DataGRID

DEVELOPER GUIDE FOR EDG REPLICA MANAGER 1.8.1

WP2 EDG REPLICA MANAGER

Document identifier:	**DataGrid-02-ERM-DEV-GUIDE**
EDMS id:	
Date:	February 4, 2005
Work package:	**WP2: Data Management**
Partner(s):	**CERN, PPARC, HIP, INFN, ITC, KDC**
Lead Partner:	**CERN**
Document status:	**FINAL**
Author(s):	WP2
File:	**edg-replica-manager-devguide**

Abstract: This is the developer guide for the EDG Replica Manager. It explains how to use the EDG Replica Manager C++ and Java APIs.

CONTENTS

CONTENTS

Document Change Record

Issue	Date	Comment	Author
1_0	25/03/03	Initial version based on previous versions by Heinz and Kurt Stockinger.	Kurt Stockinger
1_1	09/10/03	New version for 2.1 release.	David Cameron

1. INTRODUCTION

1.1. OBJECTIVES OF THIS DOCUMENT

This document is the Java and C++ developer guide for the EDG Replica Manager (Reptor). The usage of the EDG Replica Manager is described in the user guide [A5].

1.2. APPLICATION AREA

WP2 Data Management.

1.3. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] D2.2 WP2 Architecture and Design Document.
[A2] D12.2 DataGrid Naming Conventions document, WP12 Project Management. http://edms.cern.ch/document/328838
[A3] D12.2 DataGrid Developers Guide, Quality Assurance Group. DataGrid-12-TED-358824-1-1, https://edms.cern.ch/document/358824/1.1
[A4] D12.4 DataGrid Architecture
[A5] D2.5 EDG Replica Manager User Guide
[A6] D2.5 EDG Replica Manager Install Guide

Reference documents

[R1] L. Guy, E. Laure, P. Kunszt, E. Laure, H. Stockinger, K. Stockinger, Replica Management in DataGrids. Presented at GGF5. http://edms.cern.ch/document/350430.
[R2] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, F. Zini. Design of a Replica Optimization Framework. Technical Report DataGrid-02-TED-021215, CERN, Geneva, Switzerland, December 2002.
[R3] Ann Chervenak, Ewa Deelman, Ian Foster, Wolfgang Hoschek, Adriana Iamnitchi, Carl Kesselman, Peter Kunszt, Matei Ripeanu, Heinz Stockinger, Kurt Stockinger, and Brian Tierney. Giggle: A Framework for ConstructingScalable Replica Location Services. In Proc. of the Int'l. IEEE Supercomputing Conference (SC 2002), Baltimore, USA, November 2002.

2. JAVA APPLICATION PROGRAM INTERFACE

2.1. MANAGEMENT COMMANDS

The management commands are:

- `copyAndRegisterFile`
- `replicateFile`
- `deleteFile`

2.1.1. `copyAndRegisterFile`

```
public java.net.URI copyAndRegisterFile(java.net.URI source,
                                        java.net.URI dest,
                                        java.net.URI logicalName,
                                        java.lang.String protocol,
                                        int nstreams)
```

This method performs the task of copying a file into grid-aware storage and registering the copy in the Replica Catalog as an atomic operation. This is the preferred method to 'bring files into the grid'.

The same could be done by issuing copyFile and registerFile in sequence, but since copyFile may not have an SE or SURL as its destination, the user would need to specify a valid transport file name (using gridftp for example) to a location that happens to be the proper location for the file and then register it using registerFile. That is much more error prone and cumbersome than this method. As a rule, copyFile should be used to copy files around to non-grid aware storage, or to simply copy files out of the grid (i.e. the source may be grid-aware but not the destination).

registerFile should be used to register files that are already in the grid-aware store before it was made grid-aware or to register files that appear there through other means (like new data files). If the files that need to be put into the grid are not yet in their grid-aware location, copyAndRegisterFile is the most robust method to use.

PARAMETERS

`source` - the file to be registered. It must be accessible through this name using the specified protocol in the URI. The source may be:

- A local file.
- A transport URI with a valid protocol schema.

`dest` - (optional) The physical destination file. dest may be specified in three different ways:

- As a fully qualified storage file name (SURL). If this file already exists, the operation fails.
- Only the host of the SE where the file should be stored, i.e. the URI was constructed just by giving it a host string, which will be stored in the URI path. The storage location will be determined automatically depending on the user's VO and a suitable file name will be chosen.
- If no destination is specified at all (i.e. the argument is null, the 'closest' SE is located and the file is stored as in the previous case.

`logicalName` - (optional) The logical name to be used in the catalog to find the file again. If this parameter is null, only the GUID is used.

`protocol` - (optional) The protocol to be used (can be null in which case the default protocol for the given SE is used)

`nstreams` - (optional) The number of parallel streams to be used for the copy. If zero or less than zero, the default is used.

RETURN VALUE

The GUID of the registered file.

Exceptions

`java.io.FileNotFoundException` - if source does not exist

`InvalidFileNameException` - if source or destination is not valid

`FileHandleException` - if there is a problem with source or destination

`FileCopierException` - if the copy operation fails

`ReptorException` - in case of a runtime problem, like bad configuration.

Example

```
copyAndRegisterFile(gsiftp://testbed008.cern.ch/tmp/higgs,
                    surl://testbed007.cnaf.infn.it/shared/wpsix/higgs000,
                    lfn:myHiggs000,
                    ``gridFTP'', 8);
```

2.1.2. `replicateFile`

```
public java.net.URI replicateFile(java.net.URI source,
                                  java.net.URI dest,
                                  java.lang.String protocol,
                                  int nstreams)
```

This method performs the task of replicating a file between grid-aware stores and registering the replica in the Replica Catalog as an atomic operation. This is the preferred method to ‘copy files around in the grid’. The difference to copyAndRegisterFile is that this operation only allows for GUID, LFN or SURL as the source file wheras copyAndRegisterFile explicitly forbids that. No new GUID is generated for the replica (hence the term).

Parameters

`source` - the file to be replicated. The source may be:

- A GUID. The SE will be chosen automatically.
- An LFN. The SE will be chosen automatically.
- An SURL. The given SE will be used as the source for this file.

`dest` - The physical destination. dest may be specified in three different ways:

- As a fully qualified storage file name (SURL). If this file already exists, the operation fails.
- Only the host of the SE where the file should be stored, i.e. the URI was constructed just by giving it a host string, which will be stored in the URI path. The storage location will be determined automatically depending on the user’s VO and a suitable file name will be chosen.
- If no destination is specified at all (i.e. the argument is null, the ‘closest’ SE is located and the file is replicated to it as in the previous case (if it’s not already available there)).

`protocol` - (optional) The protocol to be used (can be null in which case the default protocol for the given destination SE is used)

`nstreams` - (optional) The number of parallel streams to be used for the copy. If zero or less than zero, the default is used.

RETURN VALUE

The SURL of the new replica.

EXCEPTIONS

`java.io.FileNotFoundException` - if source does not exist

`InvalidFileNameException` - if source or destination is not valid

`FileHandleException` - if there is a problem with source or destination

`FileCopierException` - if the copy operation fails

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
replicateFile(lfn:myHiggs000,
              surl://se01.nikhef.nl/flatfiles/wpsix/higgs1,
              "gsiftp", 8);
```

2.1.3. `deleteFile`

```
public void deleteFile(java.net.URI file,
                       java.lang.String seHost,
                       boolean all)
```

Delete a file from storage and unregister it. It may be a single replica or all instances.

PARAMETERS

`file` - LFN, GUID or SURL of the file to be deleted.

`seHost` - if LFN or GUID is given, this indicates from which SE the replica should be removed. It needn't be given for an SURL.

`all` - boolean flag indicating whether all instances should be removed (only if file is a GUID).

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
deleteFile(lfn:myHiggs000, "se01.nikhef.nl", false);
```

2.2. CATALOG COMMANDS

The catalog commands are:

- `registerFile`

- `registerGUID`
- `unregisterFile`
- `listReplicas`
- `listGUID`
- `addAlias`

2.2.1. `registerFile`

```
public java.net.URI registerFile(java.net.URI file,
                                 java.net.URI logicalName)
```

Register a file in the Replica Catalog that is already stored on a Storage Element. It takes two arguments: source file and logical name. The source file needs to be a qualified storage file name URI, following the rule srm://storage.element.host/path/file.name The logical name is optional, it may declare a logical identifier that can be used later to look up any instance of the file. The method returns the GUID, the Grid Unique IDentifier of the file. The detailed semantics are:

- If the source file is not registered yet:
  - no logical name is given: a new GUID is generated, the file is registered and the GUID is returned to the caller.
  - a logical name is given: a new GUID will be allocated and returned, and the logical name will also be registered and can be used to retrieve the file later.
- If the file is already registered:
  - no logical name is given: the already existing GUID is returned.
  - a logical name is given that does not exist yet: it is added as an alias in the Replica Catalog so that the file can be looked up later using that name as well.
  - a logical name is given which already exists: it is checked whether the logical name really corresponds to the file. If not, an exception is thrown, otherwise the existing GUID is returned.

This method does not include any data movement (i.e. the storage file is not copied) and assumes that the storage file is already located at a known Storage Element.

PARAMETERS

`file` - the file. It must be a valid Storage File Name, i.e. the host part of the given URI needs to be a recognized Storage Element.

`logicalName` - the logical name to be used in the catalog to find the file again. If this parameter is null, an automatic name is generated and used.

RETURN VALUE

The GUID of the source file that this entry was registered with.

EXCEPTIONS

`InvalidFileNameException` - if file is not valid or the SE is not found

`java.io.FileNotFoundException` - if file does not exist

`ReptorException` - in case of a runtime problem, like bad configuration

EXAMPLE

```
registerFile(``gsiftp://testbed008.cern.ch/tmp/higgs'', null);
```

2.2.2. `registerGUID`

```
public java.net.URI registerGUID(java.net.URI file,
                                 java.net.URI guid)
```

Register a file in the Replica Catalog with a known GUID. It takes two arguments: source file and GUID. The source file needs to be a qualified storage file name URI. The GUID is the GUID that this file should be registered with for the case if the file was copied by other means. (TruckFTP use case). While there is only one centralized catalog, there is no need for this method, but if the catalogs are not centralized, this method can be used to register a file with a known GUID. Be careful, if the GUID corresponds to a wrong file in the remote (probably unattainable) catalogs then once the connection is established, the distributed catalog will inconsistent data.

PARAMETERS

`file` - the file. It must be a valid Storage File Name, i.e. the host part of the given URI needs to be a recognized Storage Element.

`guid` - the GUID to be used in the catalog to find the file again.

RETURN VALUE

The GUID, just like registerFile.

EXCEPTIONS

`InvalidFileNameException` - if file is not valid or the SE is not found

`java.io.FileNotFoundException` - if file does not exist

`ReptorException` - in case of a runtime problem, like bad configuration

EXAMPLE

```
registerGUID(``gsiftp://testbed008.cern.ch/tmp/higgs'',
                ``a6289c7c-4966-11d7-bc63-d91230733e2d'');
```

2.2.3. `unregisterFile`

```
public void unregisterFile(java.net.URI file,
                           java.net.URI guid)
```

Unregister a file from the Replica Catalog. There are two arguments: the actual fully qualified storage file name URI and the GUID under which it was registered. If the GUID is given, also all logical names are unregistered and the file cannot be found using the replica manager anymore, so this method should be used cautiously. If the file should be removed altogether, use deleteFile instead. This method does not include any data deletion (i.e. the storage file is not removed).

PARAMETERS

`file` - the storage file to unregister. It must be a valid Storage File Name, i.e. the host part of the given URI needs to be a recognized Storage Element.

`guid` - the GUID in the catalog.

EXCEPTIONS

`InvalidFileNameException` - if file is not valid or the SE is not found

`java.io.FileNotFoundException` - if file does not exist

`ReptorException` - in case of a runtime problem, like bad configuration

EXAMPLE

```
unregisterFile(``gsiftp://testbed008.cern.ch/tmp/higgs'',
               ``guid:272f86f8-8b66-11d7-94df-dd776a8aa1c6'');
```

2.2.4. `listReplicas`

```
public java.util.List listReplicas(java.net.URI file)
```

Returns the list of all known replicas of the file name that may be an LFN, GUID or SURL.

PARAMETERS

`file` - LFN, GUID or SURL of the file to get the replicas of.

RETURN VALUE

List of SURL URIs.

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
listReplicas(``lfn:myHiggs000'');
```

2.2.5. `listGUID`

```
public java.net.URI listGUID(java.net.URI file)
```

Returns the GUID of the given LFN or SURL.

PARAMETERS

`file` - LFN or SURL of the file to get the GUID of.

RETURN VALUE

GUID of the specified file.

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
listGUID(``gsiftp://testbed008.cern.ch/tmp/higgs'');
```

2.2.6. `addAlias`

```
public void addAlias(java.net.URI guid,
                     java.net.URI lfn)
```

Add an alias LFN to a known GUID.

PARAMETERS

`guid` - GUID to add new alias for
`lfn` - the new LFN

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
addAlias(``a6289c7c-4966-11d7-bc63-d91230733e2d'', ``lfn:myHiggs'');
```

2.3. OPTIMIZATION COMMANDS

The optimization commands are:

- `listBestFile`
- `getBestfile`
- `getAccessCost`

2.3.1. `listBestFile`

```
public java.net.URI listBestFile(java.net.URI logicalName,
                                 java.lang.String seHost,
                                 java.lang.String turlProtocol)
```

Return the 'best' replica for a given logical file identifier. Based on network and storage metrics the replica that is cheapest to access from a given Storage Element is returned. If the file is already on that storage, of course that is the one to be returned. listBestFile is identical to getBestFile except that it will not trigger replication to the specified destination SE if the file is not there yet.

PARAMETERS

`logicalName` - logical file name or GUID of the file to get the cheapest replica of.

`seHost` - the reference Storage Element. If this is null, the local SE will be taken.

`turlProtocol` - the protocol that the TURL will be returned as.

RETURN VALUE

URI of the SURL that is the cheapest to access from the seHost.

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
listBestFile(``lfn:myHiggs'', ``pcrd24.cern.ch'');
```

2.3.2. `getBestFile`

```
public java.net.URI getBestFile(java.net.URI logicalName,
                                java.lang.String seHost,
                                java.lang.String protocol,
                                java.lang.String turlProtocol,
                                int nstreams)
```

Return the storage file name (SURL) of the best file in terms of network latencies. getBestFile will always trigger replication if the best replica is not on a local SE.

PARAMETERS

`logicalName` - the LFN or GUID of the file to find the best replica of.

`seHost` - destination StorageElement.

`protocol` - Protocol to be used to copy the file.

`turlProtocol` - The protocol for the TURL that will be returned. If not specified, the SURL will be returned.

`nstreams` - Number of streams.

RETURN VALUE

SURL (or TURL) of best file.

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
getBestFile(``lfn:myHiggs'', ``pcrd24.cern.ch'', "gsiftp", null, 8);
```

2.3.3. `getAccessCost`

```
public org.edg.data.ros.AccessCost[] getAccessCost(java.net.URI[] logicalName,
                                                   java.lang.String[] ceIDs,
                                                   java.lang.String[] protocol)
```

Calculates the expected cost of accessing all the files specified by logicalName from each Computing Element host specified by ceHosts.

PARAMETERS

`logicalName` - the array of LFNs or GUIDs.

`ceHosts` - the array of ComputingElement hosts.

`protocol` - Transport protocol to be used.

RETURN VALUE

The AccessCost array in the same order as ceHosts array.

EXCEPTIONS

`ReptorException` - in case of a runtime problem, like bad configuration.

EXAMPLE

```
URI lfns[] = new URI[1];
lfns[0] = new URI(lfn.toString());

String hosts[] = new String[1];
hosts[0] = new String("lxshare0313.cern.ch");

String protocols[] = new String[1];
protocols[0] = new String("gsiftp");

getAccessCost(lfns, CEIDs, protocols);
```

2.4. FILE TRANSFER COMMANDS

The file transfer commands are:

- `copyfile`

2.4.1. COPYFILE

```
public void copyFile(java.net.URI     source,
                     java.net.URI     dest,
                     java.lang.String protocol,
                     int              nstreams,
                     boolean          force)
```

Copies a physical file from source to destination using the specified transport mechanism. This action does not involve any updates to the replica catalogs as the destination cannot be a grid-aware store, in order to avoid catalog corruption. copyAndRegisterFile needs to be used for that purpose, i.e. to bring a file into the grid. The destination does not accept GUIDs, LFNs or SURLs. As said, for those cases copyAndRegisterFile or replicateFile needs to be used.

copyFile provides all the capability of globus-url-copy and more, accepting also GUIDs, LFNs and SURLs as the source file. Since the destination is not on grid-storage, the destination file will not be registered in the catalog.

Example: To get a local copy of a grid file, the source file can be specified as a GUID or LFN and the destination file as a local file. The result will be that the file will be copied ‘out of the grid’ to the local file. Local file URIs need to have the ‘file’ scheme.

PARAMETERS

`source` - the source file. source may be one of the following:

- A GUID - the ‘best’ SE will be located to find the cheapest replica to copy from
- An LFN - the ‘best’ replica will be located as for the GUID
- A valid SURL URI, having the SE as its host name
- A valid transport URI, with a real protocol as its schema. Currently http, https, ftp and gsiftp are supported.

- A local file (specified with the 'file' schema).

`dest` - the physical destination file. Must be one of the following:

- A local file
- A transport URI with a valid protocol. The same protocols are supported as for the source.

`protocol` - the protocol to be used. This parameter is only considered if the source is specified using an SURL, LFN or GUID. If it's not set (i.e. null), the default protocol is used as specified in the configuration file.

`nstreams` - the number of streams to be used (if the protocol supports multiple streams, otherwise this parameter has no effect). If it's set ≤ 0 the default number of streams is used as specified in the configuration file.

`force` - overwrite the destination if it is already there.

EXCEPTIONS

`InvalidFileNameException` - if the source or destination file are invalid

`FileHandleException` - if there is a problem with source or destination

`FileCopierException` - if the copy operation fails

`InvalidSETypeException`

`ConfigurationException`

`ReptorException`

EXAMPLE

```
copyFile(tmp.toURI(), file, null, -1);

copyFile(gsiftp://testbed008.cern.ch/tmp/higgs,
         surl://testbed007.cnaf.infn.it/shared/wpsix/higgs000,
         ``gridFTP'', 8);
```

2.5. EXAMPLE PROGRAM

The Java program below shows example uses of the 4 methods *copyAndRegisterFile*, *replicateFile*, *listReplicas*, and *getBestFile*. After setting up the configuration and creating a ReplicaManager instance, a file stored on the local file system is copied to Grid storage and registered in the catalogs with a LFN (copyAndRegisterFile). The file is then replicated to Grid storage on another site (replicateFile) and the RM is asked for a list of the replicas of the LFN now available (listReplicas).

Then the RM is asked for a copy of the file to be used by a locally running job (getBestFile). If the file is not available on the local SE, a replica will be created there.

```
/**
 * EDGReplicaManagerExample.java
 *
 * Copyright (c) 2002-2003 CERN, on behalf of the EU DataGrid.
 * For license conditions see LICENSE file or
 * http://www.edg.org/license.html
 *
 */

import java.net.URI;
import java.net.URISyntaxException;

import java.util.Iterator;
import java.util.Set;

import org.edg.data.reptor.ReplicaManager;
import org.edg.data.reptor.ReplicaManagerImpl;
import org.edg.data.reptor.ReplicaManagerException;
import org.edg.data.reptor.Configuration;

public class EDGReplicaManagerExample {


    public EDGReplicaManagerExample() {
    }

    /**
     * Set up the configuration and the ReplicaManager instance.
     */
    public void setUp() throws Exception {
        try {
            m_config = new Configuration("/opt/edg/etc/edg-replica-manager \
                               /edg-replica-manager.conf", "wpsix");
            m_replicaManager = new ReplicaManagerImpl(m_config);
        } catch(Exception e) {
            System.out.println("Could not create ReplicaManagerImpl object "
                                    + e.getMessage());
    throw new Exception(e);
        }
    }

    /**
     * Here we copy and register a file to the grid, replicate it
     * to another site, list the replicas of the file, and then
     * call getBestFile to obtain a local copy of the file.
     */
    public void startReplication() throws Exception {

        // prepare source (local file), first SE destination and LFN
        URI source = new URI("file:/tmp/higgs0123");
        URI destination =
               new URI("srm://ced-rc0.datagrid.cnr.it/flatfiles/SE00/tutor/higgs0123");
        URI lfn = new URI("lfn:myHiggs0123");

        // copy the file into the Grid
        try {
            m_replicaManager.copyAndRegisterFile(source, destination,
                                                 lfn, null, 0);
        } catch(ReptorException e1) {
            System.out.println("copyAndRegister failure " + e1.getMessage());
        }
        // replicate to a second SE and let the RM decide
        // about the distination dir
        destination = new URI (null, "hepbf2.ph.qmul.ac.uk", null, null);
        try {
            m_replicaManager.replicateFile(lfn, destination, "gsiftp", 1);
        } catch(ReptorException e2) {
            System.out.println("replicateFile failure " + e2.getMessage());
        }

        // list the replicas - there should be two now
        System.out.println("List replicas...");
        Set replicas = null;
        try {
            replicas = m_replicaManager.listReplicas(lfn);

            Iterator it = replicas.iterator();
            while(it.hasNext()) {
                URI replica = (URI) it.next();
                System.out.println(replica.toString());
            }
        } catch(ReptorException e3) {
            System.out.println("listReplicas failure " + e3.getMessage());
        }

        // Now we want a copy of the file locally to use for a job.
        // We call getBestFile and it will replicate to the local SE
        // if the file is not already there.
        try {
            System.out.println("Calling getBestFile...");

            // if the destination is null, the local (default) SE is used
            URI localFile = m_replicaManager.getBestFile(lfn, null, "gsiftp",
                                                         null, 0);
        } catch(ReptorException e4) {
             System.out.println("getBestFile failure " + e4.getMessage());
        }
    }


    /*
     * Main Program
     */
    public static void main(String[] args) {

        System.out.println("EDG Replica Manager: example use of JavaAPI");

        EDGReplicaManagerExample example = new EDGReplicaManagerExample();
        try {
            example.setUp();
            example.startReplication();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /*
```

```
     * member variables
     */
    private static ReplicaManager m_replicaManager;
    private static Configuration m_config;

}
```

3. C++ Application Program Interface

In addition to the Java API, Reptor also supports three 3 C++ methods that are mainly used for the Resource Broker. The methods are:

- `listReplicas`
- `listBestFile`
- `getAccessCost`

For all the commands the following types are defined:

```
typedef std::vector<std::string>                      LFNList_t;
typedef std::vector<std::string>                      SURLList_t;
typedef std::vector<std::string>                      CEList_t;
typedef std::vector<std::string>                      SEList_t;
typedef std::vector<std::string>                      Protocols_t;
typedef std::vector<EdgReplicaOptimization::AccessCost> AccessCosts_t;
```

3.1. Commands

3.1.1. `listReplicas`

```
SURLList_t  listReplicas(std::string fileID)
```

Returns the list of all known replicas of the file name that may be an LFN, GUID or SURL.

Input Parameters

`fileID` - LFN, GUID or SURL of the file to get the replicas of.

Return Value

`SURLList_t` - List of SURL URIs.

Example

```
SURLList_t mySurls;

mySurls = listReplicas("lfn:myHiggs");
```

3.1.2. `listBestFile`

```
std::string listBestFile(std::string file, std::string seHost)
```

INPUT PARAMETERS

`file` - logical file name or GUID of the file to get the cheapest replica of.

`seHost` - the reference Storage Element. If this is null, the local SE will be taken.

RETURN VALUE

URI of the SURL that is the cheapest to access from the seHost.

EXAMPLE

```
std::string bestFile = listBestFile("lfn:myHiggs", "se01.nikhef.nl");
```

3.1.3. `getAccessCost`

```
AccessCosts_t getAccessCost(LFNList_t LFNList,
                            CEList_t CEList,
                            Protocols_t protocols)
```

Calculates the expected cost of accessing all the files specified by LFNList from each Computing Element host specified by CEList.

INPUT PARAMETERS

`LFNList` - the array of LFNs or GUIDs.

`CEList` - the array of ComputingElement hosts.

`protocols` - array of transport protocols to be used.

RETURN VALUE

`AccessCosts_t` - the AccessCost array in the same order as the CE array.

EXAMPLE

```
LFNList_t lfns;
lfns.resize(1);
lfns[0]=''lfn:myHiggs'';

CEList_t ces;
ces.resize(1);
ces[0]=''lxshare0313.cern.ch'';

Protocols_t protocols;
```

```
protocols.resize(1);
protocols[0]=''gsiftp'';

AccessCosts_t accessCosts;

accessCosts = getAccessCost(lfns, ces, protocols);
```