



# DataGrid

## EDG-BROKERINFO USER GUIDE

---

Document identifier:

Date: **15/07/2003**

Work package: **WP1-WP2**

Document status: **DRAFT**

Deliverable identifier:

---

Abstract: This document presents what is the BrokerInfo file, the edg-brokerinfo command line interface and the correspondent API, with some examples as well.

### Delivery Slip

	Name	Partner	Date	Signature
From	Flavia Donno	INFN	15/07/2003	
Verified by				

### Document Log

Issue	Date	Comment	Author
0.0	21/03/2003	First draft	Flavia Donn., Livio Salconi, Massimo Sgaravatto
2.1	11/07/2003	Add support for SE Mountpoint and FreeSpace	Flavia Donno
2.1	15/07/2003	Add linking example	Flavia Donno

### Document Change Record

Issue	Item	Reason for Change
2.1	Brokerinfo API/CLI	Add Client APIs and CLI

### Software / Documents / URLs

Link	Refer to:
[U1]	<a href="http://www.pd.infn.it/~sgaravat/Grid/datagrid-01-not-0113-1_2.pdf">http://www.pd.infn.it/~sgaravat/Grid/datagrid-01-not-0113-1_2.pdf</a>
[U2]	<a href="http://datagrid.in2p3.fr/distribution/datagrid/wp2/RPMS/BrokerInfo-gcc32dbg-3.0-2.i386.rpm">http://datagrid.in2p3.fr/distribution/datagrid/wp2/RPMS/BrokerInfo-gcc32dbg-3.0-2.i386.rpm</a>

### Terminology / Glossary

EDG	Eropean DataGrid
UI	User Interface
WN	Worker Node
CE	Computing Element
SE	Storage Element
RB	Resource Broker



## Table of Contents

1. INTRODUCTION .....	5
2. BRIEF DESCRIPTION FOR THE NEED OF THE BROKERINFO FILE .....	6
3. THE BROKER INFO FILE .....	7
4. USING THE COMMAND LINE INTERFACE.....	9
4.1. EXAMPLE: GETTING THE CE ID USING THE EDG-BROKERINFO COMMAND FROM THE WN .....	10
4.2. EXAMPLE: PARSING THE BROKERINFO FILE FROM THE UI.....	12
5. THE BROKERINFO API - C++ CLASS.....	14





## 1. INTRODUCTION

In the BrokerInfo file (created by the RB), the various information concerning a job (chosen CE, location of the considered data, etc.) is available, so that they can be used by the applications (in particular to get the handle to the physical file names that the application has to consider).

The BrokerInfo file is wrote relying on the Condor ClassAds, and therefore it can not be so easy to read and interpret it. We therefore provide a command line tool and a C++ API (described in the next chapters) to allow users to easily parse this file.

A first document discussing about BrokerInfo file was [U1]: chapters [2] and [3] come from that document.

The edg-brokerinfo software is available in the EDG repository ([U2]): BrokerInfo\*.rpm

## 2. BRIEF DESCRIPTION FOR THE NEED OF THE BROKERINFO FILE

When the user specifies in the *InputData* field of the JDL expression (used when a job is submitted) the input data selection, the Resource Broker finds out where (in which Storage Element(s)) these data are physically stored, and based on this info, chooses the closest most suitable Computing Element where to submit the job. Therefore the Resource Broker takes into account the replica information for scheduling. In the *InputData* field, the user can specify lists of Logical File Names and/or GUID or logical file collections LCN.

The Storage Element is chosen considering also the protocol that the application is able to “speak” (this is defined in the *DataAccessProtocol* field of the JDL expression) and “satisfied” by the Storage Element. Here we give a set of definitions to better understand what follows:

### ***LFN = Logical File Name***

It is an arbitrary string corresponding to the logical name of the file, is an alias to one SFN.

Eg.: lfn:myfile.dat

### ***GUID = Grid Unique Identifiers***

It is a number that identify one (and only one) SFN.

Eg: guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70

### ***LCN = Logical Collection Names***

Is a label for identify an already defined file collection.

Eg: lcn:Higgs

### ***SFN = Storage File Name***

It has the following URL form: sfn://<hostname>/<path>/<filename>, where <hostname> is the hostname of the Storage Element serving the file, <path> is a protocol independent common path and <filename> is an arbitrary string which corresponds to the physical name of the file.

Eg.: sfn://se1.cern.ch/data/myfile.dat

### 3. THE BROKER INFO FILE

WP1 and WP2 agreed on a possible approach to this problem: the idea is to “pack up” this information in a file (*.BrokerInfo*), which is sent to the job-working directory of the worker node with the input application sandbox. Here is the format (based on Condor ClassAds) and the content of this broker info file:

```
[
ComputingElement=[
    name="cel.infn.it:2119/jobmanager-pbs-short";
    AccessCost=[ time=3456; size=10000 ];
    CloseStorageElements={
        [
            GlueSASStateAvailableSpace = 10027844;
            GlueCESEBindCEAccesspoint = "/diskmi";
            mount = "GlueCESEBindCEAccessPoint";
            name = "se1.pd.infn.it";
            freespace = "GlueSASStateAvailableSpace"
        ],
        [
            GlueSASStateAvailableSpace = 10324254;
            GlueCESEBindCEAccesspoint = "/diskpd";
            mount = "GlueCESEBindCEAccessPoint";
            name = "se2.pd.infn.it";
            freespace = "GlueSASStateAvailableSpace"
        ]
    }
];
DataAccessProtocol={ "gridftp" , "file" };
InputFNs={ [ name="lfn:file76.dat";
            SFNs={ "sfn://se1.pd.infn.it/data00/file76.dat",
                  "sfn://se.cern.ch/cms/mn/file76.dat" }
            ],
            [ name="lfn:filex";
            SFNs={ "sfn://se1.pd.infn.it/data00/filex",
                  "sfn://se.in2p3.fr/data/00/filex" }
            ],
            [ name="lfn:nmdb";
            SFNs={ "sfn://se.cern.ch/cms/mn/cal1.DB" }
            ]
};
StorageElements={ [ name="se1.pd.infn.it";
                    protocols={ [ name="gridftp"; port=4444 ],
                                [ name="file"; port=undefined ] }
                    }
```

```
    ],
    [ name="se.cern.ch";
      protocols={ [ name="gridftp"; port=4444 ],
                  [ name="rfio"; port=5555 ] }
    ],
    [ name="se.in2p3.fr";
      protocols={ [ name="gridftp"; port=4455 ] }
    ]
  ];
VirtualOrganization="CMS";
]
```

*CE = ResourceId* specifies the identifier of the Computing Element where the job has been dispatched.

*DataAccessProtocol* = {*proto*<sub>1</sub>, *proto*<sub>2</sub>, ..., *proto*<sub>p</sub>} is the list of protocols that the application is able “to speak” (this is the value specified as *DataAccessProtocol* in the JDL expression).

*InputFNs* is the list of logical file names specified in the *InputData* attribute of the JDL expression.

Then, for each logical file name specified in the *InputData* attribute, the list of all correspondent physical file names is specified: with *lfn* one specifies the LFNs specified in the *InputData* attribute, while *PFNs* is a list of list: the first list represents the physical file names associated to the first logical file specified in the *LFNs* list, the second list corresponds to the second LFN, etc...

Then the list *SEs* is specified: these are Storage Elements storing files specified in the *PFNs* and/or *LFN2PFN* lists.

For each of these storage elements, a list of “supported” protocols (attribute *SEProtocols*), and, for each protocol, the correspondent port number (attribute *SEPorts*) are then provided.

*CloseSEs* = {*SE*<sub>1</sub>, *SE*<sub>2</sub>, ..., *SE*<sub>n</sub>} defines the list of Storage Elements “close” to the Computing Element where the job has been submitted. We assume that this information is available in the MDS (provided by the WP4 information providers).

Each storage element is identified by the correspondent full host name.

For each of these Storage Elements, the mount point to that Storage Element from the Computing Element where the job has been dispatched (if there is “local access”) is specified: the first element of the list is the mount point for the first SE specified in the *CloseSEs* list, the second element corresponds to the second element of the *CloseSEs* list, etc...

Both a command line tool (described in section 4) and a C++ API (described in section 5) is provided, to allow users to easily “read” the content of this BrokerInfo file.



## 4. USING THE COMMAND LINE INTERFACE

The edg-brokerinfo is the command line tool to “read” the BrokerInfo file.

The edg-brokerinfo command is installed on UI and WN machines. It is normally located in:  
`/opt/edg/bin/edg-brokerinfo`

If this command is called from the shell prompt without specifying any arguments, a message is returned:

```
[user@UI] # /opt/edg/bin/edg-brokerinfo

/opt/edg/bin/edg-brokerinfo: please specify a function with parameters
use /opt/edg/bin/edg-brokerinfo --help for functions and parameters list
```

With the `--help` option, users can get a the list about available flags, functions and arguments:

```
[user@UI] # edg-brokerinfo --help

Use: # edg-brokerinfo [-v] [-f filename] function [parameter] [parameter] ...
-v : for verbose, formatted output
-f : for specify a brokerinfo file

supported funtions are:
./biCLI getCE
./biCLI getDataAccessProtocol
./biCLI getInputData
./biCLI getSEs
./biCLI getCloseSEs
./biCLI getSEMoutPoint <SE>
./biCLI getSEFreeSpace <SE>
./biCLI getLFN2SFN <PFN>
./biCLI getSEProtocols <SE>
./biCLI getSEPort <SE> <Protocol>
./biCLI getVirtualOrganization
./biCLI getAccessCost
```

Therefore there are two possible flags (-v and -f):

`-v` : for a verbose, formatted output. The interface gives various information like: the API function name and namespace, the API returned value (BI\_SUCCESS or BI\_ERROR) and, of course, the desired output value. For example compare the two following outputs (with and without `-v` flag):

```
[user@UI] # edg-brokerinfo -v getCE
```

```
BrokerInfo::getBIFileName() : .BrokerInfo
BrokerInfo::getCE() :
-> lxde01.pd.infn.it:2119/jobmanager-lsf-grid01
-> BI_SUCCESS
```

```
[user@UI] # edg-brokerinfo getCE
lxde01.pd.infn.it:2119/jobmanager-lsf-grid01
```

The default is without verbose output.

**-f** : on UI let users specify the pathname of the BrokerInfo file, e.g.:

```
[user@UI] # edg-brokerinfo -v -f bi_bile_1 getCE
lxde01.pd.infn.it:2119/jobmanager-lsf-grid01
```

There are basically two ways for parsing elements from a BrokerInfo file.

The first one is directly from the job, and therefore from the WN where the job is running (see the example in section 4.1).

The second one is from the UI, and therefore after job execution, after having retrieved the BrokerInfo file (in section 4.2 an example is provided).

To use the edg-brokerinfo command, the **EDG\_WL\_RB\_BROKERINFO** must be defined: it should refer to the pathname of the BrokerInfo file. If is not defined the interface will give back an error message. It is not necessary to set this variable in the worker node, since it is “automatically” defined by the system.

For what concerns the various functions:

**functionName <functionParameter> :**

they are explained in details in the API sections (sections 5).

#### 4.1. EXAMPLE: GETTING THE CE ID USING THE EDG-BROKERINFO COMMAND FROM THE WN

Write a brokerinfo\_test\_1.jdl file, referring to a job which issues the edg-brokerinfo command to get the CE where the job has been dispatched, e.g.:



```
Executable      = "/opt/edg/bin/edg-brokerinfo";
Arguments       = "-v getCE";
StdOutput       = "stdout.txt";
StdError        = "stderr.txt";
OutputSandbox   = {"stdout.txt", "stderr.txt"};
```

Then submit this job from an UI machine to the grid:

```
[salconi@testbed009 jdl]$ dg-job-submit -o JId brokerinfo_test_1.jdl
```

Connecting to host grid004f.cnaf.infn.it, port 7771

Logging to host grid004f.cnaf.infn.it, port 15830

```
===== dg-job-submit Success =====
The job has been successfully submitted to the Resource Broker.
Use dg-job-status command to check job current status. Your job identifier (dg_jobId) is:
https://grid004f.cnaf.infn.it:7846/131.154.99.139/124918297052489?grid004f.cnaf.infn.it:7771
The dg_jobId has been saved in the following file:
/home/salconi/jdl/JId
=====
```

Then retrieve the sandbox files:

```
[salconi@testbed009 jdl]$ dg-job-get-output -i JId
```

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

Output sandbox files for the job:
-
https://grid004f.cnaf.infn.it:7846/131.154.99.139/124918297052489?grid004f.cnaf.infn.it:7771
have been successfully retrieved and stored in the directory:
/tmp/124918297052489
*****
```

Then check the standard output file (i.e. the output of the edg-brokerinfo command):

```
[salconi@testbed009 jdl]$ cat /tmp/124918297052489/stdout.txt
BrokerInfo::getCE():
-> testbed008.cnaf.infn.it:2119/jobmanager-pbs-short
-> BI_SUCCESS
```

## 4.2. EXAMPLE: PARSING THE BROKERINFO FILE FROM THE UI

Write a brokerinfo\_test\_2.jdl file like this:

```
Executable      = "getBI.sh";
StdOutput       = "stdout.txt";
StdError        = "stderr.txt";
InputSandbox    = {"getBIfile.sh "};
OutputSandbox   = {"stdout.txt", "stderr.txt"};
```

Where *getBIfile.sh* is a simple bash script like this:

```
#
# !/bin/sh
#
cat $EDG_WL_RB_BROKERINFO
```

which therefore prints in the standard output the content of the BrokerInfo file.

Submit the job and retrieve the output sandbox files.

```
[salconi@testbed009 jdl]$ dg-job-submit -o JId brokerinfo_test_2.jdl
```

```
Connecting to host grid004f.cnaf.infn.it, port 7771
```

```
Logging to host grid004f.cnaf.infn.it, port 15830
```

```
===== dg-job-submit Success =====
The job has been successfully submitted to the Resource Broker.
Use dg-job-status command to check job current status. Your job identifier (dg_jobId) is:
https://grid004f.cnaf.infn.it:7846/131.154.99.139/130820297825808?grid004f.cnaf.infn.it:7771
The dg_jobId has been saved in the following file:
/home/salconi/jdl/JId
=====
```

```
[salconi@testbed009 jdl]$ dg-job-get-output -i JId
```

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

Output sandbox files for the job:
-
https://grid004f.cnaf.infn.it:7846/131.154.99.139/131239299403696?grid004f.cnaf.infn.it:7771
have been successfully retrieved and stored in the directory:
/tmp/131239299403696
*****
```



---

```
[salconi@testbed009 jdl]$ cp /tmp/131239299403696/stdout.txt file_to_parse
```

Now the BrokerInfo file is a local file (*file\_to\_parse*) and is possible to parse it and to get from it all the needed information. For example it is possible to get the CE id:

```
[salconi@testbed009 jdl]$ edg-brokerinfo -v -f file_to_parse getCE
```

```
BrokerInfo::getCE() :  
-> testbed008.cnaf.infn.it:2119/jobmanager-pbs-short  
-> BI_SUCCESS  
[salconi@testbed009 jdl]$
```

## 5. THE BROKERINFO API - C++ CLASS

This is a simple class for parsing the *.BrokerInfo* file, which contains technical details and information for running an application. BrokerInfo has been developed like a singleton class, so only one object's instance can be created in a program context. There isn't a public data structure. Its private data structure contains:

- a string *BrokerInfoFile\_*, filled by the value of environment's variable EDG\_WL\_RB\_BROKERINFO;
- one *instance\_* handler, used for referencing the BrokerInfo object;
- one ClassAd\* *ad\_* object for parsing the file;

The public methods are:

- *BrokerInfo\* instance(void);*  
returns the handle of the singleton object.
- *BI\_Result getCE(std::string& CE);*  
returns BI\_SUCCESS if the file has a CE parameter specified, otherwise, if operation fails, a BI\_ERROR.  
The parameter *CE* is referenced to CE name, selected by the Resource Broker.
- *BI\_Result getDataAccessProtocol(std::vector<std::string>& DAPs);*  
returns BI\_SUCCESS if the file has a DataAccessProtocol parameter list specified from the user in the JDL file, otherwise, if operation fails, a BI\_ERROR.  
The string vector *DAPs* is referenced to the specified data access protocol list.
- *BI\_Result getInputData(std::vector<sd::string>& IDs);*  
returns BI\_SUCCESS if the file has an InputData parameter list specified from the user in the JDL file, otherwise, if operation fails, a BI\_ERROR.  
The string vector *IDs* is referenced to the InputData list: LFN, GUID or LFC.
- *BI\_Result getLFN2SFN(std::string LFN, std::vector<std::string>& SFNs);*  
returns BI\_SUCCESS if the file has a LFN to SFNs parameter list with correct mapping, otherwise, if operation fails, a BI\_ERROR.  
The parameter *LFN* contains the SFN, GUID or LCN value specified and the parameter *SFNs* will be referenced to the list of storage file name matched.

- *BI\_Result getSEs(std::vector<std::string>& SEs);*  
returns BI\_SUCCESS if the file has SE parameter list, otherwise, if operation fails, a BI\_ERROR.  
The parameter *SEs* is referenced to the list of storage element specified by Resource Broker.
- *BI\_Result getSEprotocols(std::string SE, std::vector<std::string>& SEProtos);*  
returns BI\_SUCCESS if the file has SEProtocols parameter list, otherwise, if operation fails, a BI\_ERROR.  
The parameter *SE* contains one storage element name and parameter *SEprotocols* is referenced to the list of protocols implemented on specified SE, values come from ResourceBroker matching.
- *BI\_Result getSEPort(std::string SE, std::string SEProtocol, std::string& SEPort);*  
returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
The parameter *SE* contains one storage element name and parameter *SEProtocol* contains the protocol name, parameter *SEPort* will be referenced to the value of matching port for that protocol, matching is done by Resource Broker.
- *BI\_Result getCloseSEs(std::vector<std::string>& SEs);*  
returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
The parameter list *SEs* will be referenced to the list of the computer element's nearest storage element hosts, values come from Resource Broker matching and from local CE farm configuration.
- *BI\_Result getSEMountPoint(std::string CloseSE, std::string& SEMount);*  
returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
The parameter *CloseSE* contains one "near" storage element name and parameter *SEMount* will be referenced to the relative CE NFS mount point.
- *BI\_Result getSEFreeSpace(std::string CloseSE, std::string& SEFreeSpace);*  
returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
The parameter *CloseSE* contains one "near" storage element name and parameter *SEFreeSpace* is returned and correspond to the amount of FreeSpace available on the SE.
- *BI\_Result getVirtualOrganization(std::string& VO);*  
Returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
The parameter *VO* is referenced to user's Virtual Organization.
- *BI\_Result getAccessCost(std::string& timeTotal, std::string& byteToTransf);*  
Returns BI\_SUCCESS or, if operation fails, a BI\_ERROR.  
Returns the WP2 getAccessCost function output for the the selected CE. The parameter *timeTotal* is referenced with approx. time needed to



replicate all files to a near SE and parameter *byteToTransf* is referenced to the approx. data size that must be replicated.

To link your program with the BrokerInfo library you can use the following example:

```
g++ mybrokerinfo_program.C -g -o mybrokerinfo_program \  
-I${EDG_LOCATION}/BrokerInfo \  
-L${EDG_LOCATION}/lib \  
-lBrokerInfo
```



The private methods implemented in BrokerInfo class are:

- *BrokerInfo(void);*  
is the constructor class, this method search and check the environment variable and read the brokerinfo file, create and fill the ClassAd object. If is not possible to parse the file with specified PARSE\_RULE value a fault assertion will be generated.
- *BI\_Result searchAD(std::string attrName, std::string& attrExpr, classad::ClassAd\* clAd);*  
is used for searching for an attribute attrName value inside a classad object.
- *void prettyStrList(std::string buffer, std::vector<std::string>& outList);*  
converts a string list into a string vector.
- *void prettyCAdList(std::string buffer, std::vector<std::string>& outList);*  
converts a classad list into a classad vector.
- *void prettyString(std::string& outStr);*  
*classad::ClassAd\* parserAD(std::string buffer);*  
removes quoting ("...") from a string, if the string is not quoted leaves it untouched.