



## Design and architecture for QCDgrid implementation of ILDG interface to File Catalogue and Storage Element

**Project Title:** QCDgrid

**Document Title:** Design and architecture for QCDgrid implementation of ILDG interface to File Catalogue and Storage Element

**Document Identifier:** QCDGRID2-D4-4-DES

**Document Filename:** QCDGrid2-DAofFCforILDG.doc

**Distribution Classification:** Public

**Authorship:** Radoslaw Ostrowski

**Approval List:** QCDgrid Project Management Board

**Distribution List:** Public

### Document History:

<i>Personnel</i>	<i>Date</i>	<i>Summary</i>	<i>Version</i>
RHO	29/MAY/07	First release	1.0

## Contents

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
1.1	<b>The QCDgrid Project .....</b>	<b>3</b>
<b>2</b>	<b>Background to the ILDG file catalogue.....</b>	<b>4</b>
2.1	<b>Glossary .....</b>	<b>5</b>
<b>3</b>	<b>Design for authentication/authorisation setup.....</b>	<b>6</b>
3.1	<b>What should be the appropriate permissions of files of QCDgrid installation and storage folders? .....</b>	<b>6</b>
3.2	<b>How to create pooled accounts so they work with new setup? .....</b>	<b>6</b>
3.3	<b>How to modify the grid-mapfile generation script to add/classify ILDG members?.....</b>	<b>7</b>
3.4	<b>How to modify a configuration file of the RLS? .....</b>	<b>7</b>
3.5	<b>Enumerate updates to QCDgrid test script to check setup of pooled accounts and permissions on data files. ....</b>	<b>8</b>
<b>4</b>	<b>DiGS Version 2.0.....</b>	<b>9</b>
4.1	<b>Changes required to the DiGS software to support new RLS information, plus correctly handle addition, replication and deletion of files. ....</b>	<b>9</b>
4.1.1	<b>Files permissions .....</b>	<b>9</b>
4.1.2	<b>Addition .....</b>	<b>10</b>
4.1.3	<b>Replication.....</b>	<b>11</b>
4.1.4	<b>Deletion .....</b>	<b>11</b>
4.2	<b>Establish modifications to existing command-line tools for DiGS 2.0.....</b>	<b>11</b>
4.2.1	<b>Digs-list .....</b>	<b>11</b>
4.2.2	<b>Digs-catalogue-rebuild .....</b>	<b>12</b>
4.3	<b>Design new command-line tools.....</b>	<b>13</b>
<b>5</b>	<b>Design for ILDG File Catalogue and File Transfer Service .....</b>	<b>14</b>
5.1	<b>Design for F/C Web Service .....</b>	<b>14</b>
5.1.1	<b>gLite Delegation Service operations.....</b>	<b>14</b>
5.1.2	<b>File Catalogue specific operation – getURL .....</b>	<b>15</b>
5.1.3	<b>Server-side implementation of getURL .....</b>	<b>17</b>
5.2	<b>Direct file access .....</b>	<b>18</b>
5.3	<b>File transfer service .....</b>	<b>18</b>
<b>6</b>	<b>Conclusions.....</b>	<b>20</b>
<b>7</b>	<b>References.....</b>	<b>21</b>

# 1 Introduction

## 1.1 The QCDgrid Project

The QCDgrid project [6] is a core activity of UKQCD [5], a collaboration of UK academics and researchers that aims to procure and jointly exploit computing facilities for lattice field theory (commonly referred to as Lattice QCD) calculations. The primary aim is to increase the predictive power of the *Standard Model of elementary particle interactions* through numerical simulation of *Quantum Chromodynamics*. Such numerical simulations produce significant amounts of data and the purpose of the QCDgrid project is to provide software and supporting infrastructure that simplifies the management, storage, and manipulation of this data.

In the first three years of the project (2002—2004), software engineers at EPCC developed QCDgrid—a data management system that combines the distributed resources of the collaborators into a robust facility called the *UKQCD Grid*. The result is a multi-terabyte storage facility over seven sites at: University of Columbia, University of Edinburgh (including the UoE Advanced Computing Facility), University of Liverpool, Rutherford Appleton Laboratories (RAL), University of Southampton, and University of Wales Swansea. The University of Glasgow is also a member of the consortium.

The facility is based on commodity hardware and open-source software. The hardware consists primarily of high specification, PC-based servers running the Linux operating system and managing large RAID storage arrays. On top of this infrastructure, the QCDgrid software (built using components from the Globus Toolkit [2], EGEE application stack [1], and an XML database) provides *Datagrid* management and user functionality – furnishing a simple and intuitive environment that hides the complexities of the underlying grid and presents a standard file system to the user. It incorporates a robustness metric that automatically disperses datasets across the grid, providing a resilience that ensures data is not affected by the loss of one (or possibly more) storage nodes. Security is leveraged from the Globus Toolkit, based on X.509 digital certificates issued by an approved Certificate Authority. The result is a reliable and secure data management system.

UKQCD is an important contributor to the International Lattice Data Grid (ILDG) [3], a group of like-minded scientists, working around the world, who aim to share their data to accelerate scientific progress in the field of Lattice QCD. The ILDG was initiated in 2002 by UKQCD and, at the time of writing, has significant representation from research groups in Australia, France, Germany, Italy, Japan, UK and USA.

The ILDG infrastructure is being assembled as a web services layer that will aggregate the resources of each contributing collaboration (that is, regional grid such as the UKQCD Grid) for the benefit of the wider community. To achieve its objectives, ILDG has established two working groups:

- Metadata Working Group – to facilitate data sharing, through the standardisation of the format and content for Lattice QCD scientific data and associated metadata.
- Middleware Working Group – to produce a set of specifications that define an architecture for an international *Grid of Grids* for Lattice QCD.

This document describes the design for the *ILDG File Catalogue*, a key component of the ILDG infrastructure. In Section 2, we explain the purpose of the file catalogue in the context of the wider infrastructure. Then in Section 3, we consider several security aspects concerning user authentication and authorisation to perform any operations on *UKQCD Grid*. In Section 4, we discuss any changes that are necessary before a version 2 of the QCDgrid software – which will be called DiGS (Distributed Grid Storage) – can be released. Then in Section 5, we describe the design for the *ILDG FC Web Service* and *File Transfer Service*. Finally in Section 6, we draw together conclusions, based on the work to date, and establish the next steps towards a functioning implementation.

## 2 Background to the ILDG file catalogue

In a *grid* context, a file catalogue is a registry that binds unique file identifiers (commonly referred to as Logical Filenames (LFNs)) to one or more instances of (internet) locations, or URLs, where a copy of the specific file exists.

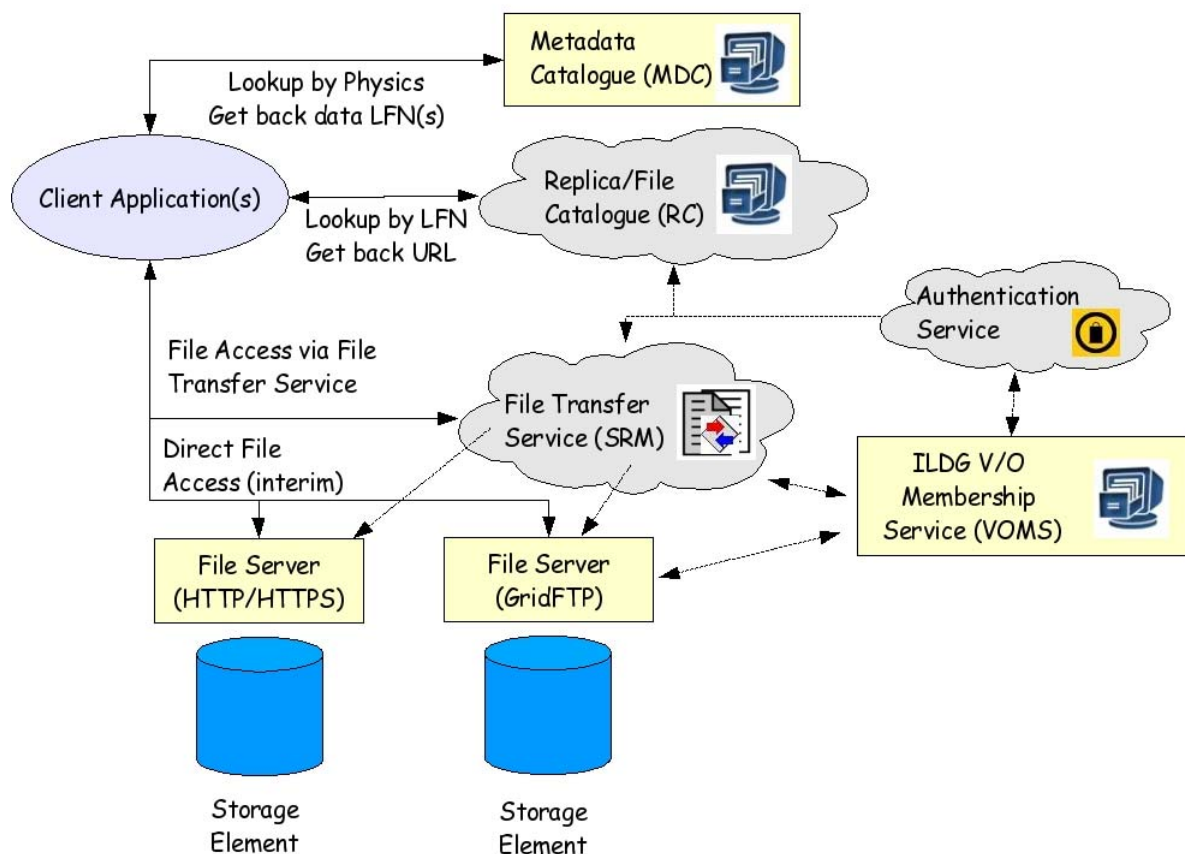


Figure 1: Modular overview of the ILDG infrastructure

A file catalogue is a core component of the ILDG infrastructure (see Figure 1) that is used to track the location of Lattice Gauge Configuration datasets within the storage resources provided by the regional grids. It is most powerful when combined with the ILDG Metadata Catalogue (see QCDgrid project deliverables from Work Package 3 [6]), which lists the lattice gauge configurations that are available, described using meaningful and standardised physics terminology and marked up using QCDML. The file catalogue contributes to the ILDG’s “Search and Retrieval” use cases [4][7].

At the time of writing, each regional grid has implemented – or is in the process of implementing – a local file catalogue to manage the datasets owned by the particular collaboration. Some groups, such as UKQCD, CSSM (Australia) and LDG (Germany), have elected to use a third-party catalogue: UKQCD and CSSM use the Globus Replica Location Service [2], while LDG use the LCG File Catalogue Service [1]. Other collaborations, such as USQCD (USA) and JLDG (Japan), have elected to build their own file catalogues from scratch.

Both of these approaches (using a third-party catalogue or writing one’s own from scratch) have advantages for the individual collaborations. However, for the ILDG as a whole, the different approaches imply that member collaborations present incompatible interfaces and provide different functionalities to the potential user. This is not a new situation for the middleware working group: it was previously encountered and successfully overcome during the specification of the ILDG Metadata Catalogue. As for the metadata catalogue, the middleware working group propose to overcome these

incompatibilities by introducing a web service layer that will present a standard interface on top of the regional catalogues.

The determination of the form and function of this web service layer has been a focus of discussions within the working group over the previous 18 months. The QCDgrid project team, as representatives for UKQCD to the middleware working group, have been heavily involved in this process through both face-to-face meetings (Japan, October 2005; Germany, July 2006; and USA, December 2006) and monthly teleconferences of the working group. During the December 2006 meeting, the working group arrived at an agreed specification for the file catalogue that is to form the basis for the design and implementation activities of the regional grids.

## 2.1 Glossary

The following terminology is used within this document:

*Logical name* – a unique identifier for the contents of a data item.

*Logical file name* – a unique identifier for the contents of a file.

*Physical file name* – the address or the location of a copy of a file on a storage system.

*Replica Location Service (RLS)* – a registry that keeps track of where replicas exist on physical storage systems. The job of the RLS is to maintain associations, or mappings, between logical names for data objects and one or more target or physical names for replicas. Users or services register data items in the RLS and query the RLS to find copies of data items.

*Authentication* – a process of identifying a user. Usually, it is done by requiring the user to provide credentials (username and password) which are then compared with those stored in a database. In the Grid Security Infrastructure (GSI), these credentials are provided by X.509 certificates.

*Authorisation* – a process of enforcing policies. Once a user has been authenticated, decisions can be made as to what types of services or resources this particular user is allowed to utilise.

*Delegation* – an act of transferring rights and privileges to another party.

*Delegatee* – a requestor of delegation, receives delegated credentials from Delegator.

*Delegator* – an entity that delegates the abilities and/or rights to the Delegatee.

*Transport Layer Security (TLS)* – a protocol that ensures privacy between communicating parties on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message sent over the network. TLS has superseded the Secure Sockets Layer (SSL).

*Proxy Certificate* – X.509 certificates issued by the end-user (the *Delegator*) to a process acting on the end-user's behalf (the *Delegatee*). The proxy certificate carries the identity of the Delegator: that is, it can be used to establish *TLS* connections, and is interpreted using the GSI as authority to perform work on the Delegator's behalf.

## 3 Design for authentication/authorisation setup

### 3.1 What should be the appropriate permissions of files of QCDgrid installation and storage folders?

As there will be different types of users allowed on the grid: *special*, *ukqcd* and *ildg* (see [11]), we have to make sure that data files which are considered *private* are only accessible by users with appropriate rights. In order to achieve this, we should protect the files using their UNIX file permissions. There are several conditions that have to be fulfilled before ILDG users are allowed on the UKQCD grid.

- There should be two user groups: *ukqcd* and *ildg*. All collaborators will belong to one or other of these groups.
- All files should be owned by *qcdgrid* user and belong to *ukqcd* group.
- Only *ukqcd* group should have permission to write to NEW folder.
- ILDG users (referred to as others) should have no write permissions to anything, by default.
- Group *ukqcd* should be able to read files from its group as well as files that are classified as *public*. In addition, group should be able to list content of the directories.
- Others should be able to list content except of NEW folder.
- Others shouldn't be able to read from data storage folders by default (only files made public should be accessible).

If any ILDG user tries to add a file to the grid, this operation will fail as they do not have write permission to the NEW folder. It should be accompanied with a clear error message.

Note, that the operation of adding files to the grid (`qcdgridPutFile()` and `qcdgridPutDirectory()` in *client.c*) should be modified so that the new files are set up with correct permissions (owned by *qcdgrid*, readable by *ukqcd* group, and not available for others).

A script called *changePermissions.pl* allows setting up the *qcdgrid* account and storage folders with appropriate UNIX file system permissions. Note, that this script can change all data files to private if required (e.g. new setup). The script can be found in NeSCForge CVS repository under *qcdgrid/tools/setupFiles*.

### 3.2 How to create pooled accounts so they work with new setup?

Allowing ILDG collaborators to use the UKQCD grid resources requires changes to the configuration of pooled accounts. The pooled accounts patch for Globus enables the usage of multiple pooled accounts. This feature should be utilised and two separate pools of accounts should be created. The steps presented below should be followed in this process:

- Create two user groups: *ukqcd* and *ildg*, unless they already exist
- Create two directories in /home: *ukqcd* and *ildg*
- Create a directory *gridmapdir* in */etc/grid-security*
- Define an environment variable with `export GRIDMAPDIR = /etc/grid-security/gridmapdir`
- Add the definition of GRIDMAPDIR environment (above) to *\$VDT\_LOCATION/setup.sh* and/or *\$GLOBUS\_LOCATION/etc/globus-user-env.sh* and, if exist, the wrapper scripts.

- Create two sets of pooled accounts (the scripts mentioned below can be found in NeSCForge CVS repository under *qcdgrid/tools/setupFiles*):
  - Run “addUKQGusers.sh” script, which will add 20 pooled accounts for UKQCD users. They should belong to ukqcd group and the accounts should be stored in /home/ukqcd
  - Run “addILDGusers.sh” script, which will add 20 pooled accounts for ILDG users. They should belong to ildg group and be stored in /home/ildg
  - Install “cleanUKQusers” and “cleanILDusers” scripts, to expire leases of both types of the pooled accounts, by copying them to /etc/cron.hourly

Once the pooled accounts are properly installed, the grid-mapfile has to be generated so it includes all kinds of users with appropriate mappings.

### 3.3 How to modify the grid-mapfile generation script to add/classify ILDG members?

The script that downloads a list of users from VOMS should be able to distinguish UKQCD and ILDG collaborators and add “.ukq” and “.ild” mappings for their DNs respectively.

```
"DN of UKQCD" .ukq
"DN of ILDG" .ild
```

This replaces the previous “.”(dot) mapping for the UKQCD users in the grid-mapfile.

New PERL script called *listAllMembersDNs.pl* provides mentioned above functionality and extends the previously used script *listMembersDNs.pl*.

### 3.4 How to modify a configuration file of the RLS?

Presented below is a fragment of the configuration file of the RLS server (*\$GLOBUS\_LOCATION/etc/globus-rls-server.conf*) that describes access control for the catalogue. The list of privileges given to the users should be separated by a whitespace. Note, that the hash symbol # means a comment. The below change should be only made on the machine hosting the RLS server.

```
# Suggested changes for the QCDGrid infrastructure:

# In order to enable a full access for everyone
# (not advised), remove the comment below:
# acl .*: all

# Account like qcdgrid and those belonging to the administrators have all privileges
acl qcdgrid: all
acl rostrow: all
acl jamesp: all

# Only read access for UKQCD users (mapped in gridmap file as '.ukq')
acl ukq0[0-9][0-9]: lrc_read rli_read stats

# Same for ILDG users (mapped in gridmap file as '.ild')
acl ild0[0-9][0-9]: lrc_read rli_read stats
```

Note, that after making any changes to the configuration file, the RLS needs to be restarted for the changes to take effect:

```
$GLOBUS_LOCATION/sbin/SXXrls stop
$GLOBUS_LOCATION/sbin/SXXrls start
or if VDT is used:
$VDT/vdt/sbin/vdt-control --off rls
```

---

```
$VDT/vdt/sbin/vdt-control --on rls
```

Or, if a restart is not desirable, by sending the following signal to the RLS server process:

```
kill -SIGHUP <RLS-server-pid>
```

### 3.5 Enumerate updates to QCDgrid test script to check setup of pooled accounts and permissions on data files.

Extending the functionality of the QCDgrid software introduces additional complexity to the initial setup. Therefore, it is required to verify if all of the components are configured correctly. This could be achieved by expanding the existing QCDgrid test script by several additional test cases. However, as we would like the software to be more generic, we should create two test suites: one general and one UKQCD specific. The tests specific only to UKQCD are prefixed with [UKQCD].

1. Pooled accounts / grid-mapfile
  - a. Check if environment variable GRIDMAPDIR is exported and points to /etc/grid-security/gridmapdir
  - b. [UKQCD] Check if there are ild000 and ukq000 type accounts in /home/ildg and /home/ukqcd respectively
  - c. [UKQCD] Check if there are “cleanUKQuser” and “cleanILDuser” scripts in /etc/cron.hourly
  - d. [UKQCD] Check grid-mapfile if it includes “.ild” and “.ukq” mappings and no “.” mappings
2. File system permissions
  - a. Files readable and executable by group: STORAGE/qcdgrid directories
  - b. Only directory listing for others (except of public files): STORAGE/qcdgrid
  - c. Files readable, writable and executable by group (not others): STORAGE/qcdgrid/NEW
3. RLS configuration
  - a. Check if there are the following entries in “\$GLOBUS\_LOCATION/etc/globus-rls-server.conf” on the node which runs the RLS server:
    - i. acl qcdgrid: all
    - ii. [UKQCD] acl ukq0[0-9][0-9]: lrc\_read rli\_read stats
    - iii. [UKQCD] acl ild0[0-9][0-9]: lrc\_read rli\_read stats

## 4 DiGS Version 2.0

### 4.1 Changes required to the DiGS software to support new RLS information, plus correctly handle addition, replication and deletion of files.

#### 4.1.1 Files permissions

To support a simultaneous existence of both private and public files on the grid, new attributes *group* and *permissions* should be introduced (together with other attributes like *submitter*, *file size* and *md5 checksum*). This requires a new client command line tools as well as introduces some additional tasks to the control thread.

##### Client

A new command lines tool shall be named:

```
digs-make-public [-r] <LFN>  
digs-make-private [-r] <LFN>
```

Those commands will allow any collaborators of the group the file belongs to, to make them private or publicly available. Initially, there will be only two groups *ukqcd* and *ildg* accompanied with permissions *private* or *public*, but this could be extended in the future.

Note that local users, who are not part of either collaboration, but who have access to the machine hosting data, will have access to the public files. We do not consider this to be a significant issue, as all those files are considered as public data anyway.

The command will have an optional ‘-r’ *recursive* switch, in line with what most of the other commands already have. It seems that our users most often operate on a collection of data as a whole, and it could get tedious having to set the group of every file in a dataset individually.

Issuing the command *digs-make-public* or *digs-make-private* should send a message to the control thread that there are files for which the permissions must be changed. Within this message there should also be the information about which *group* the user belongs to, which can be extracted from the UNIX account the user is logged on to. This mechanism should work in the similar way as when deleting the files from the grid, but rather than removing files, their permissions should be changed.

##### Control thread

The control thread should receive the message from the client and then store the files that are to be changed in its pending list. Having received the message, it should modify the *permissions* attribute (possible values currently are *private* or *public*) in the RLS entry for this particular LFN. Subsequently, the control thread should find all the physical copies of data with this LFN and modify their file permissions. Once this is complete, it should remove the entry from its pending list.

A mechanism for changing the file permissions should be created. It is necessary for three different use cases: when a file is added onto the grid, when it is replicated, and when a user requests to make it private or public.

We should create several methods which would be utilised for carrying out operations on files permissions:

`setPermissionsOnFile(group, file)` – sets up appropriate permissions on the specified file

`setPermissionsOnAllCopies(group, LFN)` – iterates over every physical copy of the LFN and calls on it `setPermissionsOnFile()`.

Additionally, we will also need getters and setters methods for the RLS attribute *permissions* and *group*:

`getGroupFromRLS(LFN)` – obtains a group that specified LFN belongs to.

`setPermissionsInRLS([public | private], LFN)` – sets appropriate permissions for the LFN.

### 4.1.2 Addition

In order to support putting files onto the grid supplied with new information some changes are required on both client and the control thread sides.

#### Client side

A new command should be created, which extends *put-file-on-qcdgrid*. It shall be named *digs-put-file*. In addition to its original input arguments *pfn* (physical file name) and *lfn* (logical file name), it should also have a *submitter* (a person who is submitting the files) and a *group* (is the file publicly available or private to some group). This command could look like: *digs-put-file (pfn, lfn, permissions)*. Additionally, this command should also obtain the *submitter* from the x509 certificate, the *group* from the UNIX account, the *size* of the file which is submitted (using `getFileLength()` in *misc.c*) and its *MD5 checksum* (using `computeMD5Checksum()` in *misc.c*). Moreover, the *time of submission* should be recorded.

Before a file is added to the grid a message should be sent to the control thread with the information presented in Table:

PFN	SUBMITTER	GROUP	PERMISSIONS	SIZE	MD5	SUBMISSION TIME
filename	DN of submitter	ukqcd	private or public	1234185	43b63a6397	nr of seconds since 1970

If the message is successfully received by the control thread, the file is copied to the NEW directory and its name is changed to reflect its location structure. This is an existing mechanism to avoid keeping track of complicated directory dependencies. All slash characters `'/'` in the filename are substituted with `"-DIR-"` tokens. A verification of checksum of the original file and the one copied over should be performed to complete this operation.

#### Control thread

When control thread receives a message from the client trying to add to the grid, it should store the attributes (submitter, group etc.) into a List File.

Periodically, storage nodes should be scanned for new data (which will be located in the NEW directory). For each new file that is found, the directory structure should be reinstated – by replacing instances of `"-DIR-"` token with slash character `'/'`. Then the List File should be checked for an instance of the newly discovered file. If no instance is found, then a warning message should be issued and the file left untouched. Otherwise, the permissions on the file should be updated (that is, group and public/private status should be set) and the file should be moved into persistent storage on the node.

Next, an entry containing LFN and PFN should be added to the RLS accompanied with the additional information (submitter, group, etc.) as attributes.

A regular check should be performed (on a daily basis) to identify and remove any outdated entries in the attribute list file which might appear if there were problems when adding to the grid (based on the *submission time*).

Methods requiring changes:

`newToNormalName()` – in this method, **before** copying a file from NEW directory to the grid, run `chown qcdgrid:group` and `chmod g+r` and `o+r` or `o-r` depending on the *group* and *permissions*.

`registerFileWithRc(host, realLfn, submitter, group, filesize, MD5)` – this method should also take additional parameters: *group*, *file size*, *submitter* and *MD5*.

### 4.1.3 Replication

Some additional modifications are also required to the replication mechanism resulting from the usage of new attributes. Having copied a file to another storage node, it must be verified if it has not been corrupted. So far, the MD5 checksums were compared on both, the original and the copied file (`thirdPartyVerify()` in `misc.c`). However, from now on, the checksum of the copied file should be matched against the attribute *MD5* stored in the RLS. This functionality should be implemented in a new method named `verifyWithRLS()`, which replaces `thirdPartyVerify()`. It should employ `getMD5FromRLS()` method that would query the RLS and obtain the *MD5* attribute.

Furthermore, having established that the file has been copied successfully, its permissions must be changed according to the *group* and *permissions* attribute. To achieve this, `setPermissionOnFile()` should be utilised.

### 4.1.4 Deletion

Existing implementation allows deleting files only to privileged users like grid administrators. This feature makes the grid vulnerable to become full of temporary or unwanted data. A new functionality could be added, which would let the users who originally submitted some files (*submitters*), to delete them. This would require a change in the implementation of `canDeleteFile()` in `background-new.c`

## 4.2 Establish modifications to existing command-line tools for DiGS 2.0

### 4.2.1 Digs-list

As described in the earlier section, the command `put-file-on-qcdgrid` has been modified and renamed to `digs-put-file`. Here, we consider corresponding changes and extensions to the `qcdgrid-list` command. Following the change in the name of the software, we will rename this function to `digs-list [arguments] [filename | wildcard]`. As there will be now more information available about the files stored on the grid, we should be able to obtain this information with this command-line tool. Below are the new arguments that should enable new features.

- ‘-l’ prints all the below arguments (like UNIX ‘ls -l’)
- ‘-n’ number of copies of each file
- ‘-w [storage-node]’ whereabouts (locations) of where the file is stored or, if specified, all the files stored on *storage-node*.

- ‘-g [*group*]’ group the file belongs to or, if specified, all the files belonging to the provided *group*
- ‘-p [*public / private*] [*group*]’ permissions of the file or, if specified, all the files *public* or *private* for a certain *group*
- ‘-c’ stored MD5 checksum
- ‘-s’ stored size
- ‘-o [*submitter*]’ original submitter or if specified, all the files uploaded by the *submitter*
- ‘-h’ or ‘--help’ shows usage information

An example invocation of *digs-list -l public.\** could produce the following output:

LFN	group	permissions	“submitter”	size	MD5	(nr of copies)	[locations]
public.pdf	ukqcd	public	“my DN”	1290	sd12ead1	(2)	[qcdgrid2.epcc.ed.ac.uk qcdgrid3.epcc.ed.ac.uk]

If the implementation of *digs-list* utilised new getter methods to contact the RLS for every single LFN, it could introduce a major performance issue. Therefore, in order to obtain all the values of the new attributes faster (in one query), the RLS *attribute search* command should be used (as in *getAllFileDisks()* in *replica.c*). Then, the response should be parsed and the desired information extracted and output to the screen.

## 4.2.2 Digs-catalogue-rebuild

Another command that requires modifications is *rebuild-qcdgrid-rc*, which will be renamed to *digs-catalogue-rebuild*. As there are many modifications to be introduced we will describe a new algorithm for the catalogue rebuild.

As a first step, a list of all the LFNs from the old catalogue together with all their attributes should be stored. Secondly, a scan of all the storage nodes should be performed to identify all the files present on the grid.

For each file found, if the file’s LFN is in the old catalogue, it should be added to the new catalogue along with all of its attributes (assuming that it is the first such file to be found). All further copies of this file should simply have their location recorded to the entry in the catalogue.

However, if a file’s LFN was not in the old RLS, the attributes have to be obtained in a different way. The *submitter* should be set to *unknown*. Setting up a default *group* and *permissions*, could be a bit tricky, so before running the *digs-catalogue-rebuild* command, the program will prompt for the default values to be used if those entries are missing. If some of those files should belong to a different group, then their permissions could be manually modified by running *digs-make-public* or *digs-make-private* described in Section 4.1.1. To keep the permissions on the files consistent, those files’ LFNs should be also added to the *RLS-group-queue*. This will invoke the control thread to modify them appropriately. In order to deal with attributes like *MD5 sum* and *file size* some additional steps must be followed. A temporary list of new LFNs should be created with associated physical locations. Once the list is complete, we could obtain the desired values. If there was only one copy of the file, we would simply take its *MD5 sum* and *file size*. However, if there were more copies, we would have to verify if all the copies are consistent and only then store their *MD5 sum* and *file size*. In case of any inconsistencies, an error should be reported, and the process of catalogue rebuild should be stopped. Additionally, the content of the temporary list could be presented.

Only if there are no more files to iterate through and all operations have completed successfully, the new catalogue should be used. Instead of discarding the old list, it should be archived and the time it was retired recorded. This could serve as a backup facility.

### 4.3 Design new command-line tools

Another new command that it would be useful to provide is *digs-version*. It could simply output the contents of a file called VERSION located in a software root directory, as presented below:

```
QCDgrid version 1.5
```

```
=====
```

```
This distribution contains
```

- Version 1.5.0 of the datagrid and job submission software;
- Version 1.5.0 of the metadata browser software.

## 5 Design for ILDG File Catalogue and File Transfer Service

### 5.1 Design for F/C Web Service

In order to build the ILDG File Catalogue Web Service, we would require a mechanism for proxy delegation. We could either implement our own or use an existing implementation. We investigated the available solutions for delegation of credentials and identified a good candidate: gLite Delegation Service [15].

#### 5.1.1 gLite Delegation Service operations

The gLite Delegation Service (DS) has been developed as part of the EGEE project. It uses the same mechanism to transfer a delegated proxy from a client to a server (delegation protocol) as the Globus Toolkit. However, as it is web service based, it is much more light-weight than the Globus implementation, which requires Web Services Resource Framework (WSRF).

It was decided that it would be more advantageous to use an existing solution rather than to implement our own. Moreover, we would not introduce any modifications to DS code and use it as it comes. This approach could save us development time (for any code manipulations) and, additionally, it would allow us to easily upgrade to a newer version of DS in the future.

A description of DS interface is presented in [16]. In this section we only describe the most important methods of the DS which allow the proxy to be transferred and destroyed: *getNewProxyReq()*, *putProxy()* and *destroy()*.

##### 5.1.1.1 getNewProxyReq

This method starts the delegation procedure by asking for a certificate signing request from the server. The server answers with a certificate signing request which includes the public key for the new delegated credentials. The method *putProxy()* has to be called to finish the procedure.

Specifically:

1. Generate a delegation ID by hashing the client DN.
2. Check if the delegation ID already exists in the storage-area. If it does, check existing info (DN) against client info. Throw exception if they do not match, because then this is the rare case of hash collision, i.e. two different clients are mapped to the same delegation ID.
3. Create a new private/public key-pair.
4. Generate a new certificate request.
5. Store private key and certificate request in storage-cache-area, along with the requesting DN.

```
NewProxyReq getNewProxyReq()  
    throws DelegationException;
```

**Returns:** The server side generated ID of the new delegation session and the new RFC 3280 style proxy certificate request in PEM format with Base64 encoding.

**Throws:** DelegationException – there were already credentials associated to the delegation ID.

### 5.1.1.2 putProxy

This method finishes the delegation procedure by sending the signed proxy certificate to the server. Specifically:

1. Check if a delegation ID was provided. If not, generate a delegation id by hashing the client DN.
2. Check if the delegation ID already exists in the storage-area. If it does, check existing info (DN) against client info. Throw exception if it does not match.
3. Check, if client information matches proxy information.
4. Check given proxy against private key of delegation ID in storage-cache-area. If they do not match, throw exception.
5. Store proxy in storage-area and clean up the storage-cache-area.

```
putProxy(string delegationID,  
         string proxy)  
    throws DelegationException;
```

#### Parameters:

- delegationID – the ID of an already existing delegation session, initiated by `getNewProxyReq()`.
- Proxy – RFC 3280 style proxy certificate, signed by the client, in PEM format with Base64 encoding.

**Throws:** `DelegationException` - there were no cached credentials associated to the delegation ID (`getNewProxyReq()` was not called previously), or the client's DN do not match the stored ones, i.e. the client is not authorized.

### 5.1.1.3 destroy

This method destroys the delegated credentials associated with the given delegation ID immediately. If there was no delegation ID, then generate one by hashing the client DN.

```
destroy(string delegationID)  
    throws DelegationException;
```

#### Parameters:

- delegationID – the ID of an already existing delegation session to be destroyed.

**Throws:** `DelegationException` – there were no credentials associated to the delegation ID, or the client's DN do not match the stored ones, i.e. the client is not authorized.

## 5.1.2 File Catalogue specific operation – getURL

Having implemented proxy delegation, we can now use the delegated proxy certificate to query the File Catalogue. To do this, a client has to call `getURL` method providing a list of LFNs for the files it wants to find the locations of. The File Catalogue should respond with zero or more URLs corresponding to each of the LFNs we queried about. The list (could be of zero length) of results will be returned to the client. For more information please refer to [20].

### getURL

This operation allows sending a list of LFNs to the file catalogue. For each LFN a result is returned which consists of a list of URLs and an access status code.

## Input: getURLRequest

lfnList	List of ILDG compliant logical file names
---------	---

```
<wsdl:message name="getURLRequest">
  <wsdl:part name="lfnList" type="impl:TArrayOfString"/>
</wsdl:message>
```

## Output: getURLResponse

resultList	List of results
numberOfResults	Number of results returned
statusCode	Operation status code (see end of this section)
queryTime	Time at which query was executed on server side
lfn	Logical file name
accessCode	File access code (see end of this section)
numberOfSURL	Number of SURLs
surl	SURL

```
<wsdl:message name="getURLResponse">
  <wsdl:part name="getURLReturn" type="impl:TGetURL"/>
</wsdl:message>
<complexType name="TGetURL">
  <sequence>
    <element name="resultList" nillable="true" type="impl:TArrayOfSURLInfo"/>
    <element name="numberOfResults" type="xsd:int"/>
    <element name="statusCode" type="xsd:string"/>
    <element name="queryTime" type="xsd:dateTime"/>
  </sequence>
</complexType>
<complexType name="TArrayOfSURLInfo">
  <sequence>
    <element name="result" maxOccurs="unbounded" nillable="true"
      type="impl:TSURLInfo"/>
  </sequence>
</complexType>
<complexType name="TSURLInfo">
  <sequence>
    <element name="lfn" type="xsd:string"/>
    <element name="accessCode" type="xsd:string"/>
    <element name="numberOfSURL" type="xsd:int"/>
    <element name="surl" maxOccurs="unbounded" nillable="true"
      type="xsd:string"/>
  </sequence>
</complexType>
```

**Status and Access Codes**

Operations returning a status code will return only one of the following operation status codes:

FC SUCCESS	Operation has been completed successfully
FC INVALID REQUEST	Bad arguments, e.g. badly formulated query
FC FAILURE	Unspecified (possibly implementation specific) failure

The access code describes any of the following file access status:

FC ACE ALLOW	Access allowed
FC ACE DENY	Access denied
FC ACE UNKNOWN	It is not known whether access is allowed or denied

### 5.1.3 Server-side implementation of getURL

For this purpose, we could reuse some of the functions located in *rls.c*. We need to achieve the following:

1. Use the delegated proxy to query the RLS and obtain the list of nodes which store the copies of the desired LFN. We could use the methods listed below:
  - `char *getFirstFileLocation(char *lfn)` – returns a first location
  - `char *getNextFileLocation()` – returns another location or NULL if there are no more copies

Note that we need to set up the environment variable `X509_USER_PROXY` to the delegated proxy before querying the RLS (see Section 5.1.3.1).
2. Make sure that the user is allowed to download this file (ILDG collaborators should be only granted access to files classified as “public”). If not, then return a meaningful error message.
3. Otherwise, find the exact physical file location (URL) on each of those nodes. To construct the complete path we could use:
  - `char *constructFilename(char *host, char *lfn)` – returns a full path to a file for a particular LFN on a particular host
4. Return the list of all URLs

#### 5.1.3.1 Employing delegated proxy to query the RLS from a command line

It has been noticed that when querying the RLS from the command line, there is no argument that would specify the alternative proxy certificate location that is to be used. The RLS reads the environment variable `X509_USER_PROXY` which is a location of proxy employed by the whole Globus Toolkit. If this variable points to the user’s delegated proxy, then other Globus components will also use this proxy certificate, which is not what is intended.

A workaround to this is to write a shell script which first exports the variable `X509_USER_PROXY` and then calls the RLS. The invoked script runs as a sub-shell, so the RLS uses the delegated proxy but the variable in the upper shell is not modified. A sample script is presented below:

```
#!/bin/sh
export X509_USER_PROXY=/path/to/delegated/user/proxy
globus-rls-admin -S rls://server.epcc.ed.ac.uk
```

An alternative solution would be to call the RLS using Java API, which has a way of specifying which proxy to use.

## 5.2 Direct file access

Once we know the physical file locations of a particular LFN we are looking for, we can use GridFTP to copy it from the storage nodes to any machine we require. Ideally the access to the files should be granted to any valid ILDG members. However, there might be some firewall issues at this point, as a few of the storage nodes have their ports blocked for the GridFTP transfers.

If it became a major issue, it could be solved by implementing a File Transfer Service.

## 5.3 File transfer service

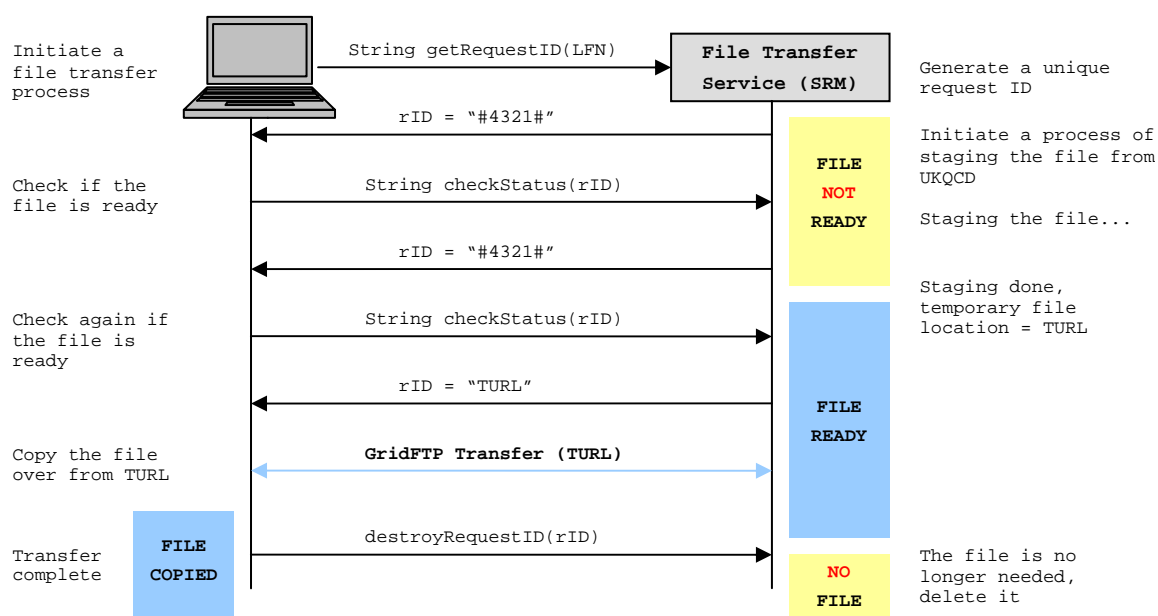


Figure 2: SRM type interface at work

To solve the firewall issue, stumbled on in the direct file access, we could provide a File Transfer Service (FTS). We could dedicate one of our machines that can access all storage nodes and have open ports in its firewall and use it as a front-end to our grid.

A possible mechanism of how it could work is presented in Figure 2. It employs a standard interface of Storage Resource Management (SRM) [17] used for accessing files from tape storage. In our case, we will consider UKQCD grid as tape storage. It can be assumed so, because it takes some time before a file is copied over from a storage node onto the local machine.

An ILDG user would ask the FTS for a file providing its LFN. They would get the request ID which would be used to track every enquiry. FTS would stage the file from the UKQCD grid into a temporary location of the front-end machine. It could be done by employing just one DiGS command, where a TURL is a temporary file location:

*digs-get-file <LFN> <TURL>*

DiGS software would search for the best available copy of the file and download it. There would be no firewall problems at this point. However, there still will be a requirement for delegation of grid

credentials, so gLite Delegation Service (Section 5.1.1) could be used before the actual file transfer on the back-end of FTS takes place. Using this mechanism would make the initial effort of contacting the RLS redundant. A client could query the status of the file staging with its request ID at any time. Once the file is ready, it could be downloaded by the client. If the client completes the transfer, it notifies the FTS, which can then delete the, no longer required, file. There should be some additional process of retiring the old temporary files, just in case if the clients would not notify the FTS that they completed the download.

We could either implement the SRM interface ourselves, or use software that already exists. There are currently two implementations of SRM available: dCache [18] and Disk Pool Manager (DPM) [19]. DPM is a light-weight implementation and therefore it does not support tape storage, what makes it unsuitable for our purpose.

An initial investigation has shown that it would be possible to employ dCache for our FTS. As it communicates with the tape storage devices via customised scripts, we could adapt one of them, to interact with UKQCD grid instead. The investigation is currently progressing.

## 6 Conclusions

This document describes the design for the *ILDG File Catalogue*, a key component of the ILDG infrastructure. It also reflects on the security aspects concerning user authentication and authorisation to perform any operations on *UKQCD Grid*. In addition, it discusses any changes that are necessary before a version 2 of the QCDgrid software – which will be called DiGS (Distributed Grid Storage) – can be released. And finally it describes the *ILDG FC Web Service* and *File Transfer Service*.

To sum up, this document establishes the next step towards the WP 4.5 “Implementation of ILDG File catalogue and basic Storage Element access”.

## 7 References

- [1] EGEE, *gLite – Lightweight Middleware for Grid Computing*. Project homepage at <http://glite.web.cern.ch/glite/> (2006).
- [2] Globus Alliance, Globus Toolkit. Homepage at <http://www.globus.org/>.
- [3] International Lattice Data Grid. Homepage at <http://www.lqcd.org/ildg/>.
- [4] M. Sato, *Lattice QCD Data Grid Middleware: The Metadata Catalogue*, Presentation to ILDG Workshop (2004). Electronic copy available at <http://www.rccp.tsukuba.ac.jp/workshop/ILDG-4/pdf/MitsuhsaSato.pdf>.
- [5] UKQCD Collaboration. Homepage at <http://ukqcd.epcc.ed.ac.uk/>.
- [6] UKQCD, *QCDgrid: Probing the building blocks of matter with the power of the Grid*, QCDgrid project homepage at <http://www.gridpp.ac.uk/qcdgrid/>.
- [7] T. Yoshie, *Status of ILDG Activity*, Presentation to ILFTN (Edinburgh, 2005). Electronic copy available from [http://www.nesc.ac.uk/talks/464/Session7/ILFTN2\\_yoshie.pdf](http://www.nesc.ac.uk/talks/464/Session7/ILFTN2_yoshie.pdf).
- [8] Pool Accounts patch for Globus, <http://www.gridsite.org/gridmapdir/>
- [9] GT 4.0 RLS: System Administrator's Guide, <http://www.globus.org/toolkit/docs/4.0/data/rls/admin-index.html>
- [10] R.H. Ostrowski and M.G. Beckett, *WP 4.2 Functional specification of ILDG File Catalogue*, QCDgrid Project Deliverable (January 2007).
- [11] R.H. Ostrowski and M.G. Beckett, *WP4.3 ILDG File Catalogue Security*, QCDgrid Project Deliverable (February 2007).
- [12] GridSite, Delegation protocol, [http://www.gridsite.org/wiki/Delegation\\_protocol](http://www.gridsite.org/wiki/Delegation_protocol)
- [13] Gridsite Delegation, <https://twiki.cern.ch/twiki/bin/view/EGEE/GridSiteDelegation>
- [14] GT 4.0 Security: Key Concepts, <http://www.globus.org/toolkit/docs/4.0/security/key-index.html>
- [15] GT4 Delegation Service Developer's Guide, <http://www.globus.org/toolkit/docs/4.0/security/delegation/developer-index.html>
- [16] GridSite Delegation Interface Description, <http://egee-jra1-data.web.cern.ch/egee-jra1-data/GridSiteDelegation/HEAD/doc/glite-security-delegation-interface/DelegationInterface.html>
- [17] SRM GridPP Wiki, <http://www.gridpp.ac.uk/wiki/SRM>
- [18] DCache GridPP Wiki, <http://www.gridpp.ac.uk/wiki/DCache>
- [19] Disk Pool Manager GridPP Wiki, [http://www.gridpp.ac.uk/wiki/Disk\\_Pool\\_Manager](http://www.gridpp.ac.uk/wiki/Disk_Pool_Manager)
- [20] ILDG FC-WS Operations (0.2), ILDG MWWG, 14 May 2007