueSEUniqueID:** The unique identifier for the SE. This is usually the SE hostname, but it should be emphasised that this should not be assumed; the SE should be contacted using the endpoint(s) specified in the Protocol objects.
- **GlueSESizeTotal:** The total size of the SE storage in GB. In the 1.3 schema version this is deprecated and split into online and nearline components.
- **GlueSESizeFree:** The total amount of space available to store new files, in GB. In the 1.3 schema version this is deprecated and split into online and nearline components.
- **GlueSETotalOnlineSize:** The total amount of online (disk) storage, in GB. New in the 1.3 schema.
- **GlueSETotalNearlineSize:** The total amount of nearline (tape) storage, in GB. New in the 1.3 schema.
- **GlueSEUsedOnlineSize:** The total amount of online (disk) storage available to store new files, in GB. New in the 1.3 schema.
- **GlueSEUsedNearlineSize:** The total amount of nearline (tape) storage available to store new files, in GB. New in the 1.3 schema.
- **GlueSEArchitecture:** This describes the general hardware architecture of the SE. The value is one of: `tape` (a system including a tape storage robot), `disk` (simple disk storage), `multidisk` (a disk array, e.g. RAID) and `other`.
- **GlueSEImplementationName:** The name of the underlying software implementation, e.g. `dCache` or `DPM`. New in the 1.3 schema.
- **GlueSEImplementationVersion:** The version number of the software implementation. New in the 1.3 schema.
- **GlueSEStatus:** The current operational status of the whole SE. Values can be `Queuing` (the SE can accept new requests but they will be kept on hold); `Production` (the SE processes requests normally); `Closed` (the SE will not accept new requests and will not process existing ones); and `Draining` (the SE will not accept new requests, but will still process existing ones). New in the 1.3 schema.

- **GlueSEAccessProtocol object**

- **GlueSEAccessProtocolType:** The protocol type, e.g. `gsiftp` or `rfio`. See the GLUE web site [21] for the full list of types.
- **GlueSEAccessProtocolVersion:** The protocol version.
- **GlueSEAccessProtocolEndpoint:** A URL specifying the endpoint for this protocol. Note that with an SRM the endpoint is normally obtained dynamically.
- **GlueSEAccessProtocolCapability:** A multivalued string allowing arbitrary capabilities to be advertised.
- **GlueSEAccessProtocolMaxStreams:** The maximum number of data streams allowed for a single transfer using this protocol. New in the 1.3 schema.

- **GlueSEControlProtocol object**

- **GlueSEControlProtocolType:** The protocol type (usually SRM in WLCG/EGEE).
- **GlueSEControlProtocolVersion:** The protocol version.
- **GlueSEControlProtocolEndpoint:** A URL specifying the endpoint for this protocol.
- **GlueSEControlProtocolCapability:** A multivalued string allowing arbitrary capabilities to be advertised.

- **GlueSA object**

- **GlueSAPath:** This defines a path name to the root directory for this area. If specified this should be prefixed to the name (SURL) used to store the file.

- **GlueSAType**: This specifies a guarantee on the lifetime of files in the storage area. Values can be `permanent` (files will not be deleted automatically), `durable` (files may be purged after notification of the owner), `volatile` (files may be purged automatically after the expiration of a lifetime), or `other`. Currently WLCG/EGEE only supports the `permanent` type, but `volatile` (scratch) files may be supported in future.
- **GlueSASStateAvailableSpace**: The total space available for new files in this SA. Note that the units are kB, which with modern storage systems is too small. In the 1.3 schema this attribute is deprecated.
- **GlueSASStateUsedSpace**: The space used by files in this SA. Note that the units are kB, which with modern storage systems is too small. In the 1.3 schema this attribute is deprecated.
- **GlueSASStateTotalOnlineSize**: The total online (disk) space in GB for this SA. New in 1.3.
- **GlueSASStateUsedOnlineSize**: The online (disk) space in GB used by files stored in this SA. New in 1.3.
- **GlueSASStateFreeOnlineSize**: The online (disk) space in GB available for new files in this SA. New in 1.3.
- **GlueSASStateReservedOnlineSize**: The online (disk) space in GB which has been reserved for a specific purpose but not yet used. New in 1.3.
- **GlueSASStateTotalNearlineSize**: The total nearline (tape) space in GB for this SA. New in 1.3.
- **GlueSASStateUsedNearlineSize**: The nearline (tape) space in GB used by files stored in this SA. New in 1.3.
- **GlueSASStateFreeNearlineSize**: The nearline (tape) space in GB available for new files in this SA. New in 1.3.
- **GlueSASStateReservedNearlineSize**: The nearline (tape) space in GB which has been reserved for a specific purpose but not yet used. New in 1.3.
- **GlueSAAccessControlBaseRule**: This defines a set of rules which specify who can store files in this SA. This is usually of the form `VO:<vo>` (or for historical reasons simply the VO name), but may also specify VOMS roles or groups.
- **GlueSARetentionPolicy**: This specifies the quality of storage for files in this SA. Values can be `custodial` (high quality, typically on tape), `output` (medium quality, typically redundant disk storage), or `replica` (low quality, files may be lost if a disk fails). New in 1.3.
- **GlueSAAccessLatency**: This specifies how quickly files stored in this SA are guaranteed to be available for use. Values can be `online` (files are always on disk and can be read immediately), `nearline` (files may not be immediately accessible, e.g. on tape, and may need to be staged in before access), or `offline` (files may need manual intervention to make them accessible). New in 1.3.
- **GlueSAExpirationMode**: The policy for expiration of files in this SA. Values can be `neverExpire`, `warnWhenExpired` (a warning is generated when the lifetime is exceeded), or `releaseWhenExpired` (files will be deleted automatically when the lifetime is exceeded). Note that currently WLCG/EGEE does not expect files to expire. New in 1.3.
- **GlueSACapability**: A multivalued string allowing arbitrary capabilities/properties to be advertised. New in 1.3.

- **GlueSAVOInfo object**

- The **GlueSAVOInfo** object allows the specification of VO-specific information for an SA which supports multiple VOs. This may also be used for subgroups or roles within a VO. New in 1.3.
- **GlueSAVOInfoPath**: A VO-specific path which supercedes the **GlueSAPath** if present.
- **GlueSAVOInfoTag**: A VO-defined string which allows an SA to be selected according to the type of file being stored.
- **GlueSAVOInfoAccessControlBaseRule**: This defines a subset of the users specified by the **GlueSAAccessControlBaseRule** for whom this **VOInfo** object applies.

F.4.5. Attributes for the CE-SE Binding

The CE-SE binding schema represents a means for advertising relationships between a CE and one or more SEs. This is typically for CEs and SEs at the same site, but this is not required. In any case the relationship is defined and published by the site hosting the CE. The implication is that files on the SE(s) can be accessed quickly from WNs composing that CE compared with general file access over the WAN. It may also imply access for protocols like `rpio` which restrict access using host-based authorisation. Among other things, the WMS uses the `CESEBind` information to schedule jobs with input files specified in the JDL, to ensure that the jobs go to CEs from which the files are rapidly accessible.

Historically the `CESEBind` was also used to advertise an NFS mount point from which files on an SE were directly visible from WNs. However, this is not currently supported in WLCG/EGEE.

- **GlueCESEBind object**

- `GlueCESEBindCEUniqueID`: The unique ID for the CE.
- `GlueCESEBindSEUniqueID`: The unique ID for the SE.
- `GlueCESEBindWeight`: If multiple SEs are bound to a CE this allows a preference order to be expressed (larger numbers are preferred). This is not generally used in WLCG/EGEE at present.