POOL FileCatalog Documentation

Authors: Zhen Xie, Maria Girone
Date: 20 April 2004
Version: 1.1
Template Version: 1.5

Document Status Sheet

Title: Pool File Catalog Component Description		
ID:		
Version	**Date**	**Reason for change**
0.1	22 Aug 02	New document
0.2	10 Nov 02	MySQL and XML implementations descriptions
0.3	18 Dec 02	EDG implementation and high level test description
0.4	27 Feb 03	Interface and command-line tool changes. Add more use-cases.
0.5	9 May 03	POOL-1-0-0 release
0.6	4 June 03	Implement consistent transaction protocol for all backends
0.7	24 June 03	POOL-1-1-0 release
0.8	10 Sept 03	POOL-1-3-0 release, add Python interface to the component
0.9	12 Nov 03	POOL-1-4-0 release, interface and schema changes
1.0	5 March 04	POOL-1-6-0 release, interface changes
1.1	10 April 04	POOL-1-6-2 release, add changes to command line tools

1 Description of the component

1.1 Purpose of the component

The file catalog component in POOL is responsible for maintaining consistent list of accessible files together with their unique and immutable file Ids. Its main user is the storage components who consult the file catalog when a new file is to be accessed.

The file catalog is also used to store some file related metadata as for example the logical filename which in contrast to the file ID may contain a memorisable text string.

The relationship between the file Ids and file names is the following:

When a file is created for the first time, it is assigned to a unique and immutable file Id and it is also assigned to a physical file name which identifies its physical location. Later on, different copies of the file may be created. Each replica of the file has its own distinct physical file name but the same file Id as its master copy. In another word, FileID is the logical identifier of a file and all its replicas. Due to the generation mechanism, the format of the FileID is not easy for user to read and to remember, as a consequence, human readable and memorisable alias may be provided to a FileID. These alias are called logical file names. This view is consistent with that of the replica management service within the context of the EU Data Grid (EDG) Project[1].

The file catalog component provides an interface to two types of users. First of all, it provides an interface for the storage components in POOL to register and lookup a file inside the application process. Command line tools are provided to handle catalog operations outside the application process, as assigning Logical File Names, registering files in the catalog, appending one catalog to the other, etc.

Three different implementations are provided:

- An XML-catalog can be used and/or produced by a single user inside one job. It is useful when user wants to run the application disconnected from the network. The content of a XML catalog or a part of it can be published to the other two types of catalogs. A part of the other two types of catalogs can be extracted into a XML catalog.
- A native MySQL catalog can be used in a production farm. It can handle multiple users and multiple jobs. However, it is not on the Grid. The content of the MySQL catalog or a part of it can be published to the Grid-aware catalog. A part of the other two types of catalogs can be extracted into a MySQL catalog.
- EDG-RLS based catalog is used by the entire Virtual Organization(VO). The EDG Project will provide the Replica Management Service, which controls files that belong to a VO. In particular, the Replica Location Service (RLS) component [2] maintains information about the physical locations of files, while the Replica Metadata Catalog (RMC) component [3] provides the information on the logical file names and metadata. The pool file catalog component provides an interface to the EDG-RLS and EDG-RMC for the Grid-aware applications.

1.2 Known problems and restrictions

The details of the interaction between the file catalog and grid components still need to be defined in more detail. In the version 1.0 release of the component we still make a few simplifying assumptions to get started. Any or all of those may not hold in the longer term

1) POOL will not directly create file replicas and assumes to use the first replica of each file.
2) POOL will assume that in case of several existing replicas for a given GUID the first one (master copy) can be used for writing/appending
3) We do not yet check authorization on the catalog level. (EDG RLS does not provide this functionality yet).
4) The implementation of the EDG container is still under development due to some missing feature of the API interface in use in this version. As a result, the cache size of the container will be the limit on the number of entries in the container.

Limits of MySQL database: limit on the length of the data type varchar is 255. This restricts the length of physical and logical filenames in the current implementation; due to the limit of the blob data type, the the length of string type metadata entries is limited to 64kB.

Inconsistency between the catalog implementations: upon connection, the XML implementation will attempt to create a new catalog if it doesn't already exist while the connection to MySQL and EDG catalogs will fail if the backends don't exist.

1.3 Repository of the component

/pool/FileCatalog	repository of the catalog interface and common utility classes code
/pool/MySQLCatalog	repository of MySQL catalog code
/pool/XMLCatalog	repository of XML catalog code
/pool/EDGCatalog	repository of EDG catalog code
/pool/Utilities/FileCatalog	repository of catalog command-line tools source code
/pool/Scripts/FileCatalog	repository of scripts used by file catalog
/pool/PyFileCatalog	repository of the Python interface of the component

2 User guide

2.1 Composite Catalog concepts

Starting from POOL_1_6_0, composite catalog features are supported. The file catalog the user operates on consists of one master catalog which is read/writable and any number of read-only catalogs. One can specify the master catalog and add read-only catalogs using the contact strings.

The writing operations on the catalog are automatically performed on the master catalog; while the lookup operations lookup first in the master catalog and then the read-only ones in the order defined by the user. For bulk lookups, results found in all the leaf catalogs are returned. For lookup operations which expect a single result, the search stops when the first result is found and returned.

2.2 How to construct the catalog contact string

To obtain the connection to the catalog, a contact string of the format:

[prefix_][protocol]://[username]:[password]@[host]:[port]/[path]

or

[prefix_]file:path

The **[prefix_]** field is used to distinguish different catalog implementations. In case of absence, a local XMLCatalog will be used.

The supported prefix are: **xmlcatalog_** , **XMLFileCatalog_**, **mysqlcatalog_**, **edgcatalog_**

The supported protocols are: **mysql** for MySQL catalog; **http** for XML and EDG catalog; **ftp** for XML catalog; **file** for XML catalog.

Some examples of the contact strings for different catalogs are shown as follows:

- **MySQL**:

  mysqlcatalog_mysql://@lxshare070d.cern.ch:3306/testFCdb

  For the MySQL catalog, the **[path]** field represents the database name. The **[username]** field should be the username of the database. In case of absence, the login name of the user will be taken. The default value for the **[port]** field is 3306.

- **XML**:

  xmlcatalog_file:/tmp/FileCatalog.xml

  file:/tmp/FileCatalog.xml

  xmlcatalog_http://pc01.cern.ch/file001, if the catalog is at remote site and read only

- **EDG:**

  edgcatalog_http://rlstest.cern.ch:7777/edg-replica-location/services/edg-local-replica-catalog

2.3 How to construct the query string

The component supports query on the file metadata. In the current release, the query is a plain string consists of the attribute, "=" or "like" predicates and the desired value of the attribute. The wildcard "%" on the attribute value is allowed. Due to the string implementation of the XML and EDG catalogs, numerical queries are not supported in this release. All the string values must be quoted within a pair of single quotes. Example of some query strings: "jobid='sim101'", "owner like '%me%'"

The query strings can be passed to the command-line tools using the **–q** option or passed to the catalog API as argument of the lookup methods.

The query attribute can be either the metadata or 'pfname', 'lfname' and 'guid'.

FileID(GUID) is and should not be explicitly defined as an attribute because it is implicitly defined when the metadata schema is created. It is invisible to the user.

In this release only 'AND' logic is supported by all implementations, e.g. "jobid='sim101' AND owner line '%me%'".

2.4 How to use command-line tools of the component

The command-line tools provided by the File Catalog are in the */pool/Utilities/FileCatalog* repository.

General options:

-h print help message

-u the catalog contact string. If absent, the contact string is picked up from the environment variable **POOL_CATALOG**. The contact string specified by *–u* option overrides that taken from the environment variable. To specify more than one catalogs in lookup operations, one should separate the contact string of each leaf catalog with a white space and close the entire string with double quotes.

-l LFN

-p PFN

-m customized cache size when using the catalog container, if this option is not given, the default cache size 1000 is assumed.

1. Register PFN

FCregisterPFN –p pfname [-u uri –t filetype -h]

Registers a PFN and assigns a unique FileID to it.

Warning: You should only run this command for testing purpose, otherwise your real FileID will be lost. A file can be registered only from inside the job.

-t file type

2. Register LFN

FCregisterLFN –p pfname –l lfname [-u uri -h]

Register the LFN (specified by **–l** option) to the PFN (specified by **–p** option).

3. Register a replica file name

FCaddReplica [–p pfname] [-g guid] –r replica [-u uri -h]

Register a replica file PFN (specified by **–r** option) to the master file PFN (specified by **–p** option) or directly to the Guid of the master file(specified by **–g** option)

4. Lookup PFNs

FClistPFN [-l lfname –q query –m cachesize –u uri -t -h]

-l option: list all PFNs with given LFN

-q option: list all PFNs satisfy the query

If no option is given, all PFNs are displayed.

-t option: print PFNs in long format: PFN, filetype

-u option: a contact string containing multiple catalogs separated by whitespaces are accepted. E.g. "mysqlcatalog_mysql://user@localhost/mycat1 xmlcatalog_file:mycat2.xml" .
The first catalog is searched first.

5. Lookup LFNs

FClistLFN [-p pfname –q query –m cachesize –u uri -h]

-p option: list all LFNs with given PFN

-q option: list all LFNs satisfy the query

If no option is given, all LFNs are displayed.

-u option: a contact string containing multiple catalogs separated by whitespaces are accepted. E.g. "mysqlcatalog_mysql://user@localhost/mycat1 xmlcatalog_file:mycat2.xml" .
The first catalog is searched first.

6. Lookup Meta Data

FClistMetaData [-l lfname –p pfname –q query –u uri –m maxcache –h]

-l option: list metadata associated with the file with given PFN.

-p option: list metadata associated with the file with given LFN.

-q option: list all MetaData entries satisfy the query

If no option is given, all metadata entries are displayed.

-u option: a contact string containing multiple catalogs separated by whitespaces are accepted. E.g. "mysqlcatalog_mysql://user@localhost/mycat1 xmlcatalog_file:mycat2.xml" .
The first catalog is searched first.

7. Describe the file meta data definition

FCdescribeMetaData [-u uri -h]

Describe meta data in the catalog.

Format of the output:

((attribute1_name, attribute1_type), (attribute2_name, attribute2_type))

8. Define the meta data specification

FCcreateMetaDataSpec [-F -m metadatadefinition –u uri –h]

Create meta data specification specified by the **–m** option.

Format of the input:

"*(attribute1_name, attribute1_type), (attribute2_name, attribute2_type) "*

-F if a meta data schema already exists, drop the old one and create a new one.

9. Update the meta data specification

FCupdateMetaDataSpec [-F -m metadatadefinition –u uri –h]

Update meta data specification specified by the **–m** option.

Format of the input:

"*(attribute1_name, attribute1_type), (attribute2_name, attribute2_type) "*

-F if a meta data schema already exists, drop the old one and create a new one.

10. Insert meta data

FCaddMetaData [-p pfname –l lfname -m metadata –u uri -h]

Insert file meta data specified by the **–m** option associated with file with given PFN or LFN. Format of the input:

"((attribute1_name, attribute1_value), (attribute1_name, attribute2_value))"

-p insert metadata associated to the PFN specified by this option.

-l insert metadata associated to the LFN specified by this option.

11. Delete the selected PFN entry

FCdeletePFN [–p pfname –q query –u uri –h]

-p delete PFN entry specified by this option.

-q delete PFN entries selected by the query specified by this option.

If the pfn is the last one in the catalog, the entire mapping is deleted which has the same functionality of the command FCdeleteEntry -p

12. Delete the selected LFN entry

FCdeleteLFN [–l lfname –q query –u uri –h]

-l delete LFN entry specified by this option.

-q delete LFN entries selected by the query specified by this option.

13. Delete the selected MetaData entry

FCdeleteMetaData [–p pfname –l lfname –q query –u uri –h]

-p delete MetaData entry associated with the PFN specified by this option.

-l delete MetaData entry associated with the LFN specified by this option.

-q delete MetaData entries selected by the query specified by this option.

14. Delete the selected PFN-LFN-MetaData mapping

FCdeleteEntry [–p pfname –l lfname –q query -u uri -h]

Delete the PFN-LFN-MetaData mapping of a file.

-p delete the mapping associated with the PFN specified by this option.

-l delete the mapping associated with the LFN specified by this option.

-q delete the mapping selected by the query specified by this option.

15. Drop all the metadata and its specification

FCdropMetadata [-u uri -h]

16. Extract a fragment from the source catalog and attach it to the destination catalog

FCpublish -d destinationcatalog [–q query –s metadatadef –m cachesize -u sourcecatalog -h]

The destination catalog is specified by the **–d** option.

-u option: a contact string containing multiple catalogs separated by whitespaces are accepted. E.g. "mysqlcatalog_mysql://user@localhost/mycat1 xmlcatalog_file:mycat2.xml"

-q option: extract/publish catalog fragment selected by given query.

-s option: redefine destination catalog metadata schema. If an empty string is given to this option, no metadata will be imported to the destination catalog.

If no query is specified, the entire source catalog will be appended to the destination catalog. The operation is atomic.

17. Rename PFN

FCrenamePFN –p pfname –n newpfname [-u sourcecatalog -h]

Rename the PFN (specified by the **–p** option) to the new one (specified by the **–n** option).

2.5 C++ API of the component

Class **IFileCatalog** is the interface of the component. It provides functions of the following types:

- Composite Catalog manipulations
- Connection and transaction control functions
- Cross catalog operations

The master catalog in the composite catalog is set by setWriteCatalog(). The addReadCatalog() method is used to add read-only catalogs into the composite catalog. The iteration of the leaf catalogs is achieved by getReadCatalog(), nReadCatalogs() and getWriteCatalog().

The catalog has two transaction states: in transaction and between transactions. Transaction starts with start() and ends with commit() or rollback(). Methods start() and commit() or rollback() should always be called in pairs. Exceptions will be thrown if these methods are not in pairs. Commit() methods takes IFileCatalog::CommitMode as arugment. REFRESH mode indicates the XML parser (for the XML catalog) will be reinitialised at the next start() method while ONHOLD mode indicates that parser states will not be changed at the next start() method. The default value of the argument is the REFRESH mode.

Between connect() and start(), start() and disconnect() are the between transaction states. Exceptions will be thrown when catalog operations are called at the between transaction states. Methods connect() and disconnect() also should be called in pairs. Exceptions will be thrown if connect() or disconnect() method is called twice in a row.

User can import a fragment of another catalog into the current catalog through IFileCatalog::importCatalog() method. The catalog fragment to be imported is selected by query.

Class **IFCAction** is the interface for catalog operations. The interface of this class is designed to be used by other POOL components. Its subclasses **FCregister**, **FClookup** and **FCAdmin** are responsible for general user register, lookup and schema manipulations. All methods in IFCAction class are also available in the subclasses. Each action instance is associated with one composite catalog by the method IFileCatalog::setAction().

The component supports associating metadata with the guid. The purpose of the metadata is to ease the file lookup and the catalog fragment selection. The metadata schema can be created, updated or droped. When dropping the metadata schema, all the file metadata will be lost as well.

The interface provides method to delete LFN, PFN and metadata entries in the catalog. However, one should use these methods with caution, if the selected PFN is the last one in the catalog, the entire mapping is deleted.

Class **IFCContainer** provides an interface to iterate on catalog entries. Only sequential iteration is supported through the method hasNext() and Next(). For scalability reason the results are retrieved first into a cache with default size of 1000 entries. The cache is used repeatedly until all results are retrieved, new batch of entries will overwrite the old entries in the cache. The iterating state can be reset through the reset() method.

Each container is bound to a given filecatalog when created. There are four types of containers: **PFNContainer**, **LFNContainer**, **MetaDataContainer** and **GuidContainer**.

An example of application code is shown below:

```
IFileCatalog* mycatalog;

mycatalog->setWriteCatalog("file:test.xml");

IFileCatalog::FileID fid;

FCregister a;

mycatalog->setAction(a);

mycatalog->connect();

mycatalog->start();

a.registerPFN("aPFN","fileformat",fid);

a.registerLFN("aPFN","lfn:aPFN");

mycatalog->commit(IFileCatalog::ONHOLD);

std::string bestpfn, filetype;

mycatalog->start();

a.lookupBestPFN(fid, IFileCatalog::READ, IFileCatalog::SEQUENTIAL, bestpfn,
filetype);

mycatalog->commit(IFileCatalog::ONHOLD);

FClookup l;

mycatalog->setAction(l);

PFNContainer mypfs(mycatalog, 100);

mycatalog->start();

l.lookupPFNByQuery("",mypfs);

while(mypfs.hasNext()){

   std::cout<<mypfs.Next()<<std::endl;

}

mycatalog->commit();
```

```
mycatalog->disconnect();
```

2.6 Python interface of the component

The component provides a Python interface which allows the catalog operations to be called from a Python script instead of a compiled C++ application. The Python extension of the component consists of the following modules PyFileCatalog.py, PyAction.py, PyFCContainer.py, PyFileCatalogError.py and the C++ binding module of the backend implementation FileCatalog.so. To import these modules, $POOLProject/$arch/lib, $POOLProject/$arch/bin must be set in the $PYTHONPATH. To load backend catalog implementations, one should also set $SEAL_PLUGINS correctly and have all the external libraries the backend depends on set in $LD_LIBRARYPATH.

Following the common style of Python extension modules, this module is self-documenting. One can use `dir` and `help` functions to see the usage of the method. All the methods provided by the C++ API of IFileCatalog, IFCAction and IFCContainer class are available in Python with the same name. Besides, the python component has one extra method PyFClookup::lookupEntryByQuery() which retrieves the entire PFN-LFN-MetaData mapping by query.

The python component defines the following constants in the global scope which behave as enum type in C++ to be used as function arguments:

REFRESH , ONHOLD (argument of the commit() method)

NO_DELETE, DELETE_REDUNDANT

(argument of the updateMetaDataSpec() method)

LFN, PFN, META, GUID (arguments of the PyFCContainer() constructor)

SEQUENTIAL, RANDOM, PRANDOM

(access-mode arguments of the lookupBestPFN() method)

READ, WRITE, UPDATE

(open-mode arguments of the lookupBestPFN() method)

Due to the language difference, the Python methods support default arguments and keyword arguments: e.g. the following calls are also legal

```
pfns=PyContainer(a_catalog_instance,  PFN)    (default cache size=1000)

pfns=PyContainer(a_catalog_instance,  “cache size”=120, “container type”=PFN)
```

Examples of the python component can by found in the component unit test area:

/pool/PyFileCatalog/tests

2.7 Catalog schema migration

The main schema of the file catalog has been changed from POOL_1_3_x releases to POOL_1_4_x releases. In the later releases, the PFN attributes “job_status” and “file_status” are removed. For the old catalogs produced by POOL_1_3_x to be readable by POOL_1_4_x software, user has to update the schema of the old catalog using the migration tools included in the POOL_1_4_x releases.

For the XML catalog, use the command:

XMLmigrate_POOL1toPOOL1.4 –u oldcatalog.xml –d newcatalog.xml

Note: here the protocol name “file:” should not be included in the catalog name.

For the MySQL catalog, use the script in src/Scripts/FileCatalog/mysqlcatalog_migrate_POOL1toPOOL1.4.sql

mysql –u username –h hostname dbname< mysqlcatalog_migrate_POOL1toPOOL1.4.sql

For the EDG catalog, use the command:

EDGmigrate_POOL1toPOOL1.4 rlstest.cern.ch:7777

Note: updating of the EDG catalog should be performed only by the administrator of the rls service of the VO. Single user should not attempt to update the EDG catalog.

3 Detailed C++ API of the component

FileCatalog component depends on the following POOL components

/pool/POOLCore

/pool/AttributeList

All classes are defined in the *pool* namespace.

3.1 Public interfaces

Interfaces

IFileCatalog Class

connect	**This method establishes the connection to the catalog backend. Exception is thrown in case of problems.**
	Syntax: void connect()
disconnect	**This method disconnect from the catalog backend. Exception is thrown in case of problems.**
	Syntax: void disconnect()
start	**This method starts the catalog transaction. Exception is thrown in case of problems**
	Syntax: void start()
commit	**This method commits the catalog transaction. Exception is thrown if the operation cannot be committed.** **CommitMode can be IFileCatalog::REFRESH or IFileCatalog::ONHOLD. The default value is REFRESH.**
	Syntax: void commit(const CommitMode)
rollback	**This method rolls back the catalog transaction. Exception is thrown if the opertation cannot be rolled back.**
	Syntax: void rollback()
addReadCatalog	**This method adds the read-only catalog specified by the contact string to the composite catalog**
	Syntax: void addReadCatalog(const std::string& contact)
addReadCatalog	**This method adds the read-only catalog instance to the composite catalog**
	Syntax: void addReadCatalog(FCLeaf* r)
setWriteCatalog	**This method sets the read/writable catalog specified by the contact string**
	Syntax: void setWriteCatalog(const std::string& contact)

setWriteCatalog	**This method sets the read/writable catalog instance in the composite catalog**
	Syntax: void setWriteCatalog(FCLeaf* w)
getWriteCatalog	**This method returns the read/writable catalog**
	Syntax: IFileCatalog* getWriteCatalog()
getReadCatalog	**This method returns the instance of the read-only catalog specified by the idex**
	Syntax: IFileCatalog* getReadCatalog(const unsigned long& idx)
nReadCatalogs	**This method returns the number of read-only catalogs**
	Syntax: const size_t nReadCatalogs() const
setAction	**This method associates an action with the catalog**
	Syntax: void setAction(IFCAction&)
importCatalog	**This method appends a fragment of the given source catalog to the current catalog. The fragment is selected by query on the source catalog metadata. If the query string is empty, entire source catalog is appended to the current catalog. One can specify the default cache size for this operation. The default value is 1000.**
	Syntax: void importCatalog(IFileCatalog* fc, const std::string& query, unsigned int cachesze=FCDEFAULT_CACHE_SIZE) const
isReadOnly	**This method tells if the catalog is read-only**
	Syntax: bool isReadOnly() const

IFCAction class

isWritableEntry	**This method tells if the given guid is from the read/writable catalog**
	Syntax: bool isWritableEntry(IFileCatalog::FileID& fid)
registerPFN	**This method registers a file with the given PFN and file type; returns the corresponding FileID from the argument list. Exception is thrown if PFN is already registered. This operation is performed on the read/writable catalog.**
	Syntax: void registerPFN(const std::string& pfn, const std::string& filetype, IFileCatalog::FileID& fid)
lookupFileByPFN	**This method returns the FileID and file type with given PFN**
	Syntax: void lookupFileByPFN(const std::string& pfn, FileID& fid, std::string& filetype)
lookupFileByLFN	**This method returns the FileID with given LFN**
	Syntax: void lookupFileByLFN(const std::string& lfn, FileID& fid)
getMetaDataSpec	**This methods returns the metadata schema definition of the catalog**
	Syntax: void getMetaDataSpec(MetaDataEntry& spec)

lookupBestPFN	**This method returns a PFN associated with the given FileID. The first PFN found is returned. The file type is also returned. FileOpenMode and FileAccessPattern are passed as a hint to the Grid components for file transfer.**
	Syntax: void lookupBestPFN(const FileID& fid, const FileOpenMode& omode, const FileAccessPattern& amode, std::string& pf, std::string& filetype)
visitFCLeaf	**This methods allows the leaf catalog to register itself**
	Syntax: void visitFCLeaf(IFileCatalog* cat)
visitFCComposite	**This methods allows the composite catalog to register itself**
	Syntax: void visitFCComposite(IFileCatalog* cat)

FCregister class

registerLFN	**This method registers a LFN associated with the given PFN.**
	Syntax: void registerLFN(const std::string& pfn, const std::string& lfn)
addReplicaPFN	**This method adds the PFN of a replica to its master copy PFN.**
	Syntax: void addReplicaPFN(const std::string& pfn, const std::string& rpf)
renamePFN	**This method replaces a given PFN with a new PFN.**
	Syntax: void renamePFN(const std::string& pfn, const std::string& newpfn)
registerMetaData	**This method inserts metadata values of a file with given FileID.**
	Syntax: void registerMetaData(const IfileCatalog::FileID& fid, MetaDataEntry& attrs)

FClookup class

lookupPFN	**This method returns PFNs associated with given FileID.**
	Syntax: void lookupPFN(const IFileCatalog::FileID& fid, PFNContainer& pfs)
lookupLFN	**This method returns LFNs associated with given FileID.**
	Syntax: void lookupLFN(const IFileCatalog::FileID& fid, LFNContainer& lfs)
lookupPFNByQuery	**This method returns PFNs satisfy the query.**
	Syntax: void lookupPFNByQuery(const std::string& query, PFNContainer& pfs)
lookupLFNByQuery	**This method returns LFNs satisfy the query.**
	Syntax: void lookupLFNByQuery(const std::string& query, LFNContainer& lfs)

lookupMetaDataByQuery	**This method returns meta data selected by the query.**
	Syntax: void lookupMetaDataByQuery(const std::string& query, MetaDataContainer& metas)
lookupPFNByLFN	**This method returns PFNs associated with given LFN**
	Syntax: void lookupPFNByLFN(const std::string& lfn, PFNContainer& pfs)
lookupLFNByPFN	**This method returns LFNs associated with given PFN**
	Syntax: void lookupLFNByPFN(const std::string& pfn, LFNContainer& lfs)
lookupFileByQuery	**This method returns all FileIDs selected by the query.**
	Syntax: void lookupFileByQuery(const std::string& query, GuidContainer& fids)

FCAdmin class

deleteLFN	**This method deletes the specified LFN.**
	Syntax: void deleteLFN(const std::string& lfn)
deletePFN	**This method deletes the specified PFN. If the PFN is the last one associated with a file, all associated LFN and metadata are deleted as well.**
	Syntax: void deletePFN(const std::string& pfn)
deleteMetaData	**This method deletes the metadata associated with the FileID.**
	Syntax: void deleteMetaData(const IFileCatalog::FileID& fid)
dropMetaDataSpec	**This method drops the metadata and its definition.**
	Syntax: void droptMetaDataSpec()
createMetaDataSpec	**This method creates the metadata definition of the catalog.**
	Syntax: void createMetaDataSpec(MetaDataEntry& spec)
updateMetaDataSpec	**This method updates metadata definition in the catalog or create one if catalog has no metadata defined. The default value of the FCMetaUpdateMode argument is NO_DELETE which only adds new attributes. DELETE_REDUNDANT mode will delete the old attributes which are absent from the new metadata definition.**
deleteEntry	**This method deletes the entire mapping associated with the given FileID.**
	Syntax: void deleteEntry(const IFileCatalog::FileID& fid)

Template<typename Item>IFCContainer,

PFNContainer, LFNContainer,MetaDataContainer, GuidContainer

IFCContainer	**The constructor creates an instance of the container bounded to a given catalog and initialised with given cachesize.**

	Syntax: IFCContainer(IFileCatalog* catalog, const size_t cachesize=1000)
reset	**This method resets the iterator state to its initial values.**
	Syntax: void reset()
hasNext	**This method tells if there is next entry in the container.**
	Syntax: bool hasNext()
Next	**This method retrieves the next item from the container.**
	Syntax: Item& Next()
max_size	**This method tells cache capacity of the container**
	Syntax: size_t max_size() const

3.2 Exceptions generated

All exceptions thrown from the File Catalog components are either seal::Exception or derived from it. User should catch seal::Exception

The explicit file catalog exceptions are:

FCduplicateLFNException	thrown when attempting to register a LFN which already exists in the catalog
FCduplicatePFNException	thrown when attempting to register a PFN which already registered in the catalog
FCnonexistentFileException	thrown when attempting to register the replica of a nonexistent physical file
FCbackendException	this exception propagates exceptions thrown by the backend libraries
FCillegalContactStringException	thrown in case of illegal contact string is used to load or connect to the catalog
FCTransactionException	thrown when attempting to do an transactional operation outside the transaction state or attempting to write into/update a read-only catalog. The transactional operations are file registrations, catalog lookups and catalog updates
FCconnectionException	thrown in case of failure of connection or disconnection
FCstringLimitException	thrown in case the string length exceeds the limit of the backend, i.e. 250 char.
FCduplicatemetadataspecException	thrown in case the metadata spec is already defined in the catalog when trying to create a metadata schema

FCduplicatemetadataException	thrown in case the guid-metadata mapping is already defined in the catalog when trying to insert a new one

Operations

4 Analysis of the component

4.1 Glossary

Catalog Contact String: The string to identify a file catalog**.**

The format compliant with **[4]:**
[prefix_][protocol]://[username]:[password]@[host]:[port]/[path]

or

[prefix_]file:path

The latter format is accepted by the XML catalog.

File Identifier (FileID): Globally unique identifier associated with a file. It may be generated by the UUID mechanism in the format of binary or a long string

Physical File Name (PFN): The name refers to the physical location of a file.

Alias : Different logical names assigned to the same file or a set of files sharing the same FileID.

Logical File Name (LFN): The same as alias.

Catalog fragment: A set of entries in the file catalog grouped by the user using certain criteria. For instance, all the files produced by a job, all the files contain data from a single run, etc.

Extract a catalog fragment: Select a subset of entries in a central catalog and put them in a local catalog. This operation should be atomic, i.e. selected entries are extracted in one transaction.

Publish a catalog fragment: Make a fragment of the file catalog available on the larger central catalog. The fragment of the catalog should be published atomically, i.e. one transaction publishes the entire fragment.

Register a file: Assign a FileID to a file and insert the PFN-FileID mapping in the catalog.

Replica: An exact copy of a file that is linked to the original file through some well-defined mechanism.

XML catalog: Catalog based on XML. It is human-readable and editable. This catalog or its fragment will eventually be published to larger catalogs.

Native MySQL catalog: Catalog based on MySQL database. It contains larger datasets than the XML catalog, all the files produced by a farm, for instance.

It is not Grid-aware.

RLS based catalog: Grid-aware file catalog built on the EDG-RLS component.

Replica Location Service (RLS): A Grid service, which maintains and provides access to information about physical location of copies of files.

4.2 Requirements

1. The file catalog contains the PFN-FileID and LFN-FileID mapping.

It supports FileID->PFN, PFN->FileID look-ups.

The FileID is globally unique and immutable.

2. A file can be registered without specifying anything for its LFN. LFNs can be assigned to files and registered in the catalog after the job is finished.

3. It should be possible to register and populate a file in one job and append to the same file later from a second job.

4. The catalog should be available both on and off the Grid. A job can be run locally independent of any catalogs on the Grid.

5. If the job crashes, the Grid catalog should be able to clean up the registration of the files that will be produced by this job and related jobs in the catalog. However, it's not the responsibility of the catalog to find and delete the partially produced file by a crashed job.

6. The catalog should support catalog browsing.

4.2.1 Implemented requirements

1. Unique FileID is implemented by the GUID algorithm.
2. LFN registration outside the job can be done using command-line tools and Python interface.
3. Cross catalog operations can be done using command-line tools and Python methods.
4. EDG catalog is on the Grid.
5. Not implemented by EDG.
6. FCBrowser is the POOL catalog browser.

4.3 Use cases reports

The first two use cases are typical interactions of the storage manager with the file catalog. They happen in a single process.

The other use cases are typical operation mode of a group of jobs.

ID	*1*	*Register a file*
Description		*This use case happens when the storage manager creates a new file. The PFN may or may not already be registered by user in the catalog before the job runs. The file catalog component generates the FileID, adds it to the catalog and returns FileID to the storage manager. In the case that PFN is pre-registered in the catalog, the FileID found in the catalog is returned.*

Flow of events	**Basic flow**	*T=t0: the storage manager passes the PFN to be registered in the catalog.* *T=t1: the file catalog component checks if the PFN is pre-registered* *T=t2: if the PFN is not pre-registered, the file catalog component generates a unique FileID.* *T=t3: the file catalog component inserts into the PFN-FileID pair in the file catalog.* *T=t4: the file catalog returns the FileID to the storage manager.*
	Alternative flow	*T=t2: if the PFN is pre-registered, the file catalog component returns the FileID to the storage manager.*
	Alternative flow	*List of actions*
Special requirements	In the case of registering files before job runs: the pre-registration is handled by the command-line tools.	
Pre conditions	It is the responsibility of the user to guarantee the uniqueness of the PFN.	
Post conditions		
Extension points		

ID	*2*	*Lookup a file*
Description	*This use case happens when the storage manager dereferences a smart pointer to open a file for reading or update.*	
Flow of events	**Basic flow**	*T=t0: The storage manager specifies a unique FileID.* *T=t1: the file catalog component looks up the PFN corresponding to the FileID in the catalog,* *T=t2: the file catalog component returns the PFN to the storage manager. In the current release, the first PFN found is returned.* *If PFN not found or the file cannot be accessed as indicated then throw exception.*
	Alternative flow	*List of actions*
	Alternative flow	*List of actions*
Special requirements	The file may be opened for update. In this case, we assume the file to be opened is the master copy and is still available for writing	
Pre conditions		
Post conditions		

Extension points	If the file is not local, it's the responsibility of the Grid to decide how to ship or replicate the file. We pass to the Grid software the FileAccessPattern indicated by the user as a hint. We pass to the Grid software also the FileOpenMode(read,write or update) indicated by the storage manager. The Grid security should decide if the user has the permission to do indicated operation on the file.

ID	*3*	*Run a write-only production job*	
Description		*In this scenario a large amount of new data are produced by the job and to be published to the collaboration. No input file is required by this job.*	
Flow of events		**Basic flow**	*T=t0: before the production starts, the production manager registers all the output file PFNs and corresponding jobIDs into the catalog.* *The job status is set to 0.* *This step is optional, PFNs can be registered also during the job.* *T=t1: the production starts and registered files are populated. When each job finishes successfully, the job status is set to 1.* *T=t2: the production ends. The production manager clean up the catalog entries where job status is 0 which means the job didn't end successfully.* *T=t3: The selected catalog fragment is published to the central catalog.*
		Alternative flow	*Depends on the setup of the production, the catalog can be Grid-based catalog if the production runs in a Grid-aware environment; a MySQL catalog if the production runs in an isolated computer farm; or a XML catalog if the production runs in an isolated environment.* *XML catalog fragment can be published to MySQL and Grid-based catalog; MySQL catalog fragment can be published to the Grid-based catalog.*
		Alternative flow	*List of actions*
Special requirements		We assume that the Grid knows how to invalidate catalog entries produced by other jobs related to a crashed job.	
Pre conditions			
Post conditions			
Extension points			

ID	4	*Run a read-write job in an isolated environment*
Description		*The job runs in an isolated environment. The job needs input files and it creates output files as well. The user decides whether or not to publish the output data later.*
Flow of events	**Basic flow**	*T=t0: User extracts the catalog fragment needed by the job from a central catalog into a local XML catalog when the network is available;* *T=t1: user runs the job disconnected from the network. During the job run, the local input XML catalog is used for reading; the output files are registered into another local XML catalog.* *T=t2: n jobs run. n output XML catalogs are produced.* *T=t3: User selects and publishes the local catalog fragments to the central catalog.*
	Alternative flow	
	Alternative flow	*List of actions*
Special requirements		
Pre conditions		*Resources needed by the job for reading are available.*
Post conditions		
Extension points		

ID	5	*Browsing of the catalog*
Description		*User needs to have available in memory a part of the catalog*
Flow of events	**Basic flow**	*T=t0: User decides which catalog start browsing* *T=t1: The browsing starts returning all LFN-FileID pairs and/or all PFN-FileID pairs.* *T=t2: Navigation from LFN to PFNs via common FileID can be performed on shell or scripting. A new file catalog may be created by command lines with part of the extracted information.*
	Alternative flow	
	Alternative flow	*List of actions*
Special requirements		

Pre conditions	
Post conditions	
Extension points	

5 Technology surveys and external libraries of the component

Name	*MySQL*	Vendor	*MYSQL AB*
Version	*4.0.13*	License	*GPL*
Description	*Database backend of native MySQl catalog and RLS based catalog*		
Component related comments	*InnoDB table type is used because it supports transaction.*		

Name	*MySQL++*	Vendor	*MYSQL AB*
Version	*1.7.9*	License	*GPL*
Description	*C++ API of MySQL client*		

Name	*Xerces-C++*	Vendor	*Apache.org*
Version	*2.3.0*	License	
Description	*Validating the XML-parser for C++.*		

Name	*edg-rls-client*	Vendor	*EDG*
Version	*2.2.1*	License	*EDG*
Description	*C++ API of EDG-RLS service*		

6 Component design

6.1 Main Strategy

Complete separation between interface and implementations.

Multiple language support: compiled C++ and scripting Python.

6.2 Schemas

6.2.1 MySQL

Table: t_lfn(

lfname varbinary(250) primary key,

guid varbinary(40),

INDEX idxt_lfn(guid)

) type=innodb

Table: t_pfn(

pfname varbinary(250) primary key,

guid varbinary(40) ,

filetype varbinary(250),

INDEX idxt_pfn(guid)

) type=innodb

Optionally, file metadata can be associated to the fileID. Schema describes the metadata is defined by user.

A SQL script for creating the MySQL catalog schema can be found in *pool/Scripts/FileCatalog/mysqlcatalog_schema_POOL1.4.sql*

6.2.2 XML DTD

<!ELEMENT POOLFILECATALOG (META*,File*)>\

<!ELEMENT META EMPTY>\

<!ELEMENT File (physical,logical,metadata*)>\

<!ATTLIST META name CDATA #REQUIRED>\

<!ATTLIST META type CDATA #REQUIRED>\

<!ELEMENT physical (pfn)+>\

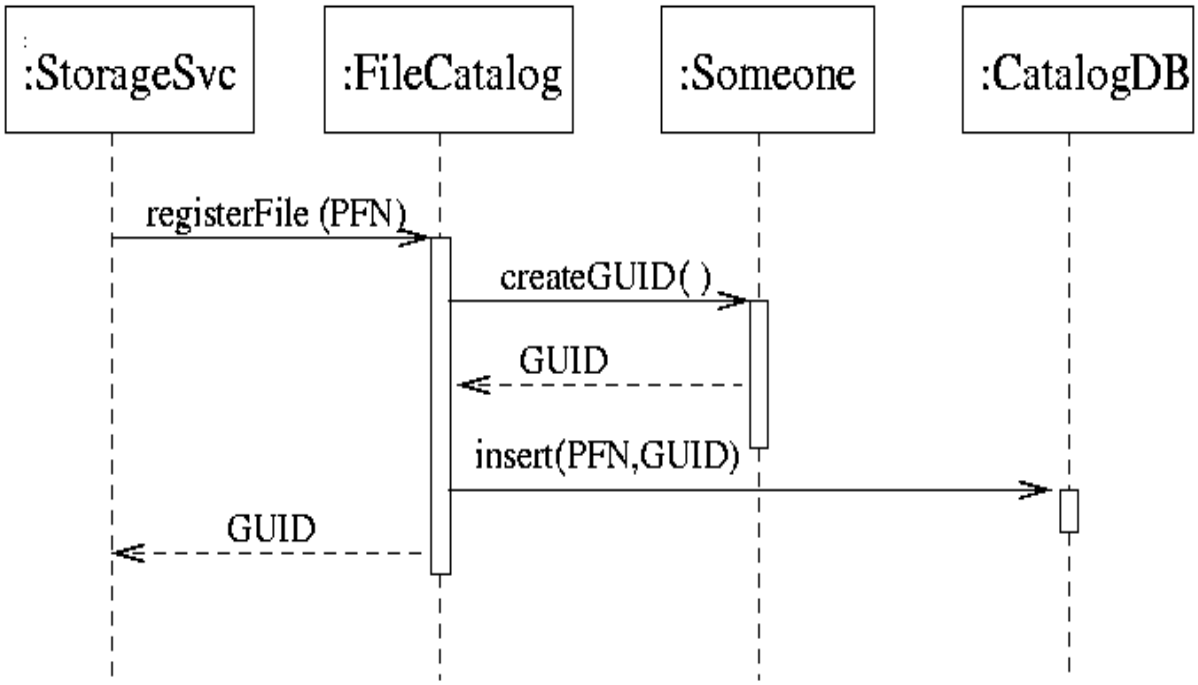
<!ELEMENT logical (lfn)*>\

<!ELEMENT metadata EMPTY>\

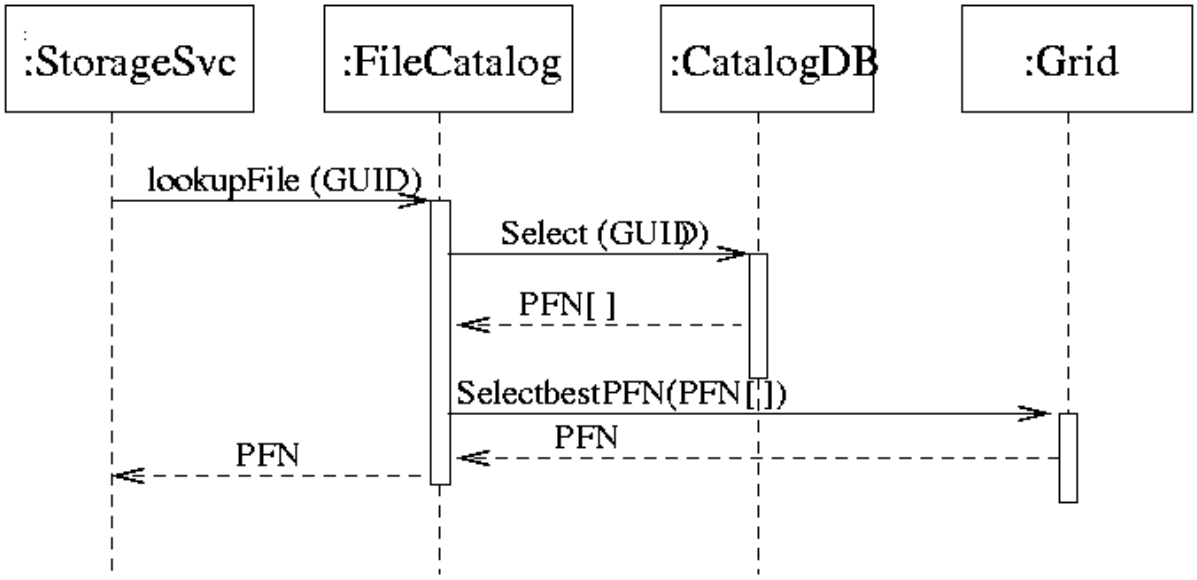
<!ELEMENT lfn EMPTY>\
<!ELEMENT pfn EMPTY>\
<!ATTLIST File ID ID #REQUIRED>\
<!ATTLIST pfn name ID #REQUIRED>\
<!ATTLIST pfn filetype CDATA #IMPLIED>\
<!ATTLIST lfn name ID #REQUIRED>\
<!ATTLIST metadata att_name CDATA #REQUIRED>\
<!ATTLIST metadata att_value CDATA #REQUIRED>\
";

6.3 UML diagrams

1. Sequence diagram for use case: register file.
:CatalogDB is the backend database of the catalog.



2. Sequence diagram for use case: lookup file.
For non Grid-aware catalog, the PFN[0] is returned.



6.4 Patterns used

List the patterns that have been used in the design.

6.5 Exceptions used

Explain the exceptions structure inside the component, identifying internal and external ones. Cut and paste the UML diagram for exceptions.

6.6 Additional comments, restriction and known problems

6.6.1 Example runtime setup script for SCRAM users.

The runtime environment variable can be set from a XML-style file:
eval `scram runtime –csh pool.env` (from csh)

eval `scram runtime –sh pool.env` (from sh)

An example pool.env is attached below:

```
<Ignore>

<Runtime name=POOL_CATALOG
value="mysqlcatalog_mysql://@lxshare070d.cern.ch:3306/testFCdb">

the MySQL db contact string</Runtime>

</Ignore>

<Runtime name=POOL_CATALOG value="xmlcatalog_file:FileCatalog.xml">

the XML db contact string </Runtime>

<Ignore>
```

```
<Runtime name=POOL_CATALOG
value="edgcatalog_http://rlstest.cern.ch:7777/test/v2.2/edg-local-replica-
catalog/services/edg-local-replica-catalog”>

the EDG RLS contact string </Runtime>

</Ignore>

<Runtime name=POOL_OUTMSG_LEVEL value=”7”>output message threshold</Runtime>
```

<Ignore> tags starts and ends the comment lines.

7 Implementation

7.1 MySQL

MySQLFileCatalog class implements the abstract interface of the component FCImpl. It contains an opaque pointer to class MySQLImpl which hides the private data members and functions used by the MySQL catalog and implements calls to the MySQL++ library.

The component throws FCException for logical errors and rethrows other MySQL++ exceptions as FCbackendException .

How to create a MySQL catalog?

The MySQL catalog can be created using the MySQL command-line tools and the script tbcreate_POOL_1.4.sql provided by POOL, e.g. :

mysql –u username mydb < tbcreate_POOL_1.4.sql

The script tbcreate_POOL_1.4.sql can be found in the bin directory of the POOL release. If your environment is set correctly, it should be already set in your PATH.

7.2 XML

The classe XMLFileCatalog is the concrete implementation of the interface FCImpl, handling file catalogs written as XML files. The DTD is held in memory and realizes the many PFNs to unique FileID to many LFNs association as described below. The catalog is parsed by using the Xerces-C libraries that provides the DOM interface with the functionalities to navigate through the trees. The catalog contact string is specified via the environment variable POOL_CATALOG. It mainly expects local files, though to browse the catalog content the http protocol is supported within the XercesC library. Other protocols might be implemented in a consistent way. In case the specified file catalog does not exist a new one is created from scratch.

Logging, error and exception handling are also ensured. Error and exception handlers are used by the component by the mechanism provided by the XercesC for the internal consistency of the XML file with respect to the DTD, and by the one defined in component interface.

The class XMLFileCatalog uses the class PoolXMLFileCatalog as the layer that provides the explicit functionalities to handle the XML specific catalog. PoolXMLFileCatalog encapsulates all the calls to the Xerces-C library. The consistency of the XML file catalog with respect to the DTD is ensured via the DOMError class that provides exceptions that are propagated up to the used code. The XercesDOMParser class provides the writing of the catalog as a DOM tree in the file specified by the contact string.

The component throws FCException for logical errors and rethrows Xerces-C exceptions as FCbackendException .

Warning: It is not recommended to strip off the XML header of the catalog for schema evolution reason.

7.3 EDG

The EDGCatalog component implements the abstract user interface IFileCatalog by encapsulating methods and functions of the edg-rlc [2] and edg-rmc [3] c++ client libraries.

The component throws FCException for logical errors and rethrows edg-rls-client libraries exceptions as FCbackendException .

8 To do

TO DO
Migrate MySQL catalog out of MySQL++

9 References

[1] http://grid-data-management.web.cern.ch/grid-data-management/docs/ReplicaManager/White-Paper.pdf

[2] http://proj-grid-data-build.web.cern.ch/proj-grid-data-build/edg-rls-server/user-guide/edg-rls-userguide.pdf

[3] http://proj-grid-data-build.web.cern.ch/proj-grid-data-build/edg-metadata-catalog/user-guide/edg-rmc-userguide.pdf

[4] http://www.ietf.org/rfc/rfc2396.txt