DataGrid

EDG Users' Guide

Integration Team (WP6)

Document identifier:	**DataGrid-06-TED-0109-1-0**
Date:	**January 15, 2003**
Workpackage:	**Integration Team (WP6)**
Partners:	**Contributions from all partners**

Abstract: This guide provides basic information about how users can get started with the EDG Testbed, simple example uses of the testbed, and links to more detailed information.

Change Log

Version	Date	Comment
1.0	15 Jan 2003	Version For EDG 1.4.3

Contents

List of Tables

List of Figures

1 Overview

The Grid makes widely distributed computing resources transparently available to the end-user. As well as purely computational resources, these include data storage and networking. The European DataGrid (EDG) collaboration builds software components which enables this access and the EDG testbeds are the vehicles for testing of the EDG software.

There are three testbeds: the development testbed, the certification testbed, and the application testbed. The last, the application testbed, is intended for semi-production use by end-users. The others are key components of the development and certification process.

The resources of the testbed are organized into a number of sites and each site contains a number of various types of machines.

User Interface (UI) This machine runs the User Interface software which allows the end-user to interact with the EDG testbed. This is typically the machine the end-user logs into to submit jobs to the grid and to retrieve the output from those jobs.

Computing Element or Service (CE) A computing element consists of one gatekeeper node and one or more worker nodes. Together these provide computational resources to the user.

- **Gatekeeper (GK)** This is the frontend of a computing element. This node handles the interaction with the rest of the grid environment—accepting jobs, dispatching them for execution, and returning the output. This provides a uniform interface to the computational resources it manages.
- **Worker Node (WN)** These nodes sit behind a gatekeeper and are typically managed via a local batch system. The details of this are hidden from the end-user by the gatekeeper; however, these are the nodes on which user computations are actually performed. Consequently, the end-user software is installed on these nodes. These nodes do not run any EDG daemons, but do have client APIs for accessing EDG services and information.

Storage Element (SE) These nodes provide uniform access to large storage spaces. The storage element may control large disk arrays, mass storage systems and the like. This element hides the details of the backend storage systems and provides a uniform interface to the grid user.

The resources within a testbed site and the total number of sites change over time as new resources are added to the testbed or are temporarily withdrawn for reasons such as maintenance.

There are also several nodes which provide shared services and are not site-specific but shared by various subgroups of the testbed users. The most visible are the following:

Resource Broker (RB) These machines accept jobs from users (via the User Interface), match the jobs' requirements to the available resources at the various sites within the testbed, and dispatch them.

Replica Catalog (RC) These machines maintain a database of the locations of master copies of files and the locations of any replicas for a Virtual Organization (see Section 2.3). They do not hold the actual data only the database describing them. These machines are used by users and grid services to locate appropriate copies of input data files.

Monitoring & Discovery Service (MDS) This node is the top-level collection point for all information published in the testbed. MDS servers running on testbed resources collect information about those resources and publish this information into the MDS system.

Information Index (II or BDII) The Information Index is a node which caches the information in the MDS for use by the Resource Broker and by end-users.

There are numerous other services which support, for example, the EDG security model, but which are used only indirectly by end-users.

2 Getting Started

This is a very brief summary on starting to use the EDG Testbed. The rest of the guide and the referenced documentation contain important details which a prudent reader will at least browse before starting. Before using the EDG Testbed resources you must do the following:

1. You must obtain a crypographic certificate from an EDG-approved Certification Authority (CA). (See Section 2.1.)
2. With your certificate loaded into a web browser, you must sign the EDG Usage Guidelines and register with at least one virtual organization. These can be done via the user registration page on the Testbed website[1]. See Section 2.3 to choose an appropriate Virtual Organization.
3. You must obtain an account on a machine which has the software to access the EDG testbed (a User Interface machine). If one is not available locally at your home institute, you may request an account on the User Interface machine at CERN. Contact the system administrators[2], if necessary.

The following sections provide details of these prerequisites.

2.1 Obtaining a Certificate

Cryptographic certificates are used to attest to the identity of a user or machine to the extent specified in the issuing Certification Authority's (CA) policy documents. Users accessing EDG resources must have a valid certificate; similarly, hosts operating within the testbed must also have one.

For users, a certificate is the grid-equivalent of a passport. As such it is *personal* and should not be shared with anyone else. You should also ensure that the private key is kept private and that you choose a secure passphrase.

The EDG-approved CAs have service areas which cover most of Europe and the United States. (Consult the current list[3] on the web.) If a user or site is not covered by an existing CA's service area, then one must either start a new CA or negociate with the French CA to extend its service area[4].

Note that the certificate application and delivery procedures for each of the CAs varies. Refer the your CA's web site for a description of it's procedures.

2.2 Installing User Certificates

2.2.1 Installing for use with Globus

The DataGrid relies on the Globus Security Infrastructure (GSI) to implement certificate management. To use the GSI you must have your certificate in PEM format. Follow the instructions in Appendix B if you need to change a P12-formatted certificate into a PEM-formatted certificate. You should then place the two files `usercert.pem` and `userkey.pem` into a `.globus` directory in your home area. The file permissions for the `userkey.pem` file should be 0600, for the other 0644 is appropriate.

[1] http://marianne.in2p3.fr/
[2] mailto:hep-project-grid-testbed-managers@cern.ch
[3] http://marianne.in2p3.fr/datagrid/ca/ca-table-ca.html
[4] Extending the service area is done only if the request involves a small number of certificates and there is someone to act as a registration authority for those certificates.

You may place your certificate and key in a non-standard location, but in this case you must define the two environmental variables X509_USER_CERT and X509_USER_KEY to point to your certificate and key, respectively before calling **grid-proxy-init** (see Section 2.4).

2.2.2 Importing Certificates into a Browser

Signing the EDG Usage Guidelines must be done via a SSL-protected web form. To gain access to this page you must have your certificate loaded into a web-browser. The procedure for loading a certificate into a browser varies greatly between browsers. Instructions are given here for two popular browsers.

Netscape To import a certificate into Netscape, first start Netscape. Then select "Communicator→Tools→Security Info" from the top menus and then click on the "Certificates/yours" section. This will open a window where you will find an "Import a Certificate" button. You will be asked a new password to protect your Netscape Certificate Database (keep track of it as you will need it to import other certificates). You will be able to choose your `.p12` file from a standard file selection window.

Internet Explorer Go to the Tools menu and select Internet Options. Choose the Content tab and click on Certificates. In the new window click on Import to get the Import wizard, then follow the instructions. Select the file for import and give the password (it should go into the "Personal" certificate store).

You should also select high security otherwise Internet Explorer remembers your pass phrase for you.

2.3 Virtual Organizations

Virtual Organizations (VOs) are used to organize the testbed users into various subgroups and are the basis for grid *authorization*. When a user runs a task, the user's certificate information is compared with a file which is populated by information from the various VOs. "John Doe" may have been added to the Alice VO, in which case the file referred to will have an entry for "John Doe" along with a directive to map his requests onto a local Alice environment. On the other hand, John would not be allowed to run jobs under other environments.

The current list of virtual organizations[5] can be found on the web. If you did not register with a virtual organization when you signed the EDG Usage Guidelines (or wish to change your VO membership), then you must contact the VO manager directly.

If none of the listed virtual organizations is appropriate for you, use the WP6 VO. It is intended as a catch-all for folks who are not members of any of the others.

Note: With the Testbed 1 software, membership in more than one virtual organization is not supported. If you are a member of more than one virtual organization (with the same certificate), then the actual VO used at a particular site will depend on the order the VOs are listed in the site's configuration file. There is no mechanism by which the user can indicate which virtual organization should be used.

If you really need different roles in the Testbed 1 context, you should request multiple certificates (with different subject names) and register the different subject names with different virtual organizations. The Usage Guidelines must be signed with each of the certificates.

2.4 "Logging into the Grid"

To access the resources of the EDG Testbed, you must have a machine available to you which has the User Interface tools installed. Ask your local site administrator if there is such a machine at your site. If not,

[5] http://marianne.in2p3.fr/datagrid/vo/vo-table.html

you may request an account on the User Interface machine at CERN. Contact the system administrators[6], if necessary.

Your access rights to the grid services are tied to the subject name of your certificate. In essence, this is your "grid user name". Access is controlled via a proxy which is a time-limited credential signed by your private key. Grid services may take actions on your behalf if they have a copy of this proxy.

If you have installed your certificate correctly, the command **grid-proxy-init** will create a new proxy for you. A successful attempt will look similar to the following:

```
$ grid-proxy-init
Your identity: /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
Enter GRID pass phrase for this identity: **********
Creating proxy ............................................. Done
Your proxy is valid until Tue Aug 13 03:15:11 2002
```

By default, the proxy will have a lifetime of 12 hours. Proxies with different lifefimes can be generated using the **-hours** option if desired. Note that you expose yourself to a greater chance of having your credentials hacked if you generate a proxy with a long lifetime.

If the certificate has not been installed correctly, then you will see a "user certificate not found" error. Check that you have followed the instructions correctly in Section 2.2.1. An incorrect passphrase will result in a "wrong pass phrase" error.

If you want to destroy explicitly the proxy before the time has expired, you can use the command **grid-proxy-destroy**; the simpliest form takes no arguments. However this only destroys (erases) the local copy of the proxy. It does not affect copies of your proxy in use by your jobs or by grid services.

To obtain information about a generated proxy, you can use the command **grid-proxy-info** with an *-all* option which returns all of the information about the proxy:

```
$ grid-proxy-info -all
subject  : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr/CN=proxy
issuer   : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
type     : full
strength : 512 bits
timeleft : 11:36:17
```

Individual elements can be selected with other command options.

For all of these commands, you can obtain more usage information by using the *-help* option.

The proxy is a plain file stored by default in the `/tmp` area of the machine. For sites with a large number of workstations, it may be more convenient to store the proxy on a shared file system. This will allow you to generate your proxy once and use it on all of the workstations. To do so, define X509_USER_PROXY in your shell startup file to point to a file in the shared file system. For example,

```
export X509_USER_PROXY=~/.globus/proxy
```

for sh-shell variants. You can then use **grid-proxy-init** as usual and it will generate the proxy in this location.

Once created, the proxy is copied automatically, when necessary, by the various grid commands. This allows various services to act on your behalf.

[6] mailto:hep-project-grid-testbed-managers@cern.ch

3 Grid Information

The aim of the Information and Monitoring Service is to deliver a flexible infrastructure that provides information on the EU DataGrid itself and grid applications. At present it is predominantly used by the Resource Broker to query the resources available on the grid in order to assist it when deciding where to submit a job.

EDG currently uses Globus MDS (Monitoring & Discovery Service) which is built on OpenLDAP. The Lightweight Directory Assess Protocol (LDAP) offers a hierarchical view of information in which the schema describes the attributes and the types of the attributes associated with data objects. A useful, annotated subset of the schema can be found in Appendix E.

A number of information providers have been produced by EDG. These include Site information, Computing Element, Storage Element and Network Monitoring scripts. At the lowest end of the MDS hierarchy, these scripts are invoked by LDAP servers (Grid Resource Information Server or GRIS) running on the resources. In the intermediate levels of the hierarchy, Grid Information Index Servers (GIIS) group the information by site, country, etc. The highest level of the hierarchy is a single node from which all of the testbed resource information can be found. Figure 3.1 shows a vertical slice of the information system structure.

As MDS is based on LDAP, queries can be posed to the current Information and Monitoring Service using LDAP search commands. (See Appendix F for information on using LDAP queries.) An LDAP search consists of the following components:

```
$ldapsearch\
  -x\
  -H ldap://lxshare0373.cern.ch:2135\
  -b 'Mds-Vo-name=local,o=grid'\
  objectclass=ComputingElement\
  CEId FreeCPUs \
  -s sub

Explanation of fields
-x simple authentication
-H ldap://lxshare0373.cern.ch:2135: uniform resource identifier
-b 'Mds-Vo-name=local,o=grid': base distinguished name for search
'objectclass=ComputingElement': filter
CEId FreeCPUs: attributes to be returned
-s base|one|sub: search scope for just the base object, one-level,
                 or the complete subtree
```

Alternatively you can use MapCentre[1] to view the current status of the DataGrid Information and Monitoring Service.

While a query can be directed to any level of the MDS hierarchy, typically the top-level node is used. For the application testbed this is lxshare0373.cern.ch on port 2135. The Resource Broker actually consults another top-level node, the BDII (Berkeley Database Information Index), which acts as a cache for the MDS information. It runs on lxshare0225.cern.ch, port 2170. The BDII can also be searched directly and is preferred for efficiency. The base distinguished name for both of these servers is 'Mds-Vo-name=local,o=grid'.

[1] http://ccwp7.in2p3.fr/mapcenter/

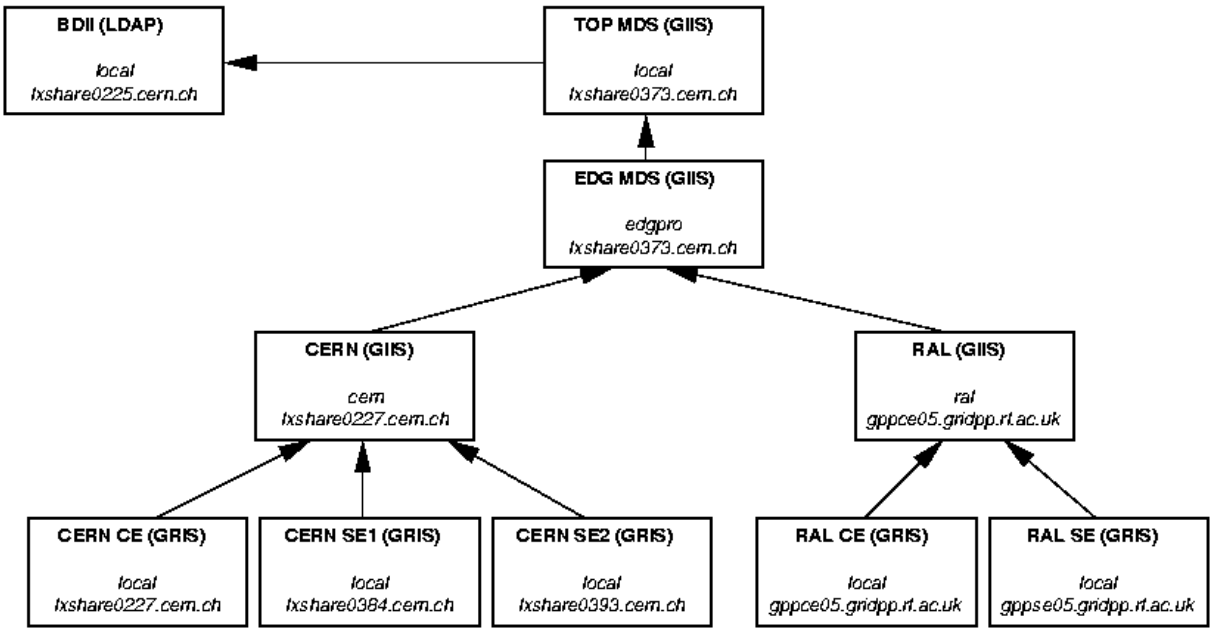

Figure 3.1: Information system hierarchy. The grid resources (along the bottom of the figure) publish state information into the MDS system. The intermediate GIIS layers buffer and group the information. All of the grid's state information is available from the top-level MDS node. The BDII buffers this information for the Resource Broker.

4 Job Submission

The job submission system functions as a large batch system with commands to submit a job, check its status, and to retrieve any output. The two main differences between a standard batch system (such as PBS, LSF etc.) and the EDG job submission service are that the job submission service:

1. adds a high-level layer permitting uniform access to the resources at different sites, and
2. matches the available resources to the requirements for the job automatically.

Uniformizing the user's interface and matching resources and requirements makes it easier for an user to fully exploit available resources.

The summary below is a brief overview of the main features of the job submission system; Figure 4.1 gives a similarly brief pictorial overview. For further information, see the Workload Management Documentation[1].

4.1 Job Submission Commands

The relevant commands for submitting a job, querying its status, retrieving the output, and cancelling a job are

```
dg-job-submit <job.jdl>
dg-job-status <jobId>
dg-job-get-output <jobId>
dg-job-cancel <jobId>
```

respectively. The submit takes as input as job description file (see below). The submit returns a jobId which can then be used to query the status of the job and retrieve its output.

The job identifiers produced by the workload management software are of the form:

```
https://lxshare0381.cern.ch:7846/137.138.181.214/1438441242444459?lxshare0381.cern.ch:7771
```

Note particularly that this contains a question mark (?) which is interpreted as a wild card in some shells (e.g. csh variants). For these shells the job identifier *must be quoted* when passing it as a command parameter.

The other commands take jobids as input and perform the corresponding action on the listed jobs. The **dg-job-get-output** pulls back the output from the job from the resource broker machine where it is cached; the output can only be retrieved when a job has reached the "OutputReady" status.

4.2 Job Description File

The key to the job submission and resource matching process is the job description file. This file describes the necessary inputs, generated outputs, and resource requirements of a job using the Job Description Language (JDL).

A typical example of a job description file:

[1] http://www.infn.it/workload-grid/documents.html

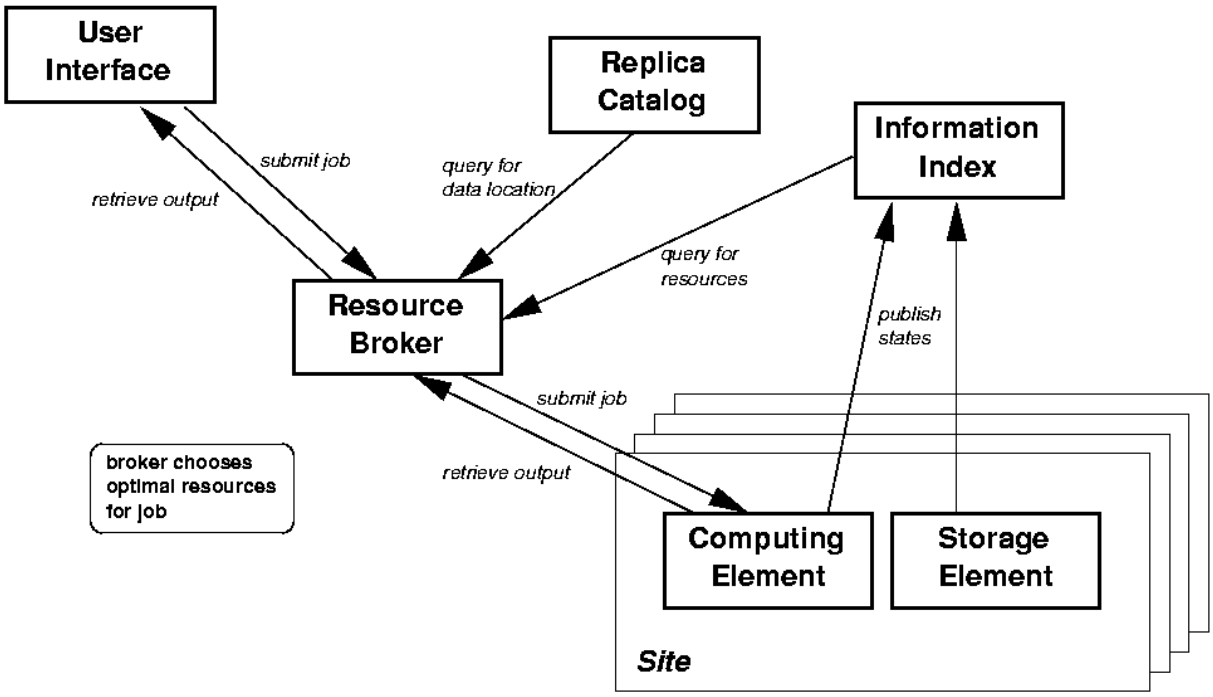

Figure 4.1: Job submission and execution. The user interacts with the grid resources via the User Interface by contacting a Resource Broker. When a user submits a job, the broker collects information for matchmaking, submits the job to the selected resource, and collects the results on completion. The broker caches the job output for later retrieval by the user.

```
Executable    = "HelloScript.sh";
Arguments     = "hello 200";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"HelloScript.sh"};
OutputSandbox = {"std.out","std.err"};
Requirements  = other.LRMSType=="PBS" && other.OpSys=="RH 6.2";
Rank          = other.FreeCPUs;
```

shows that a script is passed as input to a job, which then produces standard output and error files which will be transported back to the user (eventually with a **dg-job-get-output**).

The input and output sandboxes are intended for relatively small files (on the order of a new megabytes) like scripts, standard input, and standard output streams. If you are using large input files or generating large output files, you should instead directly read from or written to a storage element. *Abuse of the input and output sandboxes can fill the storage on the ResourceBroker and cause the broker to crash.*

The two parameters—Requirements and Rank—control the resource matching for the job. The expression given for the requirements specifies the constraints necessary for a job to run. In this case, a site running PBS and RH Linux 6.2 is required. The job will only be submitted to resources which satisfy this condition. If more than one resource matches, then the rank is used to determine which is the most desirable resource and hence the one to which the job is submitted. (Higher values are more desirable.) Both of these can be arbitrary expressions which use the fields published by the resources in the MDS system.

JDL is based on the Condor ClassAds library. More information about the supported functions and the

syntax of these expressions can be found in the ClassAds documentation[2].

ClassAds are extensible and place no restrictions on the parameter names. A side-effect of this is that misspelled parameter names are still syntatically valid, but will not act as expected. *Be extremely careful spelling the JDL parameter names.*

Using parameters in the JDL file one can steer jobs to sites which have a copy of a particular input file:

```
InputData = {"LF:file1.txt"};
ReplicaCatalog = "ldap://lxshare0226.cern.ch:9011/lc=test0, rc=Testbed1 Replica Catalog,dc=lxshare0226,dc=cern,dc=ch";
DataAccessProtocol = {"file", "gridftp"};
```

The ReplicaCatalog parameter is only required when InputData contains at least one logical filename, i.e. it can be omitted if only physical filenames are used. (The URL for your VO's replica catalog can be obtained from your VO.) Similarly, one can also select a site with a particular storage element:

```
OutputSE = "gppse05.gridpp.rl.ac.uk";
```

For more details, see the job submission examples (Section 4.4), further examples in the Data Management Section (Section 8), and the Workload Management Documentation[3].

It is often useful to check the results of the resource matching without submitting a job. For this, one can use the **dg-job-list-match** command. Given the JDL file it will return a ranked list of matching resources. The highest-ranked resource will appear first.

4.3 Long-lived Jobs

It is possible that long jobs may outlive the validity of the initial proxy; if so and the proxy is not renewed, the job will die prematurely. To avoid this the workload management software allows the proxy to be renewed automatically if your credentials are managed by a proxy server.

To use the automatic proxy renewal mechanism, first register a proxy with the MyProxy server using the command

```
myproxy-init -s <server> -t <hours>  -d -n
```

where "server" is the server address[4], "hours" is the number of hours the proxy should be valid on the server.

As this proxy is only copied to the server, you will need to create a local short-lived proxy using **grid-proxy-init** to do the job submissions. The resource broker will retrieve renewed proxies from the myproxy server for jobs which need them.

Information about your stored proxy can be obtained via the command

```
myproxy-info -s <server> -d
```

and the proxy can be removed with

```
myproxy-destroy -s <server> -d.
```

Once the proxy is removed from the server, running jobs will no longer receive renewed credentials.

[2] http://www.cs.wisc.edu/condor/classad/

[3] http://www.infn.it/workload-grid/documents.html

[4] The host lxshare0375.cern.ch is the MyProxy server associated with the CERN resource brokers. Contact the administrators of the other resource brokers to find other MyProxy servers.

4.4 Examples

Example 4.1 (Hello World) The simpliest job is simply a job which echos "Hello World" to the standard output. To run this example prepare a file "hello.jdl" which contains the following:

```
Executable    = "/bin/echo";
Arguments     = "Hello World";
StdOutput     = "std.out";
StdError      = "std.err";
OutputSandbox = {"std.out","std.err"};
```

Note that the complete path of the command is given and that the standard output and standard error are specified in the output sandbox.
Submitting the job produces the following output:

```
$ dg-job-submit HelloWorld.jdl

Connecting to host lxshare0381.cern.ch, port 7771
Logging to host lxshare0381.cern.ch, port 15830


****************************************************************************************************
                                    JOB SUBMIT OUTCOME
 The job has been successfully submitted to the Resource Broker.
 Use dg-job-status command to check job current status. Your job identifier (dg_jobId) is:

 - https://lxshare0381.cern.ch:7846/137.138.181.214/152312203546264?lxshare0381.cern.ch:7771

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

Note that this returns the job identifier associated with this job (the string starting with "https").
Using **dg-job-status** with the job identifier above yields the following output:

```
dg_JobId                =    https://lxshare0381.cern.ch:7846/137.138.181.214/152312203546264?lxshare0381.cern.ch:7771
Status                  =    OutputReady
Last Update Time (UTC)  =    Mon Aug 12 15:24:14 2002
Job Destination         =    tbn01.nikhef.nl:2119/jobmanager-pbs-qshort
Status Reason           =    terminated
Job Owner               =    /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
Status Enter Time (UTC) =    Mon Aug 12 15:24:14 2002
```

which has been edited to show only the interesting parts. When the status was requested, the job had finished and the output had been pushed back to the resource broker. States seen in the normal processing of jobs are: Accepted, Waiting, Running, Done, and OutputReady. Abnormal execution usually ends with an "Aborted" status.
To retrieve the output, use the **dg-job-get-output** command with the job identifier as the argument. The following is returned:

```
$ dg-job-get-output  https://lxshare0381.cern.ch:7846/137.138.181.214/152312203546264?lxshare0381.cern.ch:7771


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

 Output sandbox files for the job:
 - https://lxshare0381.cern.ch:7846/137.138.181.214/152312203546264?lxshare0381.cern.ch:7771
 have been successfully retrieved and stored in the directory:
 /tmp/152312203546264

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

where the output has again been edited. The important information is the location of the output files. Within this directory, `/tmp/152312203546264` in this case, will be the `std.out` and `std.err` files specified

in the output sandbox. The `std.out` file should contain the string "Hello World". The `std.err` file is empty in this case but would contain anything written to the standard error.

The contents of the JDL file transformed by many different programs during the submission process. A side effect of this is that special characters in the arguments attribute must be preceeded by triple '\', e.g.:

```
Executable = "/usr/bin/tail";
Arguments = "-f file1\\\&file2";
```

and quotes in the arguments must be escaped with the '\' character, e.g.:

```
Executable = "/bin/grep";
Arguments = "-i \"my name\" *.txt";
```

to be on the safe side or better create a script as in the next example.

Handling the job identifiers directly quickly becomes tedious. To avoid the need to handle directly the identifiers, the **dg-job-submit** will append the job identifier to a named file when using the *-o* option. On the other side, the job submission commands which take job identifiers as an argument accept the *-i* which allows the job identifier to be read from a file.

Note that the job management system does not limit the rate of jobs submitted either by a single user or in total. The broker does not respond well to high rates and has a limit of 512 concurrently running jobs. Continuous serial submissions are fine for short periods of time keeping in mind that the 512 concurrent jobs are for all users of the broker.

Example 4.2 (Hello from Script) The next example simply sends a small script with the job, executes it and returns the results. Create an executable file called `HelloScript.sh` which contains the following:

```
#!/bin/sh
/bin/echo "Hello From Script"
/bin/ls 9485968.txt
```

This will simply echo the given phrase on the standard output and put an error into the standard error (unless you have a file named `9485968.txt` on the machine on which your job runs).

The appropriate JDL file for this job is the following.

```
Executable    = "HelloScript.sh";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"HelloScript.sh"};
OutputSandbox = {"std.out","std.err"};
```

If the script is not in the current directory, then you must give the full path in the input sandbox line.

If the job executed correctly, then the standard output and standard error should contain the following

```
Hello From Script
```

and

```
/bin/ls: 9485968.txt: No such file or directory
```

respectively.

The executable named in the JDL will automatically be set with executable permissions. Other executable files may need to have the permissions set explicitly.

Important note: The input and output sandboxes are intended for relatively small files (few megabytes) like scripts, standard input, and standard output streams. If you are using large input files or generating large output files, you should instead directly read from or write to a storage element. *Abuse of the input and output sandboxes can fill the storage on the ResourceBroker and cause the broker to crash.*

Example 4.3 (Specifying Job Requirements) By specifying job requirements, the user can steer the job to sites which have the resources necessary to run the job correctly. Incompletely specifying the requirements may cause the job to fail, wasting both the resources and the user's time.

The requirements are specified with a "Requirements" attribute in the JDL description of the job. This value of this attribute is a boolean expression which specifies the necessary constraints. Nearly the full set of C operators and syntax are supported.

The values (or variables) which can be used in the requirements expression can be found by looking at the ComputingElement attributes in the information system (see Appendix E).

```
ldapsearch -x \
           -H ldap://lxshare0373.cern.ch:2135 \
           -b 'mds-vo-name=local,o=grid' \
           '(objectclass=computingelement)'
```

Attributes which are typically used are "Architecture", "OpSys", "RunTimeEnvironment", "MaxCPUTime", "MaxWallClockTime" and "FreeCPUs". Most of the attributes are self-explanatory.

To express that a job requires at least 25 minutes of CPU time and 100 minutes of real time, the expression:

```
Requirements = other.MaxCPUTime>=1500 && other.MaxWallClockTime>=6000;
```

would limit the matching to viable sites. The times are given in seconds. Note that the attribute names are prefixed with "other."; this is a remnant of the ClassAds syntax on which JDL is based. Note also that the values are *not* quoted. Using quotes around a numeric value will result in a string comparision which will produce an erroneous match (or none at all).

The "RunTimeEnvironment" is usually used to describe application software packages which are installed on a site. For example,

```
Requirements  = Member(other.RunTimeEnvironment,"ALICE-3.07.01");
```

will choose a site with the "ALICE-3.07.01" tag defined. The "RunTimeEnvironment" is a multi-valued attribute and evalues to a list. The "Member" function returns true if the given value is in the list.

Occasionally, one may wish to exclude or include a site manually. Forcing a job to a site can be accomplished with the *-resource* option of the **dg-job-submit** command. However, this entirely bypasses the matchmaking process and will not produce a `.BrokerInfo` file (see the Workload Management Documentation[5] for information on the BrokerInfo file) with the matchmaking results. Instead one can use the matchmaking with a clause like[6]

```
Requirements = other.CEId=="ccgridli03.in2p3.fr:2119/jobmanager-bqs-A";
```

[5] http://www.infn.it/workload-grid/documents.html

[6] A "CE" in this context is a batch queue. A "CEId" is a concatenation of the host, port, type of batch system, and queue name.

to do the same thing. More interestingly one can select or exclude a site:

```
Requirements  = RegExp(".*nikhef.*",other.CEId);
Requirements  = (!(RegExp(".*nikhef.*",other.CEId)));
```

which cannot be accomplished with the *-resource* option. Note that the JDL is very picky about the logical "not" syntax. Many sites also define a RunTimeEnvironment variable which identifies the site.

In the UI configuration file (`/opt/edg/etc/UI_ConfigENV.cfg`) there is a requirements clause which is added to all JDL files by default. By default this is "other.Active". Users may create a UI configuration file of their own to specify habitual requirements (or to choose an alternate resource broker, etc.). To use a custom UI configuration file set the EDG_WL_UI_CONFIG_PATH variable to the full path name, or specify the *-c* option when submitting a job with the **dg-job-submit** command.

Example 4.4 (Ranking Resources) If more than one resource matches the specified requirements, then the highest-ranked resource will be used. If the "Rank" attribute is not specified in the user's JDL description, then

```
Rank = -other.EstimatedTraversalTime;
```

will be used by default (also specified in the UI configuration file). The estimated traversal time is the expected time in seconds that a job will take to begin executing at the site.

This ranking is not always ideal, and the user may wish to choose some other criteria for the ranking. For example,

```
Rank = other.FreeCPUs;
```

will choose the site with the largest number of free CPUs. The rule to remember is that the larger the rank, the more desirable the resource is. If more than one site has exactly the same rank, then the one which is used is chosen randomly.

5 Job Environment

Currently the environment seen by jobs on grid resources contains very little "grid" customization. The standard PATH and LD_LIBRARY_PATH variables will be set allowing access to typical grid client commands. In addition, the variables EDG_LOCATION and GLOBUS_LOCATION will be set to the top of the EDG and Globus software trees.

Notably there will be no environment specific to your virtual organization setup automatically. As there is no way currently to transmit your virtual organization membership automatically, you must explictly setup this environment. A VO-customized environment is required for many of the data management tasks.

To bootstrap the environment setup, your job should first execute the command:

```
eval `\$EDG_LOCATION/bin/edg-vo-env --shell=sh atlas`
```

where "atlas" is replaced by the name of your virtual organization and "sh" is either "sh" or "csh" for sh-like or csh-like shells, respectively. If the shell is unspecified, "sh" is assumed. After execution the variables ATLAS_ROOT_DIR, GDMP_CONFIG_FILE, and RC_CONFIG_FILE will be defined. Your virtual organization may also require additional setup in which case you will be required to source a script under the ATLAS_ROOT_DIR area.

Information about the matchmaking process is provided in the `.BrokerInfo` file which is accessible from the environment variable EDG_WL_RB_BROKERINFO. A precise example on using this file can be found in Chapter 8.

6 GridFTP

6.1 File Transfers

There is often a need to manually transfer files from one node to another. The tool for doing so is **globus-url-copy**; the older **gsincftp** commands are deprecated by Globus and no longer part of the EDG release. The command syntax

```
globus-url-copy [options] sourceURL destURL
```

is rather simple and the *-help* option sufficiently explains the command's options.

Unfortunately, the inline help does not list the protocols supported or give examples of the syntax. The supported protocols are: file, gsiftp, and http. Examples are:

```
file:///home/loomis/stuff.txt
gsiftp://testbed011.cern.ch/~/stuff.txt
gsiftp://testbed011.cern.ch//tmp/stuff.txt
http://marianne.in2p3.fr/datagrid/documentation/daemon-guidelines.html
```

For the file protocol only absolute file names are accepted. All of the URLs must be fully specified, e.g. you cannot omit the file name on the output URL. For the gsiftp protocol, the tilde syntax can be used to specify a home area.

One great advantage of **globus-url-copy** is its ability to make third-party transfers. This allows transfers between two remote machines without having to funnel the data through your own machine. This avoids copying the data twice and is especially important if you are executing the command from a machine with a slow network connection or with insufficient disk space.

6.2 Client Commands

There are a set of GridFTP client commands available to do simple management of directories and files in a GridFTP server. These commands are:

```
edg-gridftp-exists
edg-gridftp-ls
edg-gridftp-mkdir
edg-gridftp-rename
edg-gridftp-rm
edg-gridftp-rmdir
```

and perform the equivalent of their unix namesakes. The only unusual one, **edg-gridftp-exists**, checks for the existence of a file or directory. Man pages for each of these commands list the valid options and arguments. All of the commands also recognize the *--help* option.

Note: The **edg-gridftp-rename**, **-rm**, and **-rmdir** commands should only be used on files which are *not* managed by a higher-level service such as the Replica Manager. Using them in this situation may destroy the coherency of the replica catalog database.

7 File Replication and Cataloguing

The main purpose of a Data Grid is to work on data, store it on permanent storage devices and retrieve it later on. A Replica Catalogue provides the means to catalogue files (replicas). Replica management tools take care of secure and efficient data movement as well as the necessary file cataloguing. Note that only files that are stored on a Storage Element can be catalogued in a Replica Catalogue. This section gives details on their usage and where to find relevant detailed information.

7.1 Replication Use Cases

Here we point out a few use cases in order to motivate the usage of replication and data management tools. The list is by far not complete but outlines the basic use cases.

1. "Bring a file into the Grid"

   One of the main questions asked is "How do I get a file into the Grid where it can be found by other Grid users?". Although a file can be created on a Computing Element, this is not the only way by which a file can be entered into the Grid, i.e. be stored on a Storage Element and registered with the Replica Catalogue Service.

   - A Grid user called Griddy first needs to discover which Storage Elements she has enough privilege to write to. This information can be obtained from the Information Service or the Replica Catalogue Service.
   - Next, Griddy needs to find a storage location for instance a directory within the Storage Element. A valid storage location might be lxshare0222.cern.ch/home/testfiles.

   Griddy has a file on her local computer and wonders how to get the file into the Storage Element. A minimal requirement is that she has some DataGrid software (e.g. Globus security and **globus-url-copy**) installed on her computer. **globus-url-copy** or edg-replica-manager.copyFile can then be used to copy the file from the local computer to the Storage Element (see Use Case 3). However, the file is still not visible in the Replica Catalogue Service and needs to be registered using a method edg-replica-manager.registerEntry (details see below).

   In the example above, there are a few possible problems:

   - Griddy might not be known to the SE even if she has rights to write through her VO
   - consistency is not automatically ensured between the Storage Element, the Replica Catalogue Service and the Replica Metadata Catalogue
   - her local computer might not meet all requirements to have access to the Grid
   - having two operations increases the possibility of committing an error

   We now assume that the Replica Manager software is available on the local machine and we combine both methods to a single one called edg-replica-manager.copyAndRegisterFile. Although we used the example of a local computer, the method edg-replica-manager.copyAndRegisterFile can also be used to atomically transfer a file from a Computing Element to a Storage Element and register it with the Replica Catalogue Service. The administrative rights on the file are transferred from Griddy to the Replica Manager which then administers the file on the SE.

2. Query existing files in the Grid

   Unlike in a conventional file system, files registered in a Grid and stored on Storage Elements, special Replica Catalogues need to be used to query information about the storage location and other possible file attributes like file size. A Replica Catalogue provides logical and physical filenames. In the EDG testbed, the Replica Catalogue command line tool or the API can be used. We recommend to use the RC interface only for retrieving information from the RC but not to store replica info. For storing and registering replica info use the specific tools edg-replica-manager or GDMP.

3. Copy a file from a Computing Element (Worker Node) to a Storage Element

   Grid applications (jobs) are executed on Computing Elements, in particular on the Worker Nodes within a Computing Element. We assume that either a single Worker Node or the entire Computing Element has some local storage. Thus, jobs can write data to the local data stores within the Computing Element and later transfer the data (also called output files) to Storage Elements where they can be registered with the Replica Catalogue Service. There are several ways to achieve this task:

   - use the **edg-replica-manger** command line tool or API (see below)
   - copy the file via **globus-url-copy** to the SE and then use a Replica Catalogue tool to register the file (not recommended; can cause inconsistencies)

4. Replicate from StorageElement to Storage Element.

   A file is already located at one Storage Element and registered with the Replica Catalogue Service and needs to be replicated to another Storage Element at a remote site. Once it has been determined that the user has proper access rights and the file has been successfully copied to the destination Storage Element, the Replica Catalogue Service needs to be updated with the location of the new replica. All of these operations are being performed as an atomic operation using the method edg-replica-manager.replicateFile, that should be used for this purpose.

5. Replicate a Set of Files between Storage Elements

   Imagine a High Energy Physics experiment that writes several hundred or even thousand files per day in one site and wants to replicate these file sets to remote sites or Regional Centres. In such a case it is best to use a tool that is capable of replicating a set of files rather than individual files. GDMP is the tool to be used for such a use case.

6. Mass Storage System at the Storage Element

   If large amounts of data are used at Storage Element, Mass Storage Systems are used commonly for staging files between disk and tape locations. Conventionally, a GridFTP server can only access files on disk but not on tape locations. GDMP provides a Mass Storage System interface with which it can stage files to/from disk if they are requested from a remote site. In the case, the GridFTP server can still access files since GDMP puts them in the correct location.

The main differences between GDMP and the edg-replica-manager (RM) have already been pointed out above, but Table 7.1 gives a summary. Based on the differences above one shall decide which tool to use.

7.2 GDMP, edg-replica-manager and Replica Catalogue Details

On the EDG testbed, you can find the following three replicating packages:

- GDMP (Grid Data Mirroring Package)
- edg-replica-manager

Table 7.1: Comparison of GDMP and Replica Manager

GDMP	RM
Replicates sets of files.	Replicates single files.
Replication between SEs	Replication between SEs; CE to SE
Mass storage interface	
File size as part of logical file attribute.	
Subscription system for newly published data.	
Notification for events like successful replication, publication.	
CRC file size check after transfers.	
Support for Objectivity.	

- Replica Catalogue (edg_rc_*)

Here we briefly outline the packages and then refer to the more specific sections below as well as to their user guides.

7.2.1 GDMP

The GDMP client software package is installed on the User Interface as well as on the ComputingElement and its worker nodes. Before you use GDMP, make sure your GDMP_CONFIG_FILE environment variable points to your VO specific GDMP config file. On the User Interface you need to set this variable explicitly to the following location:

```
/opt/edg/etc/VO/gdmp.conf
```

where *VO* corresponds to your specific VO.

The most basic commands for the use cases above are:

gdmp_register_local_file: registers a file on an SE

gdmp_publish_catalogue: publishes file information to the RC and remote SEs

gdmp_replicate_get: retrieves a file from a remote SE and stores it on the local SE

gdmp_replicate_put: retrieves a file from the local SE and stores it on a remote SE

gdmp_job_status: used to check the progress of **gdmp_replicate_get** or **put**

For more details consult the GDMP user guide (html[1], pdf[2], ps[3]).

7.2.2 edg-replica-manager

This package is a prototype of a replica manager (RM). Please note that a new RM (Reptor) for testbed release 2.0 will be provided and edg-replica-manager should serve only for getting first user feedback. It does not provide all the necessary functionality yet but is a good starting point for getting experience with a Grid Replica Manager and the use of logical file names.

[1] http://www.cern.ch/GDMP/userguide/gdmp-3-0/
[2] http://www.cern.ch/GDMP/userguide/userguide-gdmp-3.0.pdf
[3] http://www.cern.ch/GDMP/userguide/userguide-gdmp-3.0.ps

The package is installed on the User Interface as well as on Computing Elements and Worker Nodes where it can be found in `/opt/edg/bin/edg-replica-manager`. Similar to GDMP, an environment variable called RC_CONFIG_FILE needs to point to a configuration file of your VO specific RC LDAP server.

The basic command line tools and C++ APIs are as follows:

edg-replica-manager-registerEntry: registers a file in the Replica Catalogue

edg-replica-manager-unregisterEntry: unregisters a file from the Replica Catalogue

edg-replica-manager-copyFile: copy a registered file from one location to another; does *not* update RC

edg-replica-manager-copyAndRegisterFile: copy a file to an SE and register it

edg-replica-manager-replicateFile: replicate a file between two SEs

edg-replica-manager-deleteFile: delete file from RC and from disk

Note that the edg-replica-manager does not check for file size but puts the size 0 into the replica catalogue. That's a current restriction but will be removed for Reptor in testbed 2.0.

For more detailed usage refer to the Installation and User Guide (pdf[4], ps[5]).

7.2.3 Replica Catalogue Client

A replica catalogue provides a mapping from a Logical File Name to one or more Physical File Names (replicas) on disk. All replicas that can be registered in the RC need to be stored on disks of a Storage Element. Note files located on Computing Elements cannot be registered in an replica catalogue.

Normally, replication tools like GDMP or the edg-replica-manager take care of the correct interaction with the RC for inserting new entries. A client command line interface or RC API to the RC should be used to retrieve file location information.

Like edg-replica-manager, the RC command line tools are installed under `/opt/edg/bin` on the User Interface as well as CEs and WNs. The environment variable RC_CONFIG_FILE needs to be set.

The most basic commands are:

```
edg_rc_addPhysicalFileName
edg_rc_deleteLogicalFileAttribute
edg_rc_deleteLogicalFileName
edg_rc_deletePhysicalFileName
edg_rc_getLogicalFileAttributes
edg_rc_getLogicalFileName
edg_rc_getPhysicalFileNames
```

For details consult Section 10 "Replica Catalogue C++ API and Command line tools" of the GDMP user guide (html[6], pdf[7], ps[8]).

[4] http://www.cern.ch/grid-data-management/edg-replica-manager/edg-replica-manager-1.0.pdf
[5] http://www.cern.ch/grid-data-management/edg-replica-manager/edg-replica-manager-1.0.ps
[6] http://www.cern.ch/GDMP/userguide/gdmp-3-0/
[7] http://www.cern.ch/GDMP/userguide/userguide-gdmp-3.0.pdf
[8] http://www.cern.ch/GDMP/userguide/userguide-gdmp-3.0.ps

8 Data Management

This section describes the facilities for data access and management in release 1.4, which are significantly different from the 1.1 release (and will change again in testbed 2). The basic functionality needed to work is present, but there are some problems and users need to be prepared to work around them in some cases.

Data access is a major part of the DataGrid project, and involves most pieces of the middleware. There are three main areas: storing and registering new files, finding and reading existing files, and moving and replicating files. There is a general assumption that files are never modified once written (updates should be done by creating a new file with a different name).

Files can be stored on local disks on a User Interface (UI) machine or within a Computing Element (CE), but to be known to the Grid they have to be stored on a Storage Element (SE). This is an interface to storage systems, either on disk or tape, although as described below tape file access is indirect in 1.4. The files also have to be registered in a Replica Catalogue, with a unique name which refers to all replicas of the file.

There are detailed sections on each of the main areas, with pointers to the full documentation. Following this is a section of worked examples which highlight common techniques and some of the typical problems.

8.1 File Access

An SE can potentially support many access protocols, but at the moment only three are available:

1. "gridftp" is the basic Globus file transfer protocol. It puts grid certificate authentication on top of the standard FTP protocol, and also adds extra facilities like multiple connection streams and variable buffer sizes to optimise the transfer rate.

2. "file" access means that files can be accessed on a CE as though they were local, usually using a distributed file system like NFS or AFS. This access is only available from CEs which are defined to be "close" to the SE; usually that means they are on the same site (see below). At the moment the "close" relation between CEs and SEs is hard-coded by the local system manager, but in future it may be decided dynamically on the basis of network monitoring information.

3. "rfio" means the CERN RFIO package, a library which allows remote access to files, but again usually only on the local site for security reasons (there is currently no GSI authentication so it relies on the local Unix account mapping). There is also a command-line copy tool (**rfcp**) and a directory listing command (**rfdir**). These are aimed at access to the Castor tape store, but there is an RFIO server on each disk-based SE and the RFIO commands can also be used with disk files.

A note on "closeness": the information published by each site contains a list of SEs which are considered close to each CE, and conversely each SE has a list of close CEs. These are currently hard-wired by the system managers; normally closeness will mean that the machines are on the same site, but that isn't enforced, and nor is it required that the CE-SE relation has to be symmetrical. In general the assumption should be that there is a high-bandwidth connection between close elements. There is no direct closeness relation between SEs, but it is likely to be reasonable to assume that SEs which are close to the same CE are close to each other. An SE may export its files, usually by NFS, to a close CE, but this is not mandatory; "file" access is only available if this is done. Closeness is defined for each CE, so it is possible that an SE will be close to some CEs on a site and not others. "file" access is never available to non-close

SEs. The job broker considers closeness when assigning jobs which have input or output file requirements (see below).

A user application has to decide which protocol(s) to support, and there are pros and cons with each. gridftp access is available to any site, as long as an IP connection is available, but it normally requires the application to copy the file explicitly using the **globus-url-copy** command line tool. (Globus also provides an API which can be used to drive gridftp directly from a program, but that requires a significant amount of coding[1].) "file" access is completely transparent and requires no copying, but will only work if all files are stored on a single site and the same site has computing resources available for the job. RFIO has similar properties, with the additional capability of access to tape stores.

There does not appear to be any web documentation for **globus-url-copy**, but the built-in help gives:

```
globus-url-copy [options] sourceURL destURL
OPTIONS
          -help | -usage
               Print help
          -v | -version
               Print the version of this program
          -a | -ascii
               convert the file to/from netASCII format to/from local file format
          -vb | -verbose
               during the transfer, display the number of bytes transferred
               and the transfer rate per second
          -b | -binary
               Do not apply any conversion to the files. *default*
          -s  <subject> | -subject <subject>
               Use this subject to match with both the source and dest servers
          -ss <subject> | -source-subject <subject>
               Use this subject to match with the source server
          -ds <subject> | -dest-subject <subject>
               Use this subject to match with the destionation server
          -tcp-bs <size> | -tcp-buffer-size <size>
               specify the size (in bytes) of the buffer to be used by the
               underlying ftp data channels
          -bs <block size> | -block-size <block size>
               specify the size (in bytes) of the buffer to be used by the
               underlying transfer methods
          -p <parallelism> | -parallel <parallelism>
               specify the number of streams to be used in the ftp transfer
          -notpt | -no-third-party-transfers
               turn third-party transfers off (on by default)
```

The source and destination files need to be given as full URLs. The usual transports will be file: (local files) and gsiftp: (gridftp), although http: is also supported. The command can do direct third-party transfer, i.e. you can give gsiftp URLs on two remote SEs and the data will be transferred directly between them without going through the local machine (useful for dialup connections!). The most useful option is *-p*, which transfers files using multiple tcp streams to reduce the impact of packet loss. For large files try *-p 16*. An example of a simple copy of a local file to the CERN SE on lxshare0219 is:

```
globus-url-copy -vb file:‘pwd‘/testfile \
                gsiftp://lxshare0219.cern.ch/flatfiles/SE1/iteam/burke/testfile
```

[1] http://www-unix.globus.org/api/c/globus_ftp_client/html/index.html

(the *-vb* option gives a progress report on the transfer).

There does not seem to be a proper manual for RFIO anywhere, but the CERN Castor user guide[2] includes information on it.

The most useful commands are **rfcp** and **rfdir**, analogous to **cp** and **ls**. File names in Castor are of the form `/castor/cern.ch/path/to/file`. Files on a local SE can be referred to with names of the form `hostname:pathname`, e.g. `lxshare0393:/flatfiles/SE00/iteam/HelloWorld`. There are also commands **rfrename**, **rfstat**, **rfmkdir** and **rfrm**.

At present, access control on SEs is through the Unix gid (group ID). Users normally get anonymous accounts which belong to a Unix group according to VO membership, and SE permissions are supposed to be set so that anyone in the group has write permission. Certificate-based access control is likely to be used in future releases. Also, at CERN and potentially other sites with tape stores, users can be mapped to a fixed local account to get tape privileges associated with that account. Other users continue to get an anonymous account. Users need to be aware that anonymous accounts may be recycled at any time, so there is no guarantee of continued access to a file using the uid.

One potential problem is that neither GridFTP nor RFIO supports a **chmod** command, and gatekeeper access to an SE is deprecated and will be removed in the next release. This means that the only way for a user to change file permissions is through an NFS mount on a CE, and if there is no such mount the permissions cannot be changed.

8.2 File Naming

All files are identified by a Logical File Name (LFN), which must be unique within a VO. In release 1.4 files can only have a single LFN, aliases will be introduced in future releases. File replicas (identical copies) are identified by a Physical File Name (PFN) which consists of the SE hostname and the full file path on disk (tape systems are not yet directly supported, see below). In the current implementation the PFN is always formed by concatenating a fixed storage path on the SE with the LFN. Put another way, each SE has a storage directory for each VO, at a file path ending with the name of the VO, and the LFN of a file is the full pathname beyond that directory, including subdirectory names.

As an example, the CERN SE lxshare0393 has its general storage area at `/flatfiles/SE00`, and the storage directory for the Atlas VO is therefore `/flatfiles/SE00/atlas`. If the physical file name of a file is `lxshare0393.cern.ch/flatfiles/SE00/atlas/dir/subdir/file` the LFN of the file is `dir/subdir/file`. If the file is replicated to another SE it will always exist in the same directory tree under the same name, but the file path ahead of the VO directory may vary, e.g. at RAL the PFN will be `gppse05.gridpp.rl.ac.uk/flatfiles/05/atlas/dir/subdir/file`.

Future releases will introduce the concept of a "master" copy of a file, but in the current release all replicas are treated equally. Replicas can be deleted by users as required, but it's the responsibility of the user to ensure that the last replica is not deleted unless the file is no longer wanted.

8.3 File Catalogues

There are two pieces of software to manage files: GDMP, which deals with replicating batches of files between SEs, and a Replica Manager (RM) which can copy, register and replicate individual files. GDMP has more functionality but is more complex to use. In release 1.4 there is a single Replica Catalogue (RC) for each VO based on the Globus implementation, basically an LDAP directory. (GDMP also has various internal file catalogues.) There is currently no automated replica management, the process has to be controlled explicitly by the user.

[2] http://it-div-ds.web.cern.ch/it-div-ds/HSM/CASTOR/DOCUMENTATION/USERGUIDE/

Table 8.1: Contact URLs for Replica Catalogs

alice	ldap://grid-vo.nikhef.nl:10489/lc=Alice WP1 Repcat,rc=AliceReplicaCatalog,dc=eu-datagrid,dc=org
atlas	ldap://dell04.cnaf.infn.it:9011/lc=Atlas Lc1,rc=ATLAS Testbed1 Replica Catalog,dc=dell04,dc=cnaf,dc=infn,dc=it
biome	ldap://grid-vo.nikhef.nl:10389/lc=BioMedical WP1 Repcat,rc=BioMedicalReplicaCatalog,dc=eu-datagrid,dc=org
cms	ldap://grid011g.cnaf.infn.it:9211/lc=test0,rc=CMS Testbed1 Replica Catalog,dc=grid011g,dc=cnaf,dc=infn,dc=it
dzero	ldap://grid-vo.nikhef.nl:10589/lc=DZero EDG Repcat,rc=DZeroReplicaCatalog,dc=eu-datagrid,dc=org
eo	ldap://grid-vo.nikhef.nl:10689/lc=EarthOb WP1 Repcat,rc=EarthObReplicaCatalog,dc=eu-datagrid,dc=org
iteam	ldap://testbed014.cern.ch:12389/lc=TestCollection,rc=ITeamRC,dc=testbed014,dc=cern,dc=ch
lhcb	ldap://grid-vo.nikhef.nl:10889/lc=LHCb WP1 Repcat,rc=LHCbReplicaCatalog,dc=eu-datagrid,dc=org
tutor	ldap://grid-vo.nikhef.nl:10789/lc=EDGtutorial WP1 Repcat,rc=EDGtutorialReplicaCatalog,dc=eu-datagrid,dc=org
wpsix	ldap://testbed014.cern.ch:12389/lc=WPsixCollection,rc=WPsixRC,dc=testbed014,dc=cern,dc=ch

The main tool for registering and replicating files in release 1.4 is GDMP 3. This is substantially changed from GDMP 2, in particular it uses a client-server architecture and has direct support for multiple VOs. There will be further major changes in testbed 2, including a distributed Replica Catalogue and a much enhanced Replica Manager, with the GDMP functionality absorbed into the latter.

The manual for GDMP 3[3] is available in html, ps or pdf format, and there is a detailed user guide[4] includes information on commands to read and manipulate the RC directly, although in most cases this should not be necessary for normal use. For full use of GDMP there is no substitute for reading the documentation, but this document gives a basic summary.

The manual[5] for the RM commands documents a set of command line tools to examine a Replica Catalogue:

```
edg-replica-manager-ls
edg-replica-manager-pwd
edg-replica-manager-cd
edg-replica-manager-cat
```

Note that use of GDMP 2 in release 1.1 involved some manipulation of proxy certificates. That was a temporary fix and should *not* be done any more as it can compromise security.

Each VO has its own Replica Catalogue (RC). In the 1.4 production testbed this runs on a single machine, and hence is a single point of failure. Tests have exposed some limitations; if there are more than a few thousand files (depending on the length of the LFN) the client code fails to cope, and if there are a large number of simultaneous attempts to access an RC some of the accesses may fail. The catalogues are implemented using Globus software as an LDAP directory. Use of the catalogues generally needs an ldap URL as a contact string. For GDMP this is provided in the default configurations, but for other purposes users may need to know the value (see Table 8.1).

Note that these URLs may have embedded spaces which are significant. The URL is used in the JDL (see below). You can look at the RC content directly by using the URL in Netscape (not MSIE), using the Genius portal[6] and also with **ldapsearch** or **grid-info-search** (see below). The format is a machine name and port followed by a Distinguished Name (DN) which consists of a file collection (lc), a catalogue name (rc) and a string identifying the whole LDAP server (dc=, ...). You may need to include the lc in some uses and exclude it in others.

The RC contact information should also be published in the MDS, look for ObjectClass=ReplicaCatalogue. However, at the time of writing only the NIKHEF catalogues are published.

For most purposes use of the RC should be transparent. The most useful thing to do is to list all physical files associated with a given logical file name **edg_rc_getPhysicalFileNames -l** ⟨**LFN**⟩.

[3] http://project-gdmp.web.cern.ch/project-gdmp/documentation.html
[4] http://project-gdmp.web.cern.ch/project-gdmp/edg-testbed/Welcome.html
[5] http://grid-data-management.web.cern.ch/grid-data-management/edg-replica-manager/edg-replica-manager-1.2.pdf
[6] https://genius.ct.infn.it

It is possible to manipulate an RC using the Globus commands[7]. and there are some operations which can only be done with this (e.g. modifying the file path), but in general the RC should only be accessed via the EDG tools (EDG provide an API wrapper to the Globus commands, described in the GDMP manual). As of release 1.4 there are no checks of RC consistency, e.g. to see if a physical file listed in the RC has been deleted.

8.4 Output Files

As of release 1.4 the support for output files is somewhat limited. In the JDL you can target a particular SE with something like:

```
OutputSE = "gppse05.gridpp.rl.ac.uk";
```

which will attempt to direct the job to a CE which is "close" to the requested SE. In practice this is little different to targetting a job at a site in other ways. As things stand the user has to decide which SE to use, and check for themselves that their VO is supported and that there is enough space available. This should be available automatically in the JDL in future releases.

To check the properties of the various SEs you can use the following URL in netscape:

```
ldap://lxshare0373.cern.ch:2135/mds-vo-name=edgpro,o=grid??sub?objectclass=StorageElement
```

or an equivalent with **ldapsearch** or **grid-info-search**, e.g.

```
ldapsearch -x -h lxshare0373.cern.ch -p 2135 -b "mds-vo-name=edgpro,o=grid" \
           "objectclass=StorageElement"
```

NB **grid-info-search** is a wrapper around **ldapsearch**, but is only available on machines with Globus installed whereas **ldapsearch** is a standard part of Linux, as long as the OpenLDAP package is installed. Unfortunately there are two slightly incompatible versions of **ldapsearch**; leave out the *-x* option if you get an error message. Also note that the machine on which the MDS runs may change.

Use of LDAP URLs can be quite convenient, as they can be saved as bookmarks, but they don't work with Internet Explorer and some versions of Netscape seem to give inconsistent results. If you want to restrict the output you can give a list of comma-separated attribute names between the first two question marks. Conversely, if you leave out the objectclass specifier you get all objects.

SE information is fairly limited so far, but you can see the total disk space (for all VOs), a list of VOs supported with the file storage path, and the CEs to which the SE is close. You can get the total free space on the disk using an objectclass of StorageElementStatus.

Another thing lacking at the moment is indirect matching, so although the information system contains the free disk space for each SE it is not possible to specify a requirement in the JDL which says "pick a CE which is close to an SE which has more than some amount of free space". As things stand the only way to do such targetting would be to get the system managers to define RunTimeEnvironment variables on the CEs, e.g. "AtlasBigDisk" for sites with more than some amount of disk space reserved for Atlas.

At the moment the best way to put an output file into an SE is to write it locally and then copy it to an SE with gridftp using the **globus-url-copy** tool, e.g.

```
globus-url-copy file:‘pwd‘/testfile \
                gsiftp://gppse05.gridpp.rl.ac.uk/flatfiles/05/atlas/testfile
```

[7] http://www.globus.org/datagrid/deliverables/replicaGettingStarted.pdf

There are a few problems with this. One is that you have to hard-code the SE you want, at the moment the grid middleware can't tell you which one to use. You also need to know the directory where the file is stored. Each SE has a fixed storage directory, `/flatfiles/05/atlas` in this case, where all files are stored; as mentioned this is in the MDS, and you can also get it from the gdmp configuration on a particular SE with:

```
gdmp_job_status -S <SE> -c gdmp.conf | grep GDMP_STORAGE_DIR
```

Any directory or naming structure below this is defined by the VO.

If your job happens to be running on a "close" CE where "file" access is available the file can be copied or written directly with normal Unix commands. The path to use is published in the information system (use an objectclass of CloseStorageElement in the search strings given above) and is available to a job through the BrokerInfo API (see below). Again the file path has a directory for each VO below it. Files can also be written using RFIO. Note that the name of the NFS mount point need not be the same as the pathname on the SE, although in practice that is how most systems are configured.

Having written the file, it needs to be registered in the Replica Catalogue (see the manuals mentioned above). This is done using the **gdmp_register_local_file** and **gdmp_publish_catalogue** commands, or alternatively with **edg-replica-manager-registerEntry** or **edg-register-copyAndRegisterFile**. It is also possible to store files on SEs without registering them, in which case it is up to the application to keep track of where they are. However, in general this should be done in an area outside the one used by GDMP because there is no way to stop other users registering your files. It would need an arrangement with the local system manager to create such an area.

8.5 Input Files

The JDL has fairly complete support for input files. You need to give a list of file names, either physical names (the real name on an SE) or logical names (as known to the RC). In release 1.4 the logical name is just the physical name after the GDMP path, i.e. below the VO directory but including any subdirectories. Physical names start with the SE hostname (see the example below).

You also give the contact URL for your RC, and a list of file access protocols you are prepared to use. The RB tries to direct your job to the CE with the best access to the files you want. It should guarantee that you can access all the files somehow, but in fact it seems that in the 1.4 release it will match even if you ask for "file" access and the input files are not all on the same SE, in which case the job will not be able to read them all.
As an example:

```
InputData = {"LF:file1.txt", "PF:lxshare0219.cern.ch/SE1/iteam/file2.txt"};
ReplicaCatalog = "ldap://lxshare0226.cern.ch:9011/rc=Testbed1 Replica Catalog,dc=lxshare0226,dc=cern,dc=ch";
DataAccessProtocol = {"file", "gridftp"};
```

This means that the job wants two files, one specified by LFN and one by PFN, with the LFN being looked up in the specified RC. The job declares than it can access the files using either the "file" or "gridftp" protocols (deciding which to use at run time based on the BrokerInfo information as described below).

When your job runs, the submission software transfers some information about the decisions it makes in a file called `.BrokerInfo` (the full path to the file is stored in the variable EDG_WL_RB_BROKERINFO). The job can read the file itself, but there is also an API to get the information in a useful format. Essentially you call the API routines with your requested file names and protocols, and they tell you how you should access the file.

Note that the `.BrokerInfo` file is not created if you use the *-r* option with **dg-job-submit** to direct the job to a specific CE because this bypasses the brokering process completely. If you want to have the

`.BrokerInfo` file but also want to force the job to a specific site, use an appropriate Requirements expression in the JDL, e.g. on the CEId or the RunTimeEnvironment (most sites define a RunTimeEnvironment with the name of the site).

The BrokerInfo API is described in an appendix to the GDMP manual, and the full BrokerInfo interface is now explained in a separate manual[8]. Probably the most useful function is

```
getSelectedFile(LFN, protocol, TFN, filename)
```

where you give the LFN (logical name) of a file and the protocol you want to use, and you get back a TFN (Tranport File Name, e.g. a URL to use with **globus-url-copy**) and possibly a local file name. See the examples for more information on usage.

One issue here is that although the `.BrokerInfo` file contains a list of the LFNs requested in the JDL there is currently no API to return it. In many cases a job script will know which LFNs it needs, but a generic script will need some way to get the list of LFNs, e.g. passed in the JDL argument list or in a file sent with the input sandbox.

The command **edg-brokerinfo** provides a command line interface to the BrokerInfo API. The built-in help in the command gives:

```
Use: # edg-brokerinfo [-v] function [parameter] [parameter] ...
with -v flag you can have a verbose, formatted output
supported funtions are:
 > edg-brokerinfo getCE
 > edg-brokerinfo getDataAccessProtocol
 > edg-brokerinfo getInputPFNs
 > edg-brokerinfo getSEs
 > edg-brokerinfo getCloseSEs
 > edg-brokerinfo getSEMountPoint SE
 > edg-brokerinfo getLFN2PFN PFN
 > edg-brokerinfo getSEProto SE
 > edg-brokerinfo getPFNs
 > edg-brokerinfo getSEPort SE Proto
 > edg-brokerinfo getPhysicalFileNames LFN
 > edg-brokerinfo getTransportFileName PFN Proto
 > edg-brokerinfo getPosixFileName TFN
 > edg-brokerinfo getSelectedFile LFN Proto
 > edg-brokerinfo getBestPhisicalFileName PFN1 PFN2 ... ! Proto1 Proto2 ...
```

More detailed information can be found in the manual[9].

Note that the software does not do anything to provide you with the file itself, it only gives you information. If you use "file" or "rfio" access the job can read the file directly, but with "gridftp" you will normally have to copy the file to local storage first (and in some cases that may be a useful thing to do anyway). At the moment there is nothing to manage local disk space on a CE, so jobs will have to assume that they can use the working directory, or else have hard-coded information about scratch disks for each site.

8.6 Replicating and Deleting Files

Users currently have to initiate any file moving themselves. This can be done using the **gdmp_replicate_get** command to pull files to a particular SE. Note that this will only work if the SEs have been subscribed

[8] http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0135-0_0.pdf
[9] http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0135-0_0.pdf

to each other (**gdmp_host_subscribe**) in both directions, and the file catalogues have been published (**gdmp_publish_catalogue** for new files, **gdmp_get_catalogue** to refresh the existing list). It is also possible to configure GDMP to replicate files automatically, but this needs action by the system managers at the relevant sites. Alternatively the **edg-replica-manager-replicateFile** command can be used to replicate individual files.

In future releases there will be an intelligent Replica Manager to control replication of files between SEs, and in the long run it should be done automatically to optimise the use of both computing power and network bandwidth.

Replicas should be deleted using the replica management tools to ensure that the Replica Catalogue is updated. In addition, files which were registered or replicated with GDMP should be deleted with **gdmp_remove_local_file** to update the internal GDMP catalogues.

8.7 Mass Storage

Some mass storage (tape) systems present a disk-like interface to the user, and in this case the system can be integrated with the Grid tools in the same way as a disk-based SE. However, many systems have a specialised interface. In particular the Castor system at CERN uses the RFIO package for access, which is not Grid-enabled (a GridFTP interface is under development).

In release 1.4 there are two possible ways to interact with such systems. One is for user jobs to use the local commands as they would in a non-Grid environment. The RFIO client tools and libraries are installed on all CEs, so a job running at CERN can use e.g. **rfcp** to copy files to and from Castor (security considerations mean that access is only available on the CERN site). RFIO maps the user (uid) of the account running the client to the Castor user. For this reason, users with CERN tape accounts have a fixed mapping of their Grid identity to their CERN id when running jobs at CERN, rather than the usual mapping to a random "pool" account. Other users still have the pool accounts, which are mapped to Castor accounts with limited privileges (it should be possible to write to the directory `/castor/cern.ch/grid/`⟨`vo`⟩).

However, this access method has the limitation that it offers no integration with the Grid tools; files cannot be registered in a Replica Catalogue or replicated with GDMP or the replica manager commands. For this reason, an interim solution has been developed using the concept of staging between a disk-based SE and an associated mass storage system. Files on a disk SE can be staged to tape using a fixed mapping of the file names, e.g. a file called `/flatfiles/SE00/atlas/dir/subdir/file` on disk might be translated to `/castor/cern.ch/grid/atlas/dir/subdir/file` in Castor. Only the disk file is registered in the Replica Catalogue, but a program which knows the mapping can translate the disk name to the name in Castor, and the file can be accessed by non-Grid jobs like any other Castor file (assuming that permissions allow).

The name translation can be done in two ways. One is a simple mapping of the full file path, as in the example above. However, this may have problems if the mass storage filing system has limits on the length of file names, or on the characters they can contain. There is therefore an alternative mapping which uses a fixed-length MD5 hash of the full file name, giving file names like

```
/castor/cern.ch/grid/test/se/001/02b2bef6d82e13c7ea0b4aa489cf9a2f.1
```

These names are encoded and decoded automatically by the staging tools. The original filename is stored in a corresponding file with a `.c` extension. However, there is a loss of transparency with this method. In the 1.4 production testbed this hashing is disabled by default, although VOs can opt to use it if they choose.

The staging is done by the GDMP server, with the replica manager commands calling GDMP internally, so GDMP needs to be configured correctly even if it isn't otherwise being used. See the GDMP manual for details. Staging is implemented by calling out to stage-in and stage-out scripts defined in the gdmp.conf

file, which generally make calls to a generic staging script. Unfortunately the only way to know the details of the configuration, and in particular the staging location in the mass storage system, is to read the scripts.

Staging has been implemented at CERN for some time, although the exact configuration is subject to change. In addition there is a special SE (lxshare0384) with a very restricted grid map file which can be used to give VO production managers the right to stage files to a production area in Castor while barring access to most users, which gives a partial work-around to the lack of authorisation control. There is also software support in GDMP to allow staging to be restricted to a limited range of users on a normal SE, but this has not so far been used. More recently, staging has also been implemented at Lyon and RAL.

With the file on tape the disk copy can then be deleted to recover space. Jobs which want to access the file, either locally or to replicate to another site, then need to have the file staged back to disk before the access can succeed. GDMP and the replica manager commands have been enhanced with hooks to call staging scripts to allow this to happen.

At present, staging from disk to tape occurs when a file is either registered on or replicated to an SE with staging enabled. The relevant commands are **gdmp_register_local_file**, **gdmp_replicate_get**, **_put**, **edg-replica-manager-copyAndRegisterFile** and **edg-replica-manager-replicateFile**. The GDMP commands stage automatically, but the replica manager commands only do it if the parameter GDMP_STAGING is set to "yes" in the `rc.conf` file, which is not the default.

The replication commands also trigger staging back to disk if an attempt is made to replicate a file from an SE with staging enabled if the file is not found on disk. This occurs regardless of the `rc.conf` setting.

In addition, GDMP provides three commands to manage staging explicitly. **gdmp_prepare_open** can be called by a user job before it tries to access a file on disk. If the file is already on disk the command just returns, otherwise it attempts to stage the file from tape. There are also commands **gdmp_stage_to_mss** and **gdmp_stage_from_mss** which can be used to trigger staging in either direction by hand.

In release 1.4 there is no space management for SEs, so users need to check the space and delete files themselves if necessary. There are also no disk quotas so in most cases the space is shared between all VOs, although at some sites there may be separate NFS-mounted disks for each VO area.

8.8 Worked Examples

This section gives some simple use cases with working code. This can only be an introduction because the data management system is inherently complex, so for serious use the full documentation should be read. The examples assume that the user has an account on a UI machine and a grid certificate, and understands how to submit jobs and retrieve the output. They also, where relevant, assume that the account is on testbed010.cern.ch using the 1.4 production system as of December 2002, that the user is a member of the iteam VO, and uses the bash shell. In the text below, input commands are prefixed with "$" and output with ">". All the code has been tested, but there are no guarantees that it contains no bugs or that it will work in a particular environment.

One important point is that there are two sets of commands for data management. GDMP is a fairly sophisticated system designed to replicate large numbers of files to many sites, but it is somewhat complex to use. In addition there are "replica manager" commands which in the current release are basically wrappers around the Globus replica management tools. These are easier to use than GDMP but have substantially less flexibility. The examples below show the use of the replica manager commands first, and then the equivalent procedure with GDMP. In many cases there are several ways to achieve the same result, and the examples don't cover all possibilities.

The replica manager examples in the following text use the full command names, but the commands also have abbreviated forms, e.g. **edg-rm-r** for **edg-replica-manager-replicateFile**. See the manual for details.

8.8.1 Initial setting up

To use GDMP and replica manager commands you need to point the environment variables GDMP_CONFIG_FILE and RC_CONFIG_FILE to the right files according to the VO of which you are a member. The VO name is also used in various other places, e.g. file paths.

You can define the VO-related variables with

```
$ eval `$EDG_LOCATION/bin/edg-vo-env --shell=sh iteam`
```

where the shell can be sh or csh and the final argument is the name of the VO (on some UI machines this may be done automatically in a system profile script). The bin directory should normally be in the PATH so it should be sufficient to use the bare command name. This should set the variables:

```
$ env | grep CONFIG
>RC_CONFIG_FILE=/opt/edg/etc/iteam/rc.conf
>GDMP_CONFIG_FILE=/opt/edg/etc/iteam/gdmp.conf
```

For a job running on a CE this must currently be done at the beginning of the user job script, as there is no universal way for the system to determine the VO under which a job is running. In case of problems it's also worth checking that the files pointed to by the variables exist and have some sensible-looking content.

8.8.2 Finding Storage Elements

There are various ways to find information on which SEs are available. The SE which has been configured as being the default for a UI can be found by doing

```
$ cat /opt/edg/etc/gdmp.shared.conf | grep GDMP_LOCAL_HOST=
```

which should show the configured GDMP_LOCAL_HOST value:

```
>GDMP_LOCAL_HOST=lxshare0393.cern.ch
```

Alternatively, query the information system. This can be done with ldapsearch, e.g.:

```
$ ldapsearch -x -h lxshare0225.cern.ch -p 2170 -b "mds-vo-name=local,o=grid" \
  "objectclass=storageelement" | grep SEId:
>SEId: lxshare0393.cern.ch
>SEId: tbn03.nikhef.nl
>SEId: gppse05.gridpp.rl.ac.uk
>SEId: ccgridli04.in2p3.fr
```

but you need to know the right host, port and base DN to query (in general the host needs to be the Information Index associated with the Resource Broker to which you submit jobs, and the port and base DN are as shown above). There are also various web pages[10] [11] showing views of the system, but again you need to be aware of which information system they are querying.

You also need to check that the SE supports the VO of which you are a member. This can again be obtained from the information system in the SEvo item:

[10] http://testbed007.cern.ch/tbstatus-bin/infoindexcern.pl
[11] http://www.nordugrid.org/monitor/edgii.php

```
$ ldapsearch -x -h lxshare0393.cern.ch -p 2135 -b "mds-vo-name=local,o=grid" \
  "objectclass=storageelement" seid sevo | grep -i se[iv][do]:
>SEId: lxshare0393.cern.ch
>SEvo: alice:/flatfiles/SE00/alice
>SEvo: atlas:/flatfiles/SE00/atlas
>SEvo: cms:/flatfiles/SE00/cms
>SEvo: lhcb:/flatfiles/SE00/lhcb
>SEvo: biome:/flatfiles/SE00/biome
>SEvo: eo:/flatfiles/SE00/eo
>SEvo: wpsix:/flatfiles/SE00/wpsix
>SEvo: iteam:/flatfiles/SE00/iteam
>SEvo: tutor:/flatfiles/SE00/tutor
```

For each SE this gives a list of VOs and their corresponding storage directories (see below). Note that in this case the query is directed at the information system (GRIS) on the particular SE.

In any case, for further examples the SE on gppse05.gridpp.rl.ac.uk will be chosen to write the initial copy of the file, and lxshare0393.cern.ch will have a replica:

```
$ export SE1=gppse05.gridpp.rl.ac.uk
$ export SE2=lxshare0393.cern.ch
```

If you intend to use GDMP you can check that the servers are working and that you are authorised with **gdmp_ping**:

```
$ gdmp_ping -S $SE1
>Message: The local GDMP server gppse05.gridpp.rl.ac.uk:2000 is listening and
>you are an authorized user [ Wed Aug 21 14:29:02 2002 ]

$ gdmp_ping -S $SE2
>Message: The local GDMP server lxshare0393.cern.ch:2000 is listening and
>you are an authorized user [ Wed Aug 21 14:45:50 2002 ]
```

You also need to subscribe the SEs to each other:

```
$ gdmp_host_subscribe -S $SE1 -r $SE2
>Message: Local host subscribed to lxshare0393.cern.ch:2000 [ Thu Aug 22 15:17:54 2002 ]
$ gdmp_host_subscribe -S $SE2 -r $SE1
>Message: Local host subscribed to gppse05.gridpp.rl.ac.uk:2000 [ Thu Aug 22 15:18:18 2002 ]
```

This normally only needs to be done once and for all between each pair of SEs. You can check the subscription status with the -c option:

```
$ gdmp_host_subscribe -c -S $SE1 -r $SE2
>Message: Local host is already subscribed to lxshare0393.cern.ch:2000 [ Thu Aug22 15:19:55 2002 ]
```

8.8.3 Hello World

The data management equivalent of Hello World is to store and register a file on a Storage Element, and then submit a job to find and print it. First create a file in your working directory on a UI machine:

```
$ echo "Hello World" | cat > HelloWorld
```

Then you need to know where you can store files on the SE:

```
$ gdmp_job_status -S $SE1 -c gdmp.conf | grep GDMP_STORAGE_DIR
>GDMP_STORAGE_DIR=/flatfiles/05/iteam
```

This should end with the name of your VO. This information is also available from the information system, as decribed above.

You may want to create a subdirectory to separate your files from others. One way to do that is by running mkdir on the SE:

```
$ globus-job-run $SE1 /bin/mkdir /flatfiles/05/iteam/test
```

and to check:

```
$ globus-job-run $SE1 /bin/ls -al /flatfiles/05/iteam
>total 12
>drwxrwsr-x    3 gdmp     iteam        4096 Aug 21 13:34 .
>drwxrwsr-x   11 gdmp     root         4096 Aug 15 15:09 ..
>drwxr-sr-x    2 iteam005 iteam        4096 Aug 21 13:34 test
```

Note that the directory is not currently created group-writeable, which means that if your pool account gets recycled you lose control of it, so you may like to chmod it:

```
$ globus-job-run $SE1 /bin/chmod g+w /flatfiles/05/iteam/test
```

However, there are a couple of problems with this. It isn't possible to use **globus-job-run** from inside a running job, and in future releases it's likely that it will be disabled completely on SEs. To get around this there are now some command-line tools which use GridFTP to manipulate the SE file system. In this case there is a command

```
$ edg-gridftp-mkdir gsiftp://$SE1/flatfiles/05/iteam/test
```

There is a corresponding ls command:

```
$ edg-gridftp-ls --verbose gsiftp://$SE1/flatfiles/05/iteam/
>drwxrwsr-x    2 iteam006 iteam        4096 Jan  6 16:47 test
```

Other commands are **edg-gridftp-exists**, **-rename**, **-rm** and **-rmdir**, with the expected syntax. However, there is currently no chmod (although in this case **edg-gridftp-mkdir** has created the directory g+w).

In any case, now define a variable for the storage path:

```
$ export SP1=/flatfiles/05/iteam/test
```

Using the replica manager commands the file can be copied and registered in the Replica Catalogue in one step:

```
$ edg-replica-manager-copyAndRegisterFile -l test/HelloWorld \
  -s `hostname`/`pwd`/HelloWorld -d $SE1/$SP1/HelloWorld
>configuration file:  /opt/edg/etc/iteam/rc.conf
>logical file name:   test/HelloWorld
>source:              testbed010.cern.ch//shift/lxshare072d/data01/UIhome/burke/test1.4/HelloWorld
>destination:         gppse05.gridpp.rl.ac.uk//flatfiles/05/iteam/test/HelloWorld
>protocol:            gsiftp
>VO:
>The program was successfully executed.
```

This says that the file should be copied from the source file given by *-s* (which needs to be a full path including the hostname) to the destination file *-d*. In the current implementation the Logical File Name (LFN) given by *-l* must be the same as the full pathname of the file beyond the VO directory, so `test/HelloWorld` in this example. The destination can be just a hostname, in which case the path will be looked up in the information system automatically (sources can also be given as a hostname as long as the source is an SE, and not a UI as in this example).

Alternatively, GDMP can be used, although for the simple registration of a single file this is more complex. First copy the file using gridftp:

```
$ globus-url-copy file:‘pwd‘/HelloWorld gsiftp://$SE1/$SP1/HelloWorld
```

and check that it worked:

```
$ globus-job-run $SE1 /bin/cat $SP1/HelloWorld
>Hello World
```

(**globus-url-copy** should create files with g+w automatically.) Then use GDMP to register the file and publish it in the Replica Catalogue:

```
$ gdmp_register_local_file -S $SE1 -d $SP1
>Server Message [gppse05.gridpp.rl.ac.uk:2000]: A client has been started to register the requested files. [ Thu Aug 22 15:28:48 2002 ]
>Message: Server Log ID=gdmp_gppse05.gridpp.rl.ac.uk_4156_1030022927_1 [ Thu Aug22 15:28:48 2002 ]
```

This command is asynchronous, i.e. it starts a task on the SE and returns immediately. With the *-d* option it will register all new files under the specified directory; use *-p* to register a single file. The message includes a log file ID which can be read with **gdmp_job_status**:

```
$ gdmp_job_status -S $SE1 -L gdmp_gppse05.gridpp.rl.ac.uk_4156_1030022927_1
>(voluminous output omitted)
```

Check for completion by looking for the file in the local catalogue with:

```
$ gdmp_job_status -S $SE1 -c local_file_catalogue | grep HelloWorld
>file:gppse05.gridpp.rl.ac.uk_.flatfiles.05.iteam.test.HelloWorld:test/HelloWorld:12:2146730865:1030022784
```

If the file appears in the catalogue the registration is complete. Note that this contains the internal file ID:

```
gppse05.gridpp.rl.ac.uk_.flatfiles.05.iteam.test.HelloWorld
```

which can be used with **gdmp_replicate_get/_put** (see below). The file can then be published (this actually publishes *all* files which are registered but unpublished):

```
$ gdmp_publish_catalogue -C -S $SE1
>Message: Server Message [gppse05.gridpp.rl.ac.uk:2000]: Out of 2 host(s) 2 have
>received the published catalog [ Thu Aug 22 16:58:07 2002 ]
```

Note that the *-C* option assumes that the Replica Catalogue is using password authorisation (the password is known to the system and doesn't need to be supplied by the user). If we switch to using GSI authorisation the *-C* switch will not be needed. The command should confirm publication to the subscribed hosts and to the Replica Catalogue.

Regardless of the manner in which the file was registered, it should appear in the Replica Catalogue. This can be checked by looking up the LFN:

```
$ edg_rc_getPhysicalFileNames -l test/HelloWorld
>configuration file:     /opt/edg/etc/iteam/rc.conf
>logical file name:      test/HelloWorld
>gppse05.gridpp.rl.ac.uk/flatfiles/05/iteam/test/HelloWorld
```

Now for a job to find and read the file. Create a file called `HW.jdl`:

```
Executable = "World.sh";
Arguments = "test/HelloWorld";
StdInput = "/dev/null";
StdOutput = "HW.out";
StdError = "HW.err";
InputSandbox = "World.sh";
OutputSandbox = {"HW.out", "HW.err"};
InputData = "LF:test/HelloWorld";
ReplicaCatalog = "ldap://testbed014.cern.ch:12389/lc=TestCollection,rc=ITeamRC,dc=testbed014,dc=cern,dc=ch";
DataAccessProtocol = "file";
```

This says that the job will execute a shell script `World.sh` with argument `test/HelloWorld`, which is the LFN of the file. The script will be sent with the job, and stdout and stderr will be sent back.

The InputData requests the file as input, and the protocol is "file" so the script will expect direct file access. ReplicaCatalog is the URL of the RC for your VO, iteam in this case (note that some tools use the spelling "catalog" and others use "catalogue"). This can be obtained from one of the config files:

```
$ cat $RC_CONFIG_FILE | grep RC_LOGICAL_COLLECTION
>RC_LOGICAL_COLLECTION=ldap://testbed014.cern.ch:12389/lc=TestCollection,rc=ITeamRC,dc=testbed014,dc=cern,dc=ch
```

Then create the shell script:

```
#!/bin/bash
GSF=‘edg-brokerinfo getSelectedFile $1 file‘
TFN=‘echo $GSF | cut -d " " -f 2‘
echo Printing local file $TFN:
cat $TFN
```

This uses the BrokerInfo command line interface to get the local file name for the site chosen by the broker (in fact it returns the file name as a pseudo-URL followed by the "real" local file name, hence the cut to select the latter). If all goes well the file should be printed and sent back in the stdout file:

```
$ cat /tmp/172128290083990/HW.out
>Printing local file /flatfiles/05/flatfiles/05/iteam/test/HelloWorld:
>Hello World
```

Note that the storage path appears twice in the file name, this is a feature and not a bug!

8.8.4 Replicating a file

When a file is published it can be replicated to any SE. Using the replica manager command:

```
$ edg-replica-manager-replicateFile -l test/HelloWorld -s $SE1 -d $SE2
>configuration file:  /opt/edg/etc/iteam/rc.conf
>logical file name:   test/HelloWorld
>source:              gppse05.gridpp.rl.ac.uk
>destination:         lxshare0393.cern.ch
>protocol:            gsiftp
>Valid host !
>Valid host !
>source = gppse05.gridpp.rl.ac.uk//flatfiles/05/iteam/test/HelloWorld
>destination = lxshare0393.cern.ch//flatfiles/SE00/iteam/test/HelloWorld
>Destination Location Path /flatfiles/SE00/iteam
>The program was successfully executed.
```

Note that any subdirectories, `test` in this case, must be created on the destination SE first. The directory and file names must be the same on all SEs. This example illustrates the use of bare hostname for the source and destination.

To use GDMP for replication the SEs involved need to be subscribed to each other. If you want an SE which was not subscribed at the time of publication, or was down, you can update its catalogues with

```
$ gdmp_get_catalogue -S $SE2 -r $SE1
>Remote catalog file is successfully added. [ Mon Jan  6 18:40:28 2003 ]
```

To replicate all registered and unreplicated files from site 1 to site 2 use:

```
$ gdmp_replicate_get -C -S $SE2 -r $SE1
>Server Message [lxshare0393.cern.ch:2000]: A client has been started to replicate the requested files. [ Mon Jan  6 18:41:18 2003 ]
>Message: Server Log ID=gdmp_lxshare0393.cern.ch_14809_1041874877_1 [ Mon Jan  6 18:41:18 2003 ]
```

Note that this command is again asynchronous, it starts a process on the remote SE and returns immediately. The **gdmp_job_status** command with the *-L* option can be used to examine the log file (with the ID given in the response to the command), although this is generally long and fairly hard to interpret.

Replicating a single file

```
$ gdmp_replicate_get -C -S $SE2 -r $SE1 -i gppse05.gridpp.rl.ac.uk_.flatfiles.05.iteam.test.HelloWorld
>Server Message [lxshare0393.cern.ch:2000]: A client has been started to replicate the requested files. [ Mon Jan  6 18:44:05 2003 ]
>Message: Server Log ID=gdmp_lxshare0393.cern.ch_15182_1041875044_1 [ Mon Jan  6 18:44:05 2003 ]
```

requires the file ID. (See above for obtaining the file ID). It's also possible to use the *-f* option to filter the file names to be replicated. This will only replicate files in the relevant subdirectory and below:

```
$ gdmp_replicate_get -C -S $SE2 -r $SE1 -f $SP1
>Server Message [lxshare0393.cern.ch:2000]: A client has been started to replicate the requested files. [ Mon Jan  6 18:45:13 2003 ]
>Message: Server Log ID=gdmp_lxshare0393.cern.ch_15227_1041875113_1 [ Mon Jan  6 18:45:13 2003 ]
```

GDMP can also be configured to replicate files automatically, see the manual for details.

In either case the Replica Catalogue should be updated with the new location. One can check that this is the case with the command:

```
$ edg_rc_getPhysicalFileNames -l test/HelloWorld
>configuration file:     /opt/edg/etc/iteam/rc.conf
>logical file name:      test/HelloWorld
>gppse05.gridpp.rl.ac.uk/flatfiles/05/iteam/test/HelloWorld
>lxshare0393.cern.ch/flatfiles/SE00/iteam/test/HelloWorld
```

8.8.5 Deleting files

Files should be deleted with the same tool (replica manager or GDMP) used to register them. With the replica manager:

```
$ edg-replica-manager-deleteFile -l test/HelloWorld -s $SE1
>configuration file:  /opt/edg/etc/iteam/rc.conf
>logical file name:   test/HelloWorld
>source:              gppse05.gridpp.rl.ac.uk
>protocol:            gsiftp
>Valid host !
>source = gppse05.gridpp.rl.ac.uk//flatfiles/05/iteam/test/HelloWorld
>The program was successfully executed.
```

To use GDMP to delete files and remove the information from the various catalogues use **gdmp_remove_local_file**, e.g.:

```
$ gdmp_remove_local_file -C -S $SE1 -p $SP1/HelloWorld
>Message: Deleting /flatfiles/05/iteam/test/HelloWorld of type file [ Mon Jan  6 18:52:28 2003 ]
>Message: /flatfiles/05/iteam/test/HelloWorld deleted. [ Mon Jan  6 18:52:31 2003 ]
```

In either case the file should be deleted from disk and removed from the Replica Catalogue:

```
$ edg-gridftp-exists gsiftp://gppse05.gridpp.rl.ac.uk/flatfiles/05/iteam/test/HelloWorld
>error gsiftp://gppse05.gridpp.rl.ac.uk//flatfiles/05/iteam/test/HelloWorld does not exist
$ edg_rc_getPhysicalFileNames -l test/HelloWorld
>configuration file:     /opt/edg/etc/iteam/rc.conf
>logical file name:      test/HelloWorld
>lxshare0393.cern.ch/flatfiles/SE00/iteam/test/HelloWorld
```

Note that deleting an individual file and the RC reference to it does not delete the information on the LFN even if that was the last physical instance of the file. The **edg-replica-manager-deleteFile** command takes a *-a* option which deletes all replicas of a file and removes the LFN from the RC. GDMP can only delete individual files. There is also an **edg_rc_deleteLogicalFileName** command to explicitly remove an LFN, although such direct manipulation of an RC should be done with care.

8.8.6 Writing a file from within a job

At present there is not much specific support for output files. The JDL can have an OutputSE requirement, but all that does is target the job at a CE local to the SE, which you can do just as well in other ways. In general there are two options: you can write a file to a specified SE using **globus-url-copy**, or you can write to whatever SE is local to the CE on which the job runs.

One issue is disk space. Currently there are no disk quotas or other space management, so you can only check the overall space situation. This is published in the information system, and you can query a given SE with e.g.

```
$ ldapsearch -x -h gppse05.gridpp.rl.ac.uk -p 2135 -b "mds-vo-name=local,o=grid" \
             "objectclass=StorageElementStatus" | grep SEfreespace
>SEfreespace: 10001
```

The space (in megabytes) reported may be inaccurate on some systems, e.g. with multiple NFS-mounted areas.

To partially alleviate this problem there is now a cron job on most SEs which writes a text file in the VO storage area giving a "map" of the storage space:

```
$ globus-job-run lxshare0393.cern.ch /bin/cat /flatfiles/SE00/iteam/edg_query_vo_storage.dat
>78109300        /flatfiles/SE00/iteam
>78109300        TOTAL

$ globus-job-run lxshare0384.cern.ch /bin/cat /flatfiles/SE00/cms/edg_query_vo_storage.dat
91047584        /flatfiles/SE00/cms
74564168        /flatfiles/SE00/cms/st01
96638240        /flatfiles/SE00/cms/st02
97117228        /flatfiles/SE00/cms/st03
97596220        /flatfiles/SE00/cms/st04
97596220        /flatfiles/SE00/cms/st05
```

```
73425060        /flatfiles/SE00/cms/st06
85251052        /flatfiles/SE00/cms/st07
97596220        /flatfiles/SE00/cms/st08
97596220        /flatfiles/SE00/cms/st09
97596220        /flatfiles/SE00/cms/st10
1006024432      TOTAL
```

Note that the file should ideally be retrieved with **globus-url-copy** as the use of **globus-job-run** to an SE is deprecated, although in any case this is likely to be a temporary solution.

There is also currently no management of local space on a worker node. The information system claims to have a value MinLocalDiskSpace, but this is not meaningful. Jobs can write files in the working directory, which will in general be NFS/AFS mounted, but will have to deal with space problems themselves. Alternatively files can be written directly to a close SE using NFS/AFS. Jobs may also be able to write to `/tmp` which is likely to be on a local disk, but this method has no guarantees of any kind.

The first option, writing to a hardwired remote SE, is essentially the same as the UI-based recipe given above, although it is not possible to use **globus-job-run**. For the alternative, writing to whatever SE is "close" to the node on which the job runs, you can again use the BrokerInfo interface within the job script:

```
$ edg-brokerinfo getCloseSEs
```

This returns a list of local SE names, in general just take the first one:

```
$ CloseSE=‘edg-brokerinfo getCloseSEs | cut -d " " -f 1‘
```

You can get the NFS/AFS mount point (i.e. the place on the worker node where the SE disk area is mounted) with:

```
$ MP=‘edg-brokerinfo getSEMountPoint $CloseSE‘
```

Note that there is no guarantee that there is such a mount point, but in the current EDG testbed nearly all sites do in fact have one (to ensure this the JDL needs to ask for an input file with the "file" protocol as there is no way to ask for a protocol for an output file). In this case the file can be written directly, but note that the mount point excludes the VO name so you need to append that yourself. Alternatively, the file can still be written with **globus-url-copy** in the standard way. In either case, the file can then be registered and published as before.

Putting all this together gives a script like in Table 8.2 (for the GDMP case), which creates a file, registers it and then deletes it, printing various information along the way. Note the **chmod g+w**, the file needs to be group-writeable or gdmp will not be able to delete it. Table 8.3 shows the output of this script run at CERN.

8.8.7 Using the BrokerInfo C++ API

The BrokerInfo interface can also be used from C++. Table 8.4 shows a C++ version of the Hello World script. This can be compiled with:

```
g++ demo.C -g -o demo -I/opt/edg/include/BrokerInfo \
                      -I/opt/classads/include \
                      -I/opt/edg/include/ReplicaCatalog \
                      -L/opt/edg/lib -L/opt/classads/lib \
                      -lBrokerInfo -lclassad
```

Table 8.2: Example Script Using GDMP

```
#!/bin/bash

FileName=testfile
VO=iteam

eval `edg-vo-env --shell=sh $VO`

CloseSEList=`edg-brokerinfo getCloseSEs`
CloseSE=`echo $CloseSEList | cut -d " " -f 1`
echo Close SE is $CloseSE

MP=`edg-brokerinfo getSEMountPoint $CloseSE`
echo Local mount point is $MP

echo Creating file $FileName ...
echo "Test" | cat > $MP/$VO/$FileName
chmod g+w $MP/$VO/$FileName
ls -l $MP/$VO/$FileName

echo Check RC config file $RC_CONFIG_FILE:
cat $RC_CONFIG_FILE
echo Check GDMP config file $GDMP_CONFIG_FILE:
cat $GDMP_CONFIG_FILE
echo Check GDMP is running ...
gdmp_ping -S $CloseSE

export `gdmp_job_status -S $CloseSE -c gdmp.conf | grep GDMP_STORAGE_DIR`
echo GDMP path is $GDMP_STORAGE_DIR

echo Registering ...
gdmp_register_local_file -S $CloseSE -p $GDMP_STORAGE_DIR/$FileName
echo -n Waiting for registration to finish ...
i=0
until gdmp_job_status -c local_file_catalogue | grep -q $VO.$FileName || test $i -ge 10
   do i=$[$i+1]; echo -n .; sleep 30
done
echo

echo Publishing ...
gdmp_publish_catalogue -C -S $CloseSE

echo Checking the Replica Catalogue ...
edg_rc_getPhysicalFileNames -l $FileName

echo Deleting ...
gdmp_remove_local_file -C -S $CloseSE -p $GDMP_STORAGE_DIR/$FileName

echo Checking deletion:
ls -l $MP/$VO/$FileName

echo Checking the Replica Catalogue ...
edg_rc_getPhysicalFileNames -l $FileName
```

Table 8.3: Output of Example GDMP Script

```
Close SE is lxshare0393.cern.ch
Local mount point is /flatfiles/SE00
Creating file testfile ...
-rw-rw-r--    1 iteam005 iteam          5 Jan  6 19:54 /flatfiles/SE00/iteam/testfile
Check RC config file /opt/edg/etc/iteam/rc.conf:
<config file omitted>
Check GDMP config file /opt/edg/etc/iteam/gdmp.conf:
<config file omitted>
Check GDMP is running ...
Message: The local GDMP server lxshare0393.cern.ch:2000 is listening and you are an authorized user [ Mon Jan  6 19:54:33 2003 ]
GDMP path is /flatfiles/SE00/iteam
Registering ...
Server Message [lxshare0393.cern.ch:2000]: A client has been started to register the requested files. [ Mon Jan  6 19:54:34 2003 ]
Message: Server Log ID=gdmp_lxshare0393.cern.ch_25560_1041879273_1 [ Mon Jan  6 19:54:34 2003 ]
Waiting for registration to finish ...
Publishing ...
Message: Server Message [lxshare0393.cern.ch:2000]: Out of 2 host(s) 2 have received the published catalog [ Mon Jan  6 19:54:44 2003 ]
Checking the Replica Catalogue ...
lxshare0393.cern.ch/flatfiles/SE00/iteam/testfile
Deleting ...
Message: Deleting /flatfiles/SE00/iteam/testfile of type file [ Mon Jan  6 19:54:44 2003 ]
Message: /flatfiles/SE00/iteam/testfile deleted. [ Mon Jan  6 19:54:45 2003 ]
Checking deletion:
Checking the Replica Catalogue ...
error in entry_set_first = 0x1
```

and needs to be called with the logical file name, and optionally the protocol, as specified in the JDL. An example script is therefore:

```
#!/bin/bash
echo Compiling ...
g++ demo.C -g -o demo -I/opt/edg/include/BrokerInfo \
                      -I/opt/classads/include \
                      -I/opt/edg/include/ReplicaCatalog \
                      -L/opt/edg/lib -L/opt/classads/lib \
                      -lBrokerInfo -lclassad
echo
echo Running with LFN $1 and protocol $2 ...
./demo $1 $2
```

with JDL:

```
Executable = "demo.sh";
Arguments = "test/HelloWorld file";
StdInput = "/dev/null";
StdOutput = "demo.out";
StdError = "demo.err";
InputSandbox = {"demo.sh", "demo.C"};
OutputSandbox = {"demo.out", "demo.err"};
InputData = "LF:test/HelloWorld";
ReplicaCatalog = "ldap://testbed014.cern.ch:12389/lc=TestCollection,rc=ITeamRC,d
c=testbed014,dc=cern,dc=ch";
DataAccessProtocol = "file";
```

and the output from running at CERN is:

```
Compiling ...
```

Table 8.4: Example C-program Using BrokerInfo

```
#include <string>
#include <iostream>
#include "BrokerInfo.h"
#include "ReplicaCatalogB.h"

int main(int argc, char *argv[]) {

  if (argc < 2 || argv[1] == NULL) {
    cout << "Missing argument" << endl;
    exit(-1);
  }

  string lfn = argv[1];
  string protocol = "gridftp";
  if (argc > 2) protocol = argv[2];

  cout << "LFN is:       " << lfn << endl;
  cout << "Protocol is:  " << protocol << endl;

  string TFN, filename;
  BrokerInfo *bip;
  bip = BrokerInfo::instance();
  if (bip == NULL) {
    cout << "Can't create BrokerInfo interface" << endl;
    exit(-1);
  }

  string sOut;
  bip->getCE(sOut);
  cout << "BrokerInfo::getCE(): " << sOut << endl;

  ReplicaCatalogB RC;
  RC.getSelectedFile(lfn, protocol, TFN, filename);

  cout << "TFN is:       " << TFN << endl;
  cout << "File name is: " << filename << endl;

  if (filename != "") {
    string catfile = "cat " + filename;
    system(catfile.c_str());
  }

  return 0;

}
```

```
Running with LFN test/HelloWorld and protocol file ...
LFN is:       test/HelloWorld
Protocol is:  file
BrokerInfo::getCE(): lxshare0364.cern.ch:2119/jobmanager-pbs-short
TFN is:       file:///flatfiles/SE00/flatfiles/SE00/iteam/test/HelloWorld
File name is: /flatfiles/SE00/flatfiles/SE00/iteam/test/HelloWorld
Hello World
```

8.8.8 Using mass storage

Mass storage systems which use RFIO (e.g. Castor at CERN and HPSS at Lyon) can be used in the same way as they would be in a non-grid job, e.g. to copy a file from Castor to disk on the CERN SE:

```
globus-job-run lxshare0227.cern.ch /usr/bin/rfcp \
    /castor/cern.ch/grid/iteam/testfile /flatfiles/SE00/iteam/testfile
```

and to list a Castor directory:

```
globus-job-run lxshare0227.cern.ch /usr/bin/rfdir /castor/cern.ch/grid/iteam
```

The **rfcp** and **rfdir** commands can also be used directly from UI accounts at CERN as long as the user account maps to a CERN tape account. For security reasons the commands only work on machines at the same site as the tape system.

Alternatively, there is a staging system which automatically copies files between the CERN SE disk and Castor (and which can be set up to behave in the same way at other sites, e.g. Lyon and RAL). The exact configuration of this system is still changing, so check the current status with an expert if you need to rely on it.

Files which are replicated to the CERN SE, or which are registered there, using either GDMP or the replica manager commands, can be automatically copied to a particular directory in Castor, which is defined for each VO. With GDMP the staging is automatic; with the replica manager you need to use a modified `rc.conf` file with the parameter change GDMP_STAGING=yes. The following examples use the Atlas VO:

```
$ touch CastorTest

$ edg-replica-manager-copyAndRegisterFile -l CastorTest \
  -s 'hostname'/'pwd'/CastorTest \
  -d lxshare0393.cern.ch/flatfiles/SE00/atlas/castorTest \
  -c rc.atlas.stage.conf
>configuration file:  rc.atlas.stage.conf
>logical file name:   CastorTest
>source:              testbed010.cern.ch//shift/lxshare072d/data01/UIhome/burke/test1.4/CastorTest
>destination:         lxshare0393.cern.ch
>protocol:            gsiftp
>VO:
>Valid host !
>Calling GDMP staging script ...
>gdmp_stage_to_mss -S lxshare0393.cern.ch -P 2000 -l CastorTest -V atlas > /dev/null
>The program was successfully executed.
```

Note that the replica manager uses GDMP to do the staging. The file is staged to a fixed directory which is currently `/castor/cern.ch/grid/se/002`, although this may change. In general the user should not need to know it, but you can check that the file is there (in this case using rfio from a CERN CE so the command can be run from any UI):

```
$ globus-job-run lxshare0227.cern.ch /usr/bin/rfdir /castor/cern.ch/grid/test/se/002/CastorTest
-rw-r--r--   1 osynge   gr              0 Jan 07 12:41 /castor/cern.ch/grid/test/se/002/CastorTest
```

Note that the Replica Catalogue only lists the file on disk:

```
$ edg_rc_getPhysicalFileNames -l CastorTest
>configuration file:     /opt/edg/etc/atlas/rc.conf
>logical file name:      CastorTest
>lxshare0393.cern.ch/flatfiles/SE00/atlas/CastorTest
```

The file can now be deleted from disk (using **rfrm** for variety):

```
$ globus-job-run lxshare0227.cern.ch /usr/bin/rfrm \
  lxshare0393:/flatfiles/SE00/atlas/CastorTest
```

This still leaves the file recorded in the RC:

```
$ edg_rc_getPhysicalFileNames -l CastorTest
>configuration file:     /opt/edg/etc/atlas/rc.conf
>logical file name:      CastorTest
>lxshare0393.cern.ch/flatfiles/SE00/atlas/CastorTest
```

A job which wants to use the file needs to stage it back first. This can be done by always calling **gdmp_prepare_open** on a file on an SE with staging enabled (or on any SE) before using it:

```
$ gdmp_prepare_open -l CastorTest -S lxshare0393.cern.ch
>Prepare to open file at lxshare0393.cern.ch
>The file CastorTest has been succeessfully staged to MSS to /flatfiles/SE00/atlas at lxshare0393.cern.ch
```

Check that the file is now on disk:

```
$ globus-job-run lxshare0227.cern.ch /usr/bin/rfdir \
  lxshare0393:/flatfiles/SE00/atlas/CastorTest
>-rw-r--r--   1 osynge   atlas          0 Jan 07 12:55 lxshare0393:/flatfiles/SE00/atlas/CastorTest
```

Alternatively, if a file which is not on disk is replicated to another site it will be automatically staged back to disk first (note that this does not need the modified **rc.conf** unless you want to trigger staging to tape at the destination site):

```
$ edg-replica-manager-replicateFile -l CastorTest \
  -s lxshare0393.cern.ch -d gppse05.gridpp.rl.ac.uk
>configuration file:  /opt/edg/etc/atlas/rc.conf
>logical file name:   CastorTest
>source:              lxshare0393.cern.ch
>destination:         gppse05.gridpp.rl.ac.uk
>protocol:            gsiftp
>Valid host !
>Valid host !
>source = lxshare0393.cern.ch//flatfiles/SE00/atlas/CastorTest
>destination = gppse05.gridpp.rl.ac.uk//flatfiles/05/atlas/CastorTest
>Destination Location Path /flatfiles/05/atlas
>gdmp_stage_from_mss -S lxshare0393.cern.ch -P 2000 -l CastorTest -V atlas > /dev/null
>Calling GDMP staging script ...
>The program was successfully executed.
```

Files can be deleted from disk and the RC using GDMP or the replica manager commands in the usual way. However, the copy in Castor currently cannot be deleted by a normal user.

8.8.9 Tips and traps

This is a summary of some problems found in the use of GDMP v3 and the replica manager commands in release 1.4, together with some workarounds. The problems may be fixed after the release.

The data management commands need configuration files pointed to by variables RC_CONFIG_FILE and GDMP_CONFIG_FILE. In theory these should be defined correctly on a CE/WN, but in practice there can be problems. Check that the variables are defined, that the files themselves exist and that the content looks sensible. On a UI users will usually define the variables in their profile. See Section 8.8.1 for details. We are currently using plain-text passwords to write to the RCs; these are stored in the configuration files and must be correct for write commands to succeed.

For each SE the Replica Catalogue needs an entry which contains the file path on the SE. This should be created automatically the first time a file is registered on an SE, but there may be problems with this if the first file registered is in a subdirectory. There will also be problems if the file path on an SE is changed after the information is registered.

There is no consistency check between an RC and the files on disk, so it may be that a file which is recorded in the RC does not actually exist. In general files managed via the replica catalogue should not be deleted directly with commands like **rm** and **edg-gridftp-rm** because this will not update the RC (except when the files are staged to MSS when you want them to remain in the RC).

You need to understand that GDMP 3 is a client-server system. You issue commands on a UI or CE which communicate with a GDMP server on an SE (*-S* option) which in turn may talk to other SEs. The communication between GDMP servers is authorised using the host certificate of the SE machine, rather than your user certificate. GDMP usually needs the full hostname for *-S* even if the machine is local.

In general you can get gdmp commands to create a log file with *-L* and use **gdmp_job_status** to read it. Some commands make a log file automatically. You may have to read the log file carefully to see what happened.

A basic test is to do **gdmp_ping** between pairs of SEs in both directions, e.g.:

```
gdmp_ping -S lxshare0222.cern.ch -r tbn07.nikhef.nl
gdmp_ping -r lxshare0222.cern.ch -S tbn07.nikhef.nl
```

If this fails it's often because the SE host certificate name of one SE is missing from the grid map file of the other. In this case you need to tell the relevant system manager to update the map file.

Second, the pairs of SEs you want to replicate between have to be subscribed to each other (both directions) with **gdmp_host_subscribe**. That should only need doing once per VO per pair of SEs, but is worth checking (the *-c* option checks the status).

Files need to be stored in the right place, which varies from one SE to another. The relevant path is now in the MDS (part of the SEvo field in the SE object) but there is no particularly easy way to get hold of it inside a job. You can also extract the path from the gdmp config file, which you can get with **gdmp_job_status -c gdmp.conf**, as described in Section 8.8.3 above. Alternatively the paths can be hard-coded if a fixed SE (e.g. at CERN) is used.

gdmp_register_local_file can either register all files in a directory (*-d* option) or a single file (*-p*). If you use *-d* bear in mind that you may be registering files written by another user. You can check registration by using **gdmp_job_status -c local_file_catalogue**, but it normally works. Note that this command is asynchronous, it starts a process on the SE and then returns, so if you try to publish immediately afterwards it may not yet have your file registered.

gdmp_publish_catalogue can fail for various reasons: a subscribed SE may be down, it may not have the right host certificate in the map file (see above), or the Replica Catalogue details may be wrong.

The command can also hang indefinitely if the file path in the Replica Catalogue is wrong. Currently you must use the *-C* option as we are still using passwords rather than certificates for RC authorisation. Note that if publication to the RC fails, e.g. because you forget the *-C*, gdmp still thinks it has published the file and won't publish it again! If publication to one of the subscribed hosts failed you can use **gdmp_get_catalogue** to update the catalogue there. **gdmp_publish_catalogue** publishes all registered files and so may include files written by other people.

gdmp_replicate_get is another asynchronous command. **gdmp_replicate_put** has another layer of indirection in that it starts a **gdmp_replicate_get** command on the remote SE. It seems that in replicating a large number of files you can sometimes get failures: if one file fails to replicate it kills the whole replication process. However, if you replicate again it starts from the next file. In general, once replication starts the file is marked as "being replicated" so the system will not try to replicate it again. It may be better to use the *-i* option to replicate individual files, but note that the fileid syntax is a little strange. Alternatively, put all your files in a particular directory and use the *-f* option to filter just those files. As with publishing you need the *-C* option at the moment.

Once files have been registered with gdmp they should be deleted using **gdmp_remove_local_file** to clean the catalogues correctly. Replicated files are owned by the gdmp user so this is not a problem. However, the original files are owned by the user pool account and the VO group. The system should be set up so that the default access permission has group-write set, and the gdmp user should be a member of all the VO groups, in which case it should be possible for gdmp to delete the file, but there are possible problems in this area if the permissions or group membership are wrong. Note that any subdirectories in the path also need to be writeable by gdmp.

If GDMP replicates a file in a subdirectory which does not exist on the target SE it creates the directory, but the directory will then be owned by the gdmp user and is not currently created with group write permission, so users cannot then write their own files into the directory (although gdmp can replicate them from elsewhere).

There seems to be a minor bug in that if you replicate a file to a site, delete it with **remove_local_file** and then try to replicate it again the replication fails because gdmp thinks the file has already been replicated.

For the replica manager commands, the most important thing to remember is that the LFN (*-l*) must be the same as the final part of the source and destination file names (*-s* and *-d*) whenever they refer to a file on an SE.

The replica manager commands use **globus-url-copy** to copy files, and this does not create subdirectories in the path of a copied file. Files created in this way should have the permission set to group-writeable (g+w), if this does not happen control of the file may be lost.

The replica manager commands have to be called for each file individually, so replication of batches of files, particularly where the file names may not be known a priori, is more suited to GDMP.

The output from the replica manager commands can sometimes be very verbose and is not always clear. Some of the output seems to go to stderr rather than stdout. Many commands return a success status code even if they fail.

The support for MSS staging is a stopgap solution and the exact configuration is in a state of flux. It is not particularly easy to know where files are staged to (you have to read the staging scripts). There may be bugs in the system in general or in the configuration at a particular site, although in general it does work.

9 Metadata Management

Some DataGrid services must maintain persistent metadata in remote relational databases. However, existing relational database systems are not grid-enabled, affecting adversely cross-organizational interoperability and reuse. SpitFire[1] addresses these issues. Spitfire is designed to be used for metadata storage and retrieval.

The Spitfire middleware is inserted into the control and data path between client and RDBMS and so grid-enables the RDBMS. It introduces a uniform interface, data model, network protocol and security model. These are based on widely accepted standards and neutral with respect to programming language, platform and database product.

There are three main components to the Spitfire service: the primary server component, the client library component(s), and the browser component. Applications that have been linked to the Spitfire client library communicate to a remote instance of the server. This server is put in front of a RDBMS (e.g. MySQL or Oracle), and securely mediates all Grid access to that database. The browser is a standalone web portal that can also placed in front of a RDBMS.

The server is a fully compliant Web Service implemented in Java. It runs on Apache Axis inside a Java servlet engine (currently Apache Tomcat). The service securely mediates access to the RDBMS. The service is reasonably non-intrusive, and can be installed in front of a pre-existing RDBMS. The local database administrator retains full control of the database back-end, with only limited administration rights being exposed to properly authorized grid users.

The web services client library, at its most basic, consists of a WSDL service description that describes fully the API. This API allows SQL operations to be performed from the client application upon the remote Spitfire service, with full Grid security. Using this standard WSDL description, client stubs can be generated automatically in the programming language of choice. We provide pre-built client libraries for the Java, C and C++ programming languages.

The browser is an independent component that provides the functionality of the previous version of the Spitfire service. This service does not depend on the other components. The client is any SSL capable web browser. The Spitfire browser service is implemented as a Java servlet, and provides secure access to a standardised view of the data, based on the use of server side templates.

More information can be found on the SpitFire website[2].

[1] http://cern.ch/hep-proj-spitfire/server/doc/index.html
[2] http://cern.ch/hep-proj-spitfire

10 Application Monitoring with GRM/PROVE

GRM and PROVE are application monitoring and visualisation tools of the P-GRADE graphical parallel programming environment. These tools will be modified for application monitoring in the DataGrid. The instrumentation library of GRM is generalised for a flexible trace event specification. The components of GRM will be connected to the R-GMA using its Producer and Consumer APIs. For more information on GRM see GRM - Grid Application Monitor Users Manual[1]. For more information on PROVE, see PROVE-Visualisation tool for Grid Applications[2].

[1] http://hepunx.rl.ac.uk/edg/wp3/documentation/
[2] http://hepunx.rl.ac.uk/edg/wp3/documentation/

11 Support

11.1 Website

The main Testbed website[1] contains documentation, contact information, the bug-reporting system, links to the source and packages repositories as well as links to other sites. This serves as the single point-of-access to information about the testbed activities.

11.2 Bugzilla

The bug-tracking system, Bugzilla[2], is intended to facilitate the reporting and fixing of bugs in the European DataGrid software. This includes the DataGrid's distribution of the Globus2 system; confirmed bugs in Globus will be forwarded to the Globus team.

This system is not intended to track bugs in application software, that is, user and experimental software running on the grid. For these types of problems, please refer to the list of application support contacts found on the contacts page[3].

The Bugzilla database is publicly available and can be searched by anyone. However reporting bugs requires a valid Bugzilla account. Creating an account requires only a valid e-mail address. You will be prompted to open an account when you report your first bug. Note that Bugzilla uses cookies to keep track of your account data, so your browser must have cookie support enabled.

Concise bug reports speed the resolution of the problem. Stripped-down test cases and detailed explanations are greatly appreciated. Please do search the existing bugs to see if your problem has already been reported.

11.3 Contacts

The user's first point of contact for operational problems is the local site administrator. A list of email addresses for the site administrators can be obtained with the following command:

```
ldapsearch -LLL -H ldap://lxshare0225.cern.ch:2170 -x \
           -b 'mds-vo-name=local,o=grid' \
           '(objectclass=siteinfo)' SysAdminContact
```

For application-specific problems the appropriate application representative should be contacted. Users are welcome and encouraged to use the bug-reporting facility. The user support group[4] can be contacted for help. As a last resort, users may contact the Integration Team[5].

[1] http://marianne.in2p3.fr/
[2] http://marianne.in2p3.fr/datagrid/bugzilla/
[3] http://marianne.in2p3.fr/datagrid/contacts/
[4] http://marianne.in2p3.fr/datagrid/support/
[5] mailto:hep-proj-grid-integration-team@cern.ch

A EU DataGrid Software License

EU DataGrid Software License (v2.0, 09/09/2002)

This software includes voluntary contributions made to the EU DataGrid. For more information on the EU DataGrid, please see http://www.eu-datagrid.org/.

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.

2. The user documentation, if any, included with a redistribution, must include the following notice: "This product includes software developed by the EU DataGrid (http://www.eu-datagrid.org/)."

   Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.

3. The names "EDG", "EDG Toolkit", "EU DataGrid" and "EU DataGrid Project" may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by hep-project-grid-edg-license@cern.ch.

4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features,functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in the EU DataGrid a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

DISCLAIMER THIS SOFTWARE IS PROVIDED BY THE EU DATAGRID AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. THE EU DATAGRID AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

LIMITATION OF LIABILITY THE EU DATAGRID AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B Changing Certificate Formats

B.1 P12 Format to PEM Format

Many of the certificate authorities deliver certificates through a web browser. To use these certificates with Globus, they must be exported from the browser and then reformatted for Globus. Exporting is browser-specific so you will need to follow the help provided with your browser. Once you have extracted the certificate you should have a file with a p12 extension. This file is in the PKCS12 format; you will need to change this to PEM format. If the edg-utils package is installed on your machine, simply executing **/opt/edg/bin/pkcs12-extract** will create appropriate certificate and key files and place them in the standard location. This is a convenience method for the following:

```
openssl pkcs12 -nocerts \
        -in cert.p12 \
        -out ~user/.globus/userkey.pem

openssl pkcs12 -clcerts -nokeys
        -in cert.p12
        -out ~user/.globus/usercert.pem
```

The first command gives you your private key; this file must be readable only by you (e.g. unix permission 0600). The second command gives your public certificate (e.g. unix permission 0644). The `~ user` should be replaced by the path to your home area. The `.globus` subdirectory is standard place to put your certificates.

B.2 PEM Format to P12 Format

Popular browsers typically use certificates in PKCS12 format. Consequently you will need to modify the format of the PEM certificates used for Globus to use them within a browser. To change a certificate from PEM format into PKCS12 format (on a machine with edg-utils installed), just issue the command **/opt/edg/bin/grid-mk-pkcs12**. Again, this is a convenience method for the following:

```
openssl pkcs12 -export \
               -out file_name.p12 \
               -name "My certificate" \
               -inkey ~user/.globus/userkey.pem \
               -in ~user/.globus/usercert.pem
```

where `file_name.p12` is the name of the PKCS12 certificate, and the `~ user` in the last two lines should be replaced by the path to your home area. You must then import the certificate into your browser. (See Section 2.2.2.)

C Replica Catalogue Server

WP2 provides an RPM for the RC server with GSI enabled access (edg-rc-server-2.0). This package only needs to be installed on the server side, i.e. once per VO. There is no direct access for users. The VO-specific parameters for the RC servers are shown in Tables C.1–C.7.

Table C.1: ALICE RC Parameter

GDMP_STORAGE_DIR	/alice
GDMP_REP_CAT_HOST	ldap://grid-vo.nikhef.nl:10489
GDMP_REP_CAT_NAME	AliceReplicaCatalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxx
GDMP_REP_CAT_CN	dc=eu-datagrid,dc=org
GDMP_REP_CAT_FILE_COLL_NAME	Alice WP1 Repcat

Table C.2: ATLAS RC Parameter

GDMP_STORAGE_DIR	/atlas
GDMP_REP_CAT_HOST	ldap://grid011g.cnaf.infn.it:9011
GDMP_REP_CAT_NAME	ATLAS Testbed1 Replica Catalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxx
GDMP_REP_CAT_CN	dc=dell04,dc=cnaf,dc=infn,dc=it
GDMP_REP_CAT_FILE_COLL_NAME	Atlas Lc1

Table C.3: CMS RC Parameter

GDMP_STORAGE_DIR	/cms
GDMP_REP_CAT_HOST	ldap://grid011g.cnaf.infn.it:9211
GDMP_REP_CAT_NAME	CMS Testbed1 Replica Catalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxx
GDMP_REP_CAT_CN	dc=grid011g,dc=cnaf,dc=infn,dc=it
GDMP_REP_CAT_FILE_COLL_NAME	test0

Table C.4: LHCb RC Parameter

GDMP_STORAGE_DIR	/lhcb
GDMP_REP_CAT_HOST	ldap://grid-vo.nikhef.nl:10889
GDMP_REP_CAT_NAME	LHCbReplicaCatalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxx
GDMP_REP_CAT_CN	dc=eu-datagrid,dc=org
GDMP_REP_CAT_FILE_COLL_NAME	LHCb WP1 Repcat

Table C.5: DZero RC Parameter

GDMP_STORAGE_DIR	/dzero
GDMP_REP_CAT_HOST	ldap://grid-vo.nikhef.nl:10589
GDMP_REP_CAT_NAME	DZeroReplicaCatalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxx
GDMP_REP_CAT_CN	dc=eu-datagrid,dc=org
GDMP_REP_CAT_FILE_COLL_NAME	DZero EDG Repcat

Table C.6: Earth Observation RC Parameter

GDMP_STORAGE_DIR	/eo
GDMP_REP_CAT_HOST	ldap://grid-vo.nikhef.nl:10689
GDMP_REP_CAT_NAME	EarthObReplicaCatalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxxx
GDMP_REP_CAT_CN	dc=eu-datagrid,dc=org
GDMP_REP_CAT_FILE_COLL_NAME	EarthOb WP1 Repcat

Table C.7: Biomedical RC Parameter

GDMP_STORAGE_DIR	/biome
GDMP_REP_CAT_HOST	ldap://grid-vo.nikhef.nl:10389
GDMP_REP_CAT_NAME	BioMedicalReplicaCatalog
GDMP_REP_CAT_MANAGER_DN	Manager
GDMP_REP_CAT_MANAGER_PWD	xxxxxx
GDMP_REP_CAT_CN	dc=eu-datagrid,dc=org
GDMP_REP_CAT_FILE_COLL_NAME	BioMedical WP1 Repcat

D Relational Grid Monitoring Architecture (R-GMA)

R-GMA is based on the Grid Monitoring Architecture from the Global Grid Forum (GGF). In R-GMA the implementation is based on a relational model. R-GMA makes information from Producers available to Consumers as relations (tables). The R-GMA implementation uses HTTP Servlet technology. Communication with the servlets is achieved via an API. This API has been implemented in Java and C++ but C and other languages will be provided soon. The response from a servlet is in the form of an XML document that corresponds to an XML schema definition. Related R-GMA documentation[1] can be found on the web.

1. Information and Monitoring Services Architecture: This presents the architecture, use cases, and requirements along with the design and evaluation criteria.
2. R-GMA Users Guide: This explains what you need to know as a user of the Relational Grid Monitoring Architecture (R-GMA) Information and Monitoring Services.
3. R-GMA Installation Guide: This explains what you need to know as an installer of the various parts of the R-GMA system. It covers all the components, though most sites will only need some parts configuring.
4. R-GMA Developers Guide: This explains how to get started as an R-GMA developer. It trys to cover everything from setting up a computer to do the development to generating a R-GMA release.

[1] http://hepunx.rl.ac.uk/edg/wp3/documentation/

E Information Schema

This appendix describes the elements of the information schema for Computing Elements and Storage Elements (the information system may also contain other objects, e.g. for network monitoring and the standard Globus information). At present only CE information can be used in the JDL Rank and Requirements expressions, although other tools can look at the SE information.

Information may be static (either read from a text file written by the local system manager or taken from e.g. the batch system configuration) or dynamic (calculated from the state of the running system by information provider scripts). Dynamic items do not necessarily change rapidly, and conversely static items may change if the configuration is changed. Some attributes can appear multiple times, but most only occur once per object. The order in which objects, and items within objects, appear is not significant.

The values below are taken from the RAL site as an example. Bear in mind that the quality of the information is only as good as the configuration (for static items) or the information provider scripts (dynamic items).

E.1 ObjectClass=StorageElement

This is the main object for SE information.

SEId=gppse05.gridpp.rl.ac.uk (static)

The SEId is a unique identifier for the SE: the hostname.

CloseCE=gppce05.gridpp.rl.ac.uk:2119/jobmanager-pbs-S (static, multi)

CloseCE is a list of CEIds (see below) which identify CEs considered to be close to the SE (usually on the same site).

SEtypearchitecture=disk (static)

Always "disk" in the current system.

SEsize=332 (static)

The total disk space available for all VOs, in Gb.

SEResourceContactString=edg-site-admin@gridpp.rl.ac.uk (static)

An e-mail address to contact the SE administrator.

SEvo=alice:/flatfiles/05/alice (static, multi)

A list of names of the VOs supported on the SE, and the pathname of the VO storage area.

SESubjectName=/O=Grid/O=UKHEP/CN=host/gppse05.gridpp.rl.ac.uk (static)

The Subject Name of the SE host certificate, used for mutual authentication between GDMP servers.

E.2 ObjectClass=StorageElementProtocol

There is one StorageElementProtocol object for each access protocol supported by an SE.

SEId=gppse05.gridpp.rl.ac.uk (static)

The SE identifier.

SEProtocol=gridftp (static)

The name of the protocol.

Port=2811 (static)

The port on which the server supporting the protocol listens (if any).

E.3 ObjectClass=StorageElementStatus

There is one StorageElementStatus object per SE with dynamic status information (rather limited in the current system).

SEId=gppse05.gridpp.rl.ac.uk (static)

The SE identifier.

SEfreespace=317231 (dynamic)

The total free disk space on the SE, in megabytes. This may not be accurate if the disk configuration on the SE is complex.

E.4 ObjectClass=SiteInfo

A SiteInfo object contains general information about a site. These are usually attached to the CE information, but may also be published by an SE.

siteName=RAL The name of the site, as chosen by the local administrator. (static)

userSupportContact=edg-site-admin@gridpp.rl.ac.uk

siteSecurityContact=edg-site-admin@gridpp.rl.ac.uk

sysAdminContact=edg-site-admin@gridpp.rl.ac.uk (static)

Three email addresses for user support, security issues and other matters. These may be the same address, as here, or may be different. In some cases they may be the personal email address of the system manager, but they are often generic addresses which will route mail to the correct person. In general these addresses should be used in preference to any others.

dataGridVersion=v1_4_0 (static)

The version number of the last EDG release installed (here 1.4.0), although patches may have been added subsequently.

installationDate=20021111103000Z (static)

The date and time of the last installation, in GMT (here 10:30 on 11/11/02). Again some changes may have been made since this date.

E.5 ObjectClass=ComputingElement

This object carries most of the information about a system. There is one ComputingElement object for each batch queue reachable through a given gatekeeper machine.

CEId=gppce05.gridpp.rl.ac.uk:2119/jobmanager-pbs-S (static)

A unique identifier for the CE, consisting of the gatekeeper host name, the port on which the gatekeeper listens, the type of batch system (here PBS) and the queue name (here S).

GlobusResourceContactString=gppce05.gridpp.rl.ac.uk:2119/jobm... (static)

Information needed to contact the gatekeeper: hostname, port, type of batch system, and the Subject Name of the host certificate.

GRAMVersion=? (static)

Not used.

Architecture=intel (static)

Always "intel" in the current system.

OpSys=RH 6.2 (static)

Operating system: currently either RH 6.2 or RH 7.2.

MinPhysicalMemory=512 (static)

This should be the minimum amount of physical memory on any node accessible through this CE. However, in practice it is often not configured correctly.

MinLocalDiskSpace=2048 (static)

This should be the minimum disk space available to a job running on a batch worker, but in practice in the current system it means very little.

TotalCPUs=8 (static)

Depending on the local configuration this may be either the total number of physical CPUs accessible through this CE, or the number of slots for running jobs (these are often the same thing). Note that most sites have several queues, and hence several CEs, which can feed jobs to the same set of CPUs, so adding up CPU counts for multiple CEs may well lead to overcounting.

FreeCPUs=8 (dynamic)

The number of CPUs/slots which are free to take new jobs.

NumSMPs=4 (static)

The number of physical machines with at least 2 processors.

MinSPUProcessors=2

MaxSPUProcessors=2 (static)

These give the minimum and maximum number of processors for the SMP machines in the CE. In this case it can be seen that there are 4 dual-processor machines giving 8 CPUs in total.

TotalJobs=0 (dynamic)

The total number of jobs in this queue. Note that other queues may feed the same CPUs, so the job count will not in general match the number of non-free CPUs.

RunningJobs=0 (dynamic)

The number of executing jobs in this queue.

IdleJobs=0 (dynamic)

The number of jobs in the queue waiting for execution. It should be the case that IdleJobs + RunningJobs = TotalJobs.

MaxTotalJobs=99999 (static)

The maximum number of jobs allowed in the queue at one time, or 99999 if there is no limit.

MaxRunningJobs=99999 (static)

The maximum number of running jobs allowed in the queue at one time, or 99999 if there is no limit. Note that in any case it will not be possible to have more running jobs than slots available.

WorstTraversalTime=0

EstimatedTraversalTime=0 (dynamic)

The traversal time is defined as the real (wallclock) time in seconds between a job being submitted to a CE and the start of execution. These values should be the worst possible traversal time and an estimate of the likely time. EstimatedTraversalTime is the default for the JDL Rank expression. However, in practice it is very difficult to make accurate calculations of traversal times and the values are often substantially misleading, so they should be used with care. If the algorithm is unable to calculate anything the values will default to 999999 (WorstTraversalTime) and 99999 (EstimatedTraversalTime).

Active=TRUE (static)

This means that the queue is accepting jobs. If JDL files have a requirement on the Active value (which is the default) then a system manager can disable the queue in advance of a shutdown to drain the queue.

Priority=30 (static)

This should allow a Grid-wide definition of queue priorities, but so far there is no clear definition for what priority values should mean.

MaxCPUTime=28800 (static)

The maximum CPU time, in seconds, allowed for this queue, or 99999 if there is no limit. All but the shortest jobs should have a Requirement expression which includes a minimum acceptable value for MaxCPUTime.

MaxWallClockTime=115200 (static)

The maximum real time allowed for jobs in this queue, or 99999 if there is no limit. This will usually be substantially longer than the MaxCPUTime, but allows jobs which are e.g. stuck in a blocking I/O call to be killed.

AverageSI00=480

MinSI00=480

MaxSI00=480 (static)

This gives the minimum, maximum and average SpecInt 2000 speed ratings for the CPUs accessible through the CE. These values are likely to be general estimates, so limited reliance should be placed on them.

AuthorizedUser=/O=Grid/O=UKHEP/OU=pp.rl.ac.uk/CN=Stephen Burke (static, multi)

The certificate Subject Names of all users authorised to run jobs on the CE are listed.

RunTimeEnvironment=ATLAS-3.2.1

RunTimeEnvironment=RAL-PRO (static, multi)

RunTimeEnvironment gives a way of providing arbitrary tags to identify capabilities and features of the CE. The tags are text strings which are not interpreted in any way by the system, but they can be matched in the Requirements expression in a JDL file. They are typically used to identify installed software packages. It is also common to have one or more tags which identify the site (RAL-PRO in this case) and sometimes a site grouping (e.g. INFN).

AFSAvailable=TRUE (static)

Indicates whether the worker nodes have AFS available.

OutboundIP=TRUE (static)

Defines whether a running job can initiate an ip connection to the external network. In the current system software limitations mean that this is always TRUE, but in future many systems are likely to have batch workers on a private network with no general access to the internet.

InboundIP=FALSE (static)

This means that connections to a worker node can be initiated from outside the system.

QueueName=S (static)

The name of the queue in the local batch system.

LRMSType=pbs (static)

The type of batch system. Currently supported batch systems are PBS, LSF, BQS and Condor. The interpretation and calculation methods for other information items may vary depending on the batch system.

LRMSVersion=OpenPBS_2.3 (static)

The batch system version number.

E.6 ObjectClass=CloseStorageElement

Each CE can have zero or more CloseStorageElement objects, each of which define one SE as being close to the CE.

CEId=gppce05.gridpp.rl.ac.uk:2119/jobmanager-pbs-S (static)

The unique identifier of the CE.

CloseSE=gppse05.gridpp.rl.ac.uk (static)

The unique identifier of the close SE.

MountPoint=/flatfiles/05 (static)

The NFS/AFS mount point of the SE storage area on this CE. This is common to all VOs; the VO-specific storage area has the VO name appended to the mount point.

F LDAP

LDAP (Lightweight Directory Access Protocol) is used for several services in the EDG testbed, in particular the main information system (MDS) but also the Replica Catalogues and the VO servers. This appendix gives a brief description of LDAP and a few tools to browse an LDAP database. More information can be found in the numerous LDAP book and documents. [1] [2]

LDAP is a standard internet protocol which gives an interface to a simple, non-relational database. Data are organised into objects which have one or more types (objectclasses), which define the schema for the data in the object. The objectclasses are defined in an inheritance hierarchy as in an object-oriented programming language, but in practice it's usually only the most specific objectclass which is interesting.

The schema defines the names of the attributes of an object and their types (number, date, case-exact string, case-insensitive string etc.), although for many purposes the type is not important to the user. Attributes can be single-valued (occurring only once per object) or multi-valued (occurring any number of times). Attributes may also be optional. The order of attributes within an object is not significant, so it is not possible to have tables within an object. This can sometimes lead to somewhat clumsy constructions, for example the CloseStorageElement and StorageElementProtocol objects in the EDG information system exist only to bind pairs of attributes together.

The objects in a given LDAP database are identified by a unique Distinguished Name (DN) which consists of an ordered list of comma-separated elements of the form $x = y$. For example, in the main EDG information index this DN identifies the CE information for the L queue on the gppce05.gridpp.rl.ac.uk host at RAL:

```
ceId=gppce05.gridpp.rl.ac.uk:2119/jobmanager-pbs-L,
  hn=gppce05.gridpp.rl.ac.uk,
    Mds-Vo-name=ral,
      Mds-Vo-name=edgpro,
        Mds-Vo-name=local,
          o=grid
```

The elements of the DN form a hierarchy known as the Directory Information Tree (DIT), with the root of the tree at the rightmost end. (This is somewhat similar to the structure of the registry in Windows.)

For many purposes users can ignore the detailed structure of the DIT, since most objects contain enough information to identify them anyway. However, to query a database you must know the DN which forms the root on a given server, known as the base DN. For the information system it's mostly sufficient to use a base DN of "mds-vo-name=local,o=grid". For other systems the base DN varies and you should consult the documentation.

F.1 LDAP Queries

The EDG tools perform queries as necessary on behalf of the user. However, it's often useful to be able to query the directories directly. There are various tools for this, with their own strengths and weaknesses. The rest of this appendix lists some of the most useful.

[1] http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg244986.pdf
[2] http://www.openldap.org/

F.1.1 ldapsearch

OpenLDAP is a standard part of most Linux distributions, and comes with a command called **ldapsearch** to query an LDAP database. Unfortunately there are two incompatible versions in common use, so if command examples don't work it may be that you have a different version.

The simplest form of the command is:

```
ldapsearch -x -h <hostname> -p <port> -b <baseDN> objectclass=<objectclass>
```

The *-x* relates to a security option, try omitting it if you get an error. It may be necessary to quote some of the arguments if they contain special characters. The basic information needed to contact an LDAP server is the hostname, port and base DN, so if you have problems check that these are right (there may be multiple directories with different base DNs on the same host and port). Note that DNs can contain spaces, which are significant.

The objectclass argument specifies what type of object you want to look at, and can be "*" to return all objects. By default you will get all attributes of the selected objects, but the command can be followed by a space-separated list of attribute names to restrict the query. The command prints a formatted list of information to standard output, which can be very voluminous, so filtering with **grep**, **awk** or **perl** is often useful.

There are many other options to the **ldapsearch** command, see its man page for details.

F.1.2 grid-info-search

The Globus distribution contains a command **grid-info-search** with similar functionality to **ldapsearch**, but this is less useful as it is only available on systems where Globus is installed, and the documentation is limited.

F.2 LDAP Browsing

F.2.1 Netscape

Netscape, at least in some versions, has built-in support for displaying LDAP URLs. (Internet Explorer tries to open them as address books, which is not especially useful). The display is fairly basic, but is sufficent for simple uses. The advantages are that this works on any platform which has Netscape installed, and that the URLs can be stored as bookmarks or embedded in web pages.

An LDAP URL has the form:

```
ldap://lxshare0373.cern.ch:2135/mds-vo-name=local,o=grid?runningjobs,freecpus?sub?objectclass=computingelement
```

The first part is the standard contact information of host, port and base DN. This is followed by three arguments separated by question marks. The first is a comma-separated list of attributes to display; if this is empty you will get all attributes. The "sub" in the second argument gives you everything in the subtree below the base DN. If you omit it you get only the top level information, which is usually not useful. The final argument specifies which objectclass you want, and can be omitted to get all objects.

F.2.2 Graphical Browsers

Since LDAP is a standard protocol there are freeware browsers on the market. A good example is one written by Jarek Gawor[3].

[3] http://www.iit.edu/ gawojar/ldap

F.2.3 Web-based Tools

There are various monitoring web pages which pull information from the information system and display it in a more or less useful form. See for example, MapCenter[4] and a PHP-based monitor[5] from NorduGrid.

There are also various projects to provide user-interface portals which often include the ability to browse the information system and Replica Catalogues, see for example the Genius portal[6].

[4] http://ccwp7.in2p3.fr/mapcenter/
[5] http://www.nordugrid.org/monitor/edgstat/
[6] https://genius.ct.infn.it/