DataGrid

JDL ATTRIBUTES

Document identifier:	**DataGrid-01-TEN-0142-0_2**
Date:	**28/10/2003**
Work package:	**WP1**
Partner:	**Datamat SpA**
Document status	
Deliverable identifier:	

Abstract: This note provides the description of JDL attributes supported by the EDG WMS release-2 software.

Delivery Slip

	Name	Partner	Date	Signature
From	Fabrizio Pacini	Datamat SpA	28/10/2003	
Verified by	Stefano Beco	Datamat SpA	28/10/2003	
Approved by				

Document Log

Issue	Date	Comment	Author
0_0	16/06/2003	First issue	Fabrizio Pacini
0_1	04/09/2003		Fabrizio Pacini
0_2	28/10/2003		Fabrizio Pacini

Document Change Record

Issue	Item	Reason for Change
0_1	General Update	– Update Std/In/Out/Err attributes definitio – Added JDL examples
0_2	Update for VOMS integration	– Update description of VirtualOrganisation attribute

Files

Software Products	User files
Word 2000	DataGrid-01-TEN-0142-0_2.doc
Acrobat Exchange 5.0	DataGrid-01-TEN-0142-0_2.pdf

CONTENT

1. INTRODUCTION 5

1.1. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS 5
1.2. DOCUMENT EVOLUTION PROCEDURE 6
1.3. TERMINOLOGY 6

2. ATTRIBUTES DESCRIPTION 8

2.1. TYPE 9
2.2. JOBTYPE 9
2.3. EXECUTABLE 10
2.4. ARGUMENTS 10
2.5. STDINPUT 11
2.6. STDOUTPUT 11
2.7. STDERROR 12
2.8. INPUTSANDBOX 12
2.9. OUTPUTSANDBOX 13
2.10. ENVIRONMENT 13
2.11. INPUTDATA 15
2.12. DATAACCESSPROTOCOL 15
2.13. OUTPUTSE 16
2.14. OUTPUTDATA 17
2.14.1. OutputFile 17
2.14.2. StorageElement 17
2.14.3. LogicalFileName 17
2.15. VIRTUALORGANISATION 18
2.16. RETRYCOUNT 19
2.17. MYPROXYSERVER 20
2.18. HLRLOCATION 20
2.19. NODENUMBER 21
2.20. JOBSTEPS 21
2.21. CURRENTSTEP 23
2.22. LISTENERPORT 23
2.23. REQUIREMENTS 23
2.24. RANK 24
2.25. FUZZYRANK 25

3. SPECIAL JDL EXPRESSIONS 26

3.1. GANG-MATCHING 26
3.2. GETACCESSCOST FUNCTION 26

4. JDL EXAMPLES 28

4.1. JOB WITHOUT DATA REQUIREMENTS 28
4.2. JOB WITH OUTPUT DATA 28
4.3. JOB WITH INPUT AND OUTPUT DATA 29
4.4. INTERACTIVE JOB 30
4.5. CHECKPOINTABLE JOB 31
4.6. MPI JOB 32

1. INTRODUCTION

This document provides a description of all JDL attributes supported by the EDG WMS for release 2 and a guide for the users to the building of job descriptions.

1.1. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] Definition of the architecture, technical plan and evaluation criteria for the resource co-allocation framework and mechanisms for parallel job partitioning
(http://www.infn.it/workload-grid/docs/DataGrid-01-D1.4-0127-1_0.{doc, pdf})

[A2] DataGrid Accounting System - Architecture v 1.0
(http://www.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf)

[A3] The Glue CE Schema
(http://www.cnaf.infn.it/~sergio/datatag/glue/v11/CE/index.htm)

[A4] Job Description Language HowTo – DataGrid-01-TEN-0102-02 – 17/12/2001
(http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2.pdf)

Reference documents

[R1] The Resource Broker Info file – DataGrid-01-TEN-0135-0_0
(http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0135-0_0.{doc,pdf})

[R2] LB-API Reference Document – DataGrid-01-TED-0139-0_0
(http://lindir.ics.muni.cz/dg_public/lb_api.pdf)

[R3] Job Partitioning and Checkpointing – DataGrid-01-TED-0119-0_3
(https://edms.cern.ch/file/347730/1/DataGrid-01-TED-0119-0_3.pdf)

[R4] "Gang-Matching in EDG WMS" - DataGrid-01-TEN-014X-0_0
(To be issued)

[R5] Design of a Replica Optimisation Framework
(https://edms.cern.ch/file/337977/1.7.2/wp2_replicaopt_api.ps)

1.2. DOCUMENT EVOLUTION PROCEDURE

The content of this document will be subjected to modification according to the following events:

- Comments received from Datagrid project members,
- Changes/evolutions/additions to the JDL.

1.3. TERMINOLOGY

Definitions

Condor	Condor is a High Throughput Computing (HTC) environment that can manage very large collections of distributively owned workstations
Globus	The Globus Toolkit is a set of software tools and libraries aimed at the building of computational grids and grid-based applications.

Glossary

class-ad	Classified advertisement
CE	Computing Element
CLI	Command Line Interface
DGAS	Datagrid Grid Accounting Service
EDG	European DataGrid
FQDN	Fully Qualified Domain Name
GIS	Grid Information Service, aka MDS
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
HLR	Home Location Register
IS	Information Service
job-ad	Class-ad describing a job
JA	Job Adapter
JC	Job Controller
JDL	Job Description Language
LB	Logging and Bookkeeping Service
LM	Log Monitor
LRMS	Local Resource Management System
MDS	Metacomputing Directory Service, aka GIS
MPI	Message Passing Interface
NS	Network Server
OS	Operating System
PA	Price Authority

PID	Process Identifier
PM	Project Month
RB	Resource Broker
SE	Storage Element
SI00	Spec Int 2000
SMP	Symmetric Multi Processor
TBC	To Be Confirmed
TBD	To Be Defined
UI	User Interface
VO	Virtual Organisation
VOMS	Virtual Organisation Membership Server
WM	Workload Manager
WMS	Workload Management System
WP	Work Package

2. ATTRIBUTES DESCRIPTION

The JDL is a fully extensible language, hence the user is allowed to use whatever attribute for the description of a job without incurring in errors from the JDL parser. Anyway only a certain set of attributes that we will refer as “supported attributes” from now on, is taken into account by the Workload Management System components in order to schedule and submit a job.

JDL attributes represent job specific information and specify in some way actions that have to be performed by the WMS to schedule the job.

Some of these attributes are provided by the user when she/he edits the job description file while some other (needed by the underlying WMS components) are automatically inserted by the UI before submitting the job.

A sub-set of the attributes that are inserted by the user is mandatory, i.e. necessary for the WMS to handle the job apopropriately and can be split in two categories:

- *Mandatory*: the lack of these attributes does not allow the submission of the job
- *Mandatory with default value*: the UI is able to provide default value for these attributes if they are missing in the job description.

Next section provides the complete list of the job JDL attributes supported by the WMS together with the format and rules to follow for adding them to the job description.

It is worth recalling that the requirements and rank expressions (see 2.23 and 2.24) that are evaluated by the RB during the match making process, can include attributes describing the CEs in the IS (attributes prefixed with “other.”) that are reported and described in [A3].

Before starting with the detailed attribute description we recall that a job description is composed by entries that are strings having the format *attribute* = *expression* and are terminated by the semicolon character. The whole description has to be included between square brackets, i.e. [<*job descr.*>]. The termination with the semicolon is not mandatory for the last attribute before the closing square bracket].

Attribute expressions can span several lines provided the semicolon is put only at the end of the whole expression. Comments must be preceded by a sharp character (#) or have to follow the C++ syntax, i.e a double slash (//) at the beginning of each line or statements begun/ended respectively with “/*” and “*/”.

2.1. TYPE

This a string representing the type of the request described by the JDL, e.g.

```
Type = "Job";
```

Possible values are:

- `Job`
- `DAG` (not supported in rel 2)
- `Reservation` (not supported in rel 2)
- `Co-allocation` (not supported in rel 2)

although for release 2 only `"Job"` is supported as request type. The value for this attribute is case insensitive.

- **Mandatory**: Yes
- **Default:** `"Job"`

2.2. JOBTYPE

This a string or a list of strings representing the type of the job described by the JDL, e.g.:

```
JobType = "Interactive";
```

or

```
JobType = {"Checkpointable", "MPICH"};
```

Possible values are:

- `Normal`
- `Interactive`
- `Checkpointable`
- `MPICH`
- `Partitionable` (not supported in rel 2)
- `Checkpointable, Interactive`
- `Checkpointable, MPICH`

This attributes only makes sense when the `Type` attribute equals to `"Job"`. The value for this attribute is case insensitive.

- **Mandatory**: Yes
- **Default:** `"Normal"`

2.3. EXECUTABLE

This a string representing the executable/command name.

The user can specify an executable that lies already on the remote CE and in this case the absolute path, possibly including environment variables of this file should be specified, e.g.:

```
Executable = "/usr/local/java/j2sdk1.4.0_01/bin/java";
```

The other possibility is to provide a local executable name, which will be staged from the UI node to the Computing Element WN. In this case only the file name has to be specified as executable. The absolute path on the local file system executable should be then listed in the *InputSandbox* attribute expression to meke it be transferred. E.g.:

```
Executable = "cms_sim.exe";
InputSandbox = {"/home/edguser/sim/cms_sim.exe", …… };
```

It is important to remark that if the job needs for the execution some command line arguments, they have to be specified through the *Arguments* attribute.

- **Mandatory**: Yes
- **Default:** No

2.4. ARGUMENTS

This is a string containing all the job command line arguments.

E.g. an executable *sum* that has to be started as:

```
$ sum  N1 N2 -out result.out
```

is described by:

```
Executable = "sum";
Arguments = "N1 N2 -out result.out";
```

If you want to specify a quoted string inside the Arguments then you have to escape quotes with the \ character. E.g. when describing a job like:

```
$ grep -i "my name" *.txt
```

you will have to specify:

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

Analogously, if the job takes as argument a string containing a special character (e.g. the job is the *tail* command issued on a file whose name contains the ampersand character, say *file1&file2*), since on the shell line you would have to write:

```
$ tail -f file1\&file2
```

in the JDL you'll have to write:

```
Executable = "/usr/bin/tail";
```

```
Arguments = "-f file1\\\&file2";
```

i.e. a \ for each special character.

In general, special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \. The character "`" cannot be specified in the JDL.

- **Mandatory**: No
- **Default:** No

2.5. STDINPUT

This is a string representing the standard input of the job.

It can be an absolute path possibly including environment variables (wild cards are instead not allowed), i.e. it is already available available on the CE, e.g.

```
StdInput = "/var/tpm/jobInput";
```

or just a file name, e.g.

```
StdInput = "myjobInput";
```

and this means that file staging is required and it has to be added to the *InputSandbox* file list. The same mechanism as described for the *Executable* attribute can be applied to *StdInput*.

This attribute cannot be included in the JDL for Interactive jobs as the standard input for such jobs is either provided interactively by the user or read from a named pipe on the UI machine.

- **Mandatory**: No
- **Default:** No

2.6. STDOUTPUT

This is a string representing the file name where the standard output of the job is saved.

The user can specify either a file name or an absolute path or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdOutput = "myjobOutput";
StdOutput = "/tmp/myjobOutput";
StdOutput = "event1/myjobOutput";
```

Wild cards are not allowed. The value specified for StdError can be the same as the one for StdOutput: this means that the two standard streams of the job are saved in the same file.

To have this file staged back on the submitting machine the user has to list the file name also in the *OutputSandbox* attribute expression and use e.g the **edg-job-get-output** command.

This attribute cannot be included in the JDL for Interactive jobs as the standard output for such jobs is either displayed directly to the user or saved on a named pipe on the UI machine.

- **Mandatory**: No
- **Default:** No

2.7. STDERROR

This is a string representing the file name where the standard error of the job is saved.

The user can specify either a file name or an absolute path or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdError = "myjobError";
StdError = "/var/tmp/myjobError";
StdError = "event1/myjobError";
```

Wild cards are not allowed. The value specified for StdError can be the same as the one for StdOutput: this means that the two standard streams of the job are saved in the same file.

To have this file staged back on the submitting machine the user has to list the file name also in the *OutputSandbox* attribute expression and use e.g the **edg-job-get-output** command.

This attribute cannot be included in the JDL for Interactive jobs as the standard output for such jobs is either displayed directly to the user or saved on a named pipe on the UI machine.

- **Mandatory**: No
- **Default:** No

2.8. INPUTSANDBOX

This is a string or a list of strings identifying the list of files on the UI local disk needed by the job for running and hence needed to be transferred from the UI to the CE. Wildcards and environment variables are admitted in the specification of this attribute (both are resolved on the UI node). File names can be provided as simple file names, absolute paths or relative paths with respect to the current working directory. The *InputSandbox* file list cannot contain two or more files having the same name (even if in different paths) as when transferred on the WN they would overwrite each other.

This attribute is also used to accomplish executable and standard input staging from the submitting machine to the CE where job execution takes place as explained above.

It is important to note that since `globus-url-copy` (the Globus command used for the *InputSanbox* files staging) in general doesn't preserve the x flag, the script specified as *Executable* in the JDL (on which chmod +x is done automatically by the WP1 JobWrapper),

should perform a chmod +x for all the files needing execution permission, that are transferred within the *InputSandbox* of the job. Hereafter follows an example of *InputSandbox* setting:

```
InputSandbox = {
            "/tmp/ns.log",
            "/home/edguser/HandsOn/mytest.exe",
            "myscript.sh",
            "/home/edguser/HandsOn/file*",
            "/home/edguser/data/*",
            "$EDG_TMP/mytest.conf"
            };
```

- **Mandatory**: No
- **Default:** No

2.9. OUTPUTSANDBOX

This is a string or a list of strings identifying the list of files generated by the job on the WN, which have to be retrieved. The listed files are transferred on the UI local file system by mean of the **edg-job-get-output** command. Wildcards are currently not admitted in the specification of this attribute. File names can be provided as simple file names, absolute paths or relative paths with respect to the current working directory. The *OutputSandbox* file list cannot contain two or more files having the same name (even if in different paths), as when transferred on the UI they would overwrite each other.

Hereafter is provided an example of the OutputSandbox attribute:

```
OutputSandbox = {
            "myjobOutput",
            "myjobError",
            "run1/event1",
            "run1/event2",
            "/tmp/myjob.log"
            };
```

- **Mandatory**: No
- **Default:** No

2.10. ENVIRONMENT

This is a list of string representing environment settings that have to be performed on the submitting machine and are needed by the job to run properly. Each item of the list is an equality “VAR_NAME=VAR_VALUE”. E.g.:

```
Environment  = {"JOB_LOG_FILE=/tmp/myjob.log",
                "ORACLE_SID=edg_rdbms_1",
                "JAVABIN=/usr/local/java"};
```

- **Mandatory**: No
- **Default:** No

2.11. INPUTDATA

This is a string or a list of strings representing the Logical File Names (LFN) or Grid Unique ID-entifiers (GUID) needed by the job as input to process. A LFN has the form of a URI and a GUID is 40 characters UUID. For more information please refer to [R3]. These data are stored in SEs and published in replica catalogues.

The listed file names are used by the RB to find the CE from which the specified files can be better accessed and schedules the job to run there.

Listed names have to be prefixed with "lfn:" and "guid:" (both lowercase) to indicate that they are respectively LFNs or GUIDs. E.g.:

```
InputData = {
          "lfn:EOtestfile" ,
          "lfn:cmstestfile",
          "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70",
          "guid:32d7v341-2d8k-09e7-11f3-8c409f8c7u67"
          };
```

Wildcards are not admitted when specifying this attribute.

- **Mandatory**: Yes (only if *DataAccessProtocol* has been specified)
- **Default:** No

2.12. DATAACCESSPROTOCOL

This is a string or list of strings representing the protocol or the list of protocols that the application is able to "speak" for accessing files listed in *InputData* on a given SE. The RB matches indeed this attribute with the protocols supported by the SE, as published in the IS.

Hereafter is reported an example of the *DataAccessProtocol* attribute:

```
DataAccessProtocol = {
                "file",
                "gridftp"
                };
```

Supported protocols in the EDG testbed are currently *gridftp*, *file* and *rfio* (lowercase) but there is no restriction to the protocol names that can be specified through this attribute.

- **Mandatory**: Yes (only if *InputData* has been specified)
- **Default:** No

2.13. OUTPUTSE

This is a string representing the URI of the Storage Element where the user wants to store the output data. Once specified, this attribute is used by the RB to find a CE being "close" to this SE and schedule the job there. E.g.:

```
OutputSE = "grid001.cnaf.infn.it";
```

- **Mandatory**: No
- **Default:** No

2.14. OUTPUTDATA

This attribute allows the user to ask for the automatic upload and registration of datasets produced by the job on the WN. Through this attribute it is possible to indicate for each output file the LFN to be used for registration and the SE on which the file has to be uploaded. The *OutputData* attribute is not mandatory.

OutputData is a list of classads where each classad contains the following three attributes:

- *OutputFile*
- *StorageElement*
- *LogicalFileName*

These three attributes are only admitted if members of one of the classads composing *OutputData*. They cannot be specified independently in the job JDL.

The following sub-sections report the description of the mentioned attributes:

2.14.1. OutputFile

This is a string attribute representing the name of the output file, generated by the job on the WN, which has to be automatically uploaded and registered by the WMS.

Wildcards are not admitted in the specification of this attribute. File names can be provided as simple file names, absolute paths or relative paths with respect to the current working directory.

- **Mandatory**: Yes (only if *OutputData* has been specified)
- **Default:** No

2.14.2. StorageElement

This is a string representing the URI of the Storage Element where the output file specified in the corresponding *OutputFile* attribute has to be uploaded by the WMS.

This attribute is not taken into account during the matchmaking. If this attribute is not specified, the WMS automatically takes a SE close to the CE for uploading the file. Unless the user has particular constraints, it is hence suggested to not specify this attribute so that the close SE are considered.

- **Mandatory**: No
- **Default:** a SE close to the CE where the job is submitted

2.14.3. LogicalFileName

This is a string representing the logical file name (LFN) the user wants to associate to the output file when registering it to the Replica Catalogue. The specified name has to be prefixed by "*lfn:*" (lowercase).

If this attribute is not specified then the corresponding output file is registered with a GUID that is assigned automatically by the WP2 services (RM).

- **Mandatory**: No
- **Default:** No (a GUID is assigned by WP2 services)

Hereafter follows an example of the *OutputData* attribute:

```
OutputData = {
                    [
                                OutputFile = "dataset_1.out ";
                                LogicalFileName = "lfn:test-result1";
                     ],
                    [
                                OutputFile = "dataset_2.out ";
                                StorageElement = "se001.cnaf.infn.it";
                     ],
                    ]
                                OutputFile = "cms/dataset_3.out";
                                StorageElement = "se012.to.infn.it";
                                LogicalFileName = "lfn:cms-outfile1";
                     ],
                    [
                                OutputFile = "dataset_4.out ";
                     ]
            };
```

If the attribute *OutputData* is found in the JDL then the *JobWrapper* at the end of the job calls the WP2 *copyAndRegister* service that copies the file from the WN onto the specified SE and registers it with the given LFN. If the specified LFN is already in use, WP2 RM registers the file with a newly generated identifier GUID (Grid Unique Identifier).

During this process the *JobWrapper* creates a file (named "*DSUpload_<unique_jobid_string>.out*") with the results of the operation that is put automatically in the *OutputSandbox* attribute list by the UI and can then be retrieved by the user.

- **Mandatory**: No
- **Default:** No

2.15. VIRTUALORGANISATION

This is a string representing the name of the VO the submitting user is currently working for.

If the proxy credential used at submission time contains VOMS extensions, then the value of the *VirtualOrganisation* attribute is always set (by the UI) to the default VO name (i.e. the first one) read from the proxy credential. User selections of a VO name different from that, e.g. through the *--vo* UI command option or editing the JDL directly, are overridden.

If instead the user proxy credential does not contains VOMS extensions and the **edg-job-submit** and the **edg-job-list-match** commands are issued with the *--vo* option, then the

value of this attribute is overwritten with the VO name specified on the command line. Hereafter follows an example for this attribute:

```
VirtualOrganisation = "atlas";
```

- **Mandatory**: Yes
- **Default:** The general precedence rule for determining the value of the *VirtualOrganisation* attribute is reported hereafter:
  - the default VO from the user proxy (if it contains VOMS extensions). This overrides all other possible way to choose the VO.
  - the VO name specified through the *--vo* or *--config-vo* options of the UI
  - the VO specified in the configuration file pointed by the EDG_WL_UI_CONFIG_VO environment variable,
  - the VirtualOrganisation attribute in the JDL
  - the default VO specified in the DefaultVO field (if any) of the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` configuration file.

  No default value is applied otherwise.

2.16. RETRYCOUNT

It is an integer representing the maximun number of job re-submission to be done in case of failure due to some grid component (i.e. not to the job itself).

RetryCount has to be a number equal or grater than 0 and the actual number of submission retries for a job is represented by the minimum value between *RetryCount* itself and the value of the *MaxRetryCount* parameter in the WM configuration file (default for *MaxRetryCount* is 10).

Hereafter follows an example for this attribute:

```
RetryCount = 3;
```

For example for disabling the job re-submission mechanism it suffices specifying:

```
RetryCount = 0;
```

- **Mandatory**: No
- **Default:** The value of the parameter *RetryCount* in the UI configuration file `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` if any.

  No default otherwise

2.17. MYPROXYSERVER

This is a string representing the MYProxy server address (*<host fqdn>)* where the user has registered her/his long-term proxy certificate.

For performing this registration by means of the *myproxy-init* command the user has to specify either through the `-s` option or the MYPROXY_SERVER environment variable the host name of the MyProxy server where to store the certificate proxy. The same hostname should be specified as value of the *MyProxyServer* attribute in the JDL.

The presence of this attribute in the JDL triggers indeed the WMS proxy renewal mechanism that is very useful when submitting long-running jobs to avoid job failure because it outlived the validity of the initial proxy used for the submission.

Proxy renewal can be enabled by default through the UI configuration by adding the *MyProxyServer* parameter to the `$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf` file. If present, indeed it makes the UI automatically add to the job description the *MyProxyServer* JDL attribute (if not specified by the user).

An example of the JDL setting is provided hereafter:

```
MyProxyServer = "skurut.cesnet.cz";
```

Note that the port number must not be provided.

- **Mandatory**: No
- **Default:** The value of the parameter *MyProxyServer* in the UI configuration file `$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf` if any.
  No default otherwise

2.18. HLRLOCATION

This is a string representing the Home Location Register address in the format

```
<host fqdn>:<port>:<X509contact string>
```

HLR is the service responsible for managing the economic transactions and the accounts of user and resources. The presence of the *HLRLocation* attribute in the JDL enables accounting in the WMS, i.e.:

- after the job submission the UI contacts the user's HLR and authorizes the payment of that job.
- on the CE, while the job runs, a sensor monitors the resource usage and when the job is done those data (usage records) are sent to the HLR

- the HLR computes the job cost according to the usage records and to the resource price and then debits the user account

WMS accounting can be enabled by default through the UI configuration by adding the *HLRLocation* parameter to the `$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf` file.

If present, indeed it makes the UI automatically add to the job description the *HLRLocation* JDL attribute (if not specified by the user).

An example of the JDL setting is provided hereafter:

```
HLRLocation =
"lilith.to.infn.it:56568:/O=CESNET/O=Masaryk University/CN=Miroslav Ruda"
```

- **Mandatory**: No
- **Default:** The value of the parameter *HLRLocation* in the UI configuration file `$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf` if any.
  No default otherwise

2.19. NODENUMBER

This is an integer greater than 1 specifying the number of nodes needed for a MPI job. This attribute is only allowed if the job type is MPICH (see 2.2). An example of the JDL setting is provided hereafter:

```
NodeNumber = 5;
```

The RB uses this attribute during the matchmaking for selecting those CE having a number of CPUs equal or greater than the one specified in *NodeNumber*.

- **Mandatory**: Yes (if the job type is MPICH)
- **Default:** No

2.20. JOBSTEPS

This can be either an integer representing the number of steps for a checkpointable job, e.g.:

```
JobSteps = 100000;
```

or a list of strings representing labels associated to the steps of a checkpointable job, e.g:

```
JobSteps = {"rawdata", "d0", "d1", "d2", "gomos"};
```

As describeb in [R3] a checkpointable application can be seen as “composed” by a set of sequential steps, where for example a step can represent the processing of a file, the analysis of an HEP event, etc. The various steps can be represented by a *main stepper* set of iterations and it is usually worth to save the state of the job after each step.
The content of the *main stepper* can be defined through the JDL attribute *JobSteps* see [R3] for details.
This attribute can only be set for checkpointable jobs.

- **Mandatory**: No
- **Default:** No

2.21. CURRENTSTEP

This is an integer equal or greater than 0 indicating the step number to be taken as the initial one when submitting a checkpointable job (see [R3] for details). If *JobSteps* is a list of labels then *CurrentStep* indicates the position of the label in the list. E.g.

```
CurrentStep = 2;
```

for the second example of section 2.20 would indicate the step labelled *"d1"* as the first step.

This attribute can only be set for checkpointable jobs. If not provided by the user it is set to 0 automatically by the UI.

- **Mandatory**: Yes (for checkpointable jobs)
- **Default:** 0

2.22. LISTENERPORT

This is an integer (>0) that represents the port on which the condor grid_console_shadow process started by the UI listens for the job standard streams. E.g.:

```
ListenerPort = 44000;
```

This attribute can only be included in the JDL for interactive jobs (see 2.2).

If this attribute is not included in the JDL then the listener port is the one automatically assigned by the OS (suggested choice).

- **Mandatory**: No
- **Default:** No (the port is assigned dynamically by the OS)

2.23. REQUIREMENTS

This is a Boolean ClassAd expression that uses C-like operators. It represents job requirements on resources. The Requirements expession can contain attributes that describe the CE in the IS and are hence prefixed with "***other.***". All these attributes are reported in the Glue Schema for the CE (see [A3]).

To have a job scheduled to run on a given CE, this Requirements expression must evaluate to true on the given CE. The evaluation of this expression is performed by the RB during the match making phase. Hereafter follows and example of requirements expression:

```
Requirements =    other.GlueCEInfoLRMSType == "PBS" &&
                  other.GlueCEInfoTotalCPUs > 2 &&
      Member("IDL1.7",other.GlueHostApplicationSoftwareRunTimeEnvironment)
```

The above expression requires a CE whose local resource manager is PBS, having at least 2 CPUs and the IDL software version 1.7 already installed.

The *Requirements* attribute is mandatory in the JDL. The requirements expression is assigned automatically by the UI to:

```
Requirements = other.GlueCEStateStatus == "Production" ;
```

if not specified by the user. This default value is configurable by means of the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` configuration file.

If the user has instead provided an expression for the *Requirements* attribute in the JDL, the one specified in the configuration file is added (in AND) to the existing one. E.g. if in the JDL file the user has specified:

```
Requirements = other.GlueCEInfoLRMSType == "PBS";
```

then the job description that is passed to the NS contains:

```
Requirements = (other.GlueCEInfoLRMSType == "PBS") &&
                (other.GlueCEStateStatus == "Production");
```

Obviously the setting `TRUE` for the *Requirements* in the configuration file would result in:

```
Requirements = (other.GlueCEInfoLRMSType == "PBS") && TRUE ;
```

and hence does not have any impact on the evaluation of job requirements.

- **Mandatory**: Yes
- **Default:** `other.GlueCEStateStatus == "Production"`

2.24. RANK

This is a ClassAd Floating-Point expression that states how to rank CEs that have already met the *Requirements* expression. Essentially, rank expresses a preference. A higher numeric value equals a better rank. The RB will give to the job the CE with the highest rank.

The *Rank* expression can contain attributes that describe the CE in the IS and are hence prefixed with "***other.***". All these attributes are reported in the Glue Schema for the CE (see [A3]).

The evaluation of the rank expression is performed by the RB during the match making phase. Hereafter follows and example of requirements expression:

```
Rank = other.GlueCEPolicyMaxRunningJobs – other.GlueCEStateRunningJobs;
```

With the above Rank, the preferred CEs are the ones having the greatest number of free slots for running jobs available.

The Rank attribute is mandatory in the JDL. The ranking expression, if not specified by the user, is assigned automatically by the UI to:

```
Rank = other.GlueCEStateFreeCPUs;
```

for MPICH jobs, whilst:

```
Rank = - other.GlueCEStateEstimatedResponseTime;
```

for all other job types. This default value is configurable by means of the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` configuration file.

- **Mandatory**: Yes
- **Default:** `other.GlueCEStateFreeCPUs` (for MPICH jobs)
  `- other.GlueCEStateEstimatedResponseTime` (other job types)

2.25. FUZZYRANK

This is a Boolean attribute that enables fuzzyness in the ranking computation. In other words if this attribute is set to `TRUE`, forces the matchmaking algorithm to adopt a stochastic selection criteria while searching for the best matching CE. E.g. specifying:

```
FuzzyRank = true;
```

in the submitted JDL, the rank values associated to each matching CE represent the probability that each CE has, to be selected as the best matching one. Namely, the higher is the probability to be selected the higher the rank value.

- **Mandatory**: No
- **Default:** `FALSE`

3. SPECIAL JDL EXPRESSIONS

Next sections briefly describe how it is possible to drive the resources discovery and selection process by means of special expressions for the *Requirements* and *Rank* attributes.

3.1. GANG-MATCHING

The matchmaking mainly occurs as a two-step process: entities (i.e., servers and customers) requiring matchmaking services express their characteristics, requirements and preferences to a matchmaker in *classified advertisements* (Step 1). Attributes of candidate classads are accessed via the pseudo-attribute `other` and the matchmaker employs a very generic *matchmaking algorithm* to evaluate the requirements and rank of the involved entities (Step 2). When the RB performs the matchmaking for scheduling a job, the involved entities are the job (whose classads has been provided by the user) and the CE (whose classad is built by the RB with the information from IS).

If we consider for example a job that requires a CE and a determined amount of free space on a SE to run successfully, the matchmaking solution to this problem requires three participants in the match (i.e., job, CE and SE), which cannot be accomodated by conventional (bilateral) matchmaking. The gangmatching feature of the classads library provides a multilateral matchmaking formalism to address this deficiency.

In order to exploit this new important extension of the classads library it suffices including the appropriate classads built-in functions in the requirements expression.

A useful example, as already premised, is the usage of gangmatching to require a certain amount of free space on a SE close to the execution CE. This can be achieved specifying the job *Requirements* expression as follows:

```
Requirements =  anyMatch(   other.storage.CloseSEs ,
                            target.GlueSAStateAvailableSpace > 200);
```

This makes indeed the RB find the CEs having a close SE with at least 200 MB of free space available for the VO the user belongs to.

The newly supported classads built-in functions are:

- `anyMatch()`
- `whichMatch()`
- `allMatch()`

Information and details about gangmatching and usage of this functions are provided in document [R4].

3.2. GETACCESSCOST FUNCTION

When data requirements (i.e. the *InputData* and *DataAccessProtocol* attributes) are specified in the JDL the RB, before the actual match making, performs a pre-match processing to find out those CEs satisfying user authorisation requirements and classify them according to the number of input files stored in storage element(s) which is (are) close to the CE itself and speak at least one of the protocols specified in the *DataAccessProtocol* JDL attribute. Then it

performs the Requirements checking phase starting from the first class until one or more suitable CEs are found. If needed performs the Rank evaluation phase to choose the best CE.

There is also another path that the RB can follow to select the best CE for running a job when data requirements have been specified. This is allowed by the Replica Manager *getAccessCost* function that given a CE and a set of LFNs (and/or GUIDs), provides the total cost (time) for accessing them (see [R5] for details). Using the *getAccesCost* on all the CEs satisfying the job requirements and on the *InputData* specified by the user in the JDL, the RB ranks the suitable CEs and chooses the "best" one.

The mentioned mechanism is triggered by the following ranking expression:

```
Rank = other.DataAccessCost;
```

Indeed such *Rank* makes the RB, skip the classification procedure described at the beginning of this section, perform the requirements checking phase and rank resources using the *getAccessCost* outcomes, i.e. select for submission the CE having the lowest cost for accessing input data.

It is worth noting that it is not possible to combine the "`other.DataAccessCost`" expression with other ranking expression.

4. JDL EXAMPLES

In the following sections are reported simple example of JDL describing different types of jobs.

4.1. JOB WITHOUT DATA REQUIREMENTS

```
[
    Type = "job";
    JobType = "normal";
    Executable = "script.sh";
    Arguments = "60";
    StdOutput = "sim.out";
    StdError = "sim.err";
    MyProxyServer = "skurut.cesnet.cz";
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    // This attribute triggers accounting
    HLRLocation      =      "lilith.to.infn.it:56568:/C=IT/O=INFN/OU=Personal
        Certificate/L=Torino/CN=Andrea Guarise/Email=A.Guarise@to.infn.it";
    InputSandbox = {
        "/home/fpacini/GUI/sbin/script.sh"
    };
    rank = (other.GlueCEPolicyMaxRunningJobs-other.GlueCEStateRunningJobs);
    // This is the default requirements expression
    requirements = other.GlueCEStateStatus == "Production" ;
]
```

4.2. JOB WITH OUTPUT DATA

```
[
    Type = "job";
    JobType = "normal";
    VirtualOrganisation = "cms";
    Executable = "test.sh";
    Arguments = "1 20000 sim1";
    StdInput = "file2";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {
        "sim.out",
        "sim.err"
    };
```

```
    RetryCount = 2;
    OutputData = {
        [
            // No StorageElement is specified - Close SE is taken
            OutputFile = "dataset1.out";
            LogicalFileName = "lfn:myoutdata.1"
        ],
        [
            OutputFile = "dataset2.out";
            LogicalFileName = "lfn:myoutdata.2"
        ]
    };
    InputSandbox = {
        "/home/mytest/JNI/test.sh",
        "/home/mytest/DATA/file2",
        "/home/mytest/DATA/sim.dat"
    };
    Environment = "SIM_ROOT=/usr/local/";
    rank = other.GlueHostMainMemoryRAMSize;
    // job is submitted to a CE having a close SE with at least 5GB free
    requirements = anyMatch( other.storage.CloseSEs,
                    target.GlueSAStateAvailableSpace > 5120);
]
```

4.3. JOB WITH INPUT AND OUTPUT DATA

```
[
    Type = "job";
    // JobType is not mandatory - If not specified "normal" is the default
    JobType = "normal";
    VirtualOrganisation = "cms";
    Executable = "test.sh";
    Arguments = "1 20000 sim1";
    StdInput = "file2";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {"sim.out", "sim.err"};
    // disable job re-submission in case of failure
    RetryCount = 0;
    InputData = {
        "lfn:mydatafile1",
        "lfn:mydatafile2",
        "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"
```

```
    };
    DataAccessProtocol = {"gridftp", "file"};
    OutputData = {
        [
            OutputFile = "dataset1.out";
            StorageElement = "grid011.pd.infn.it";
            LogicalFileName = "lfn:myoutdata.1"
        ],
        [
            OutputFile = "dataset2.out";
            LogicalFileName = "lfn:myoutdata.2"
        ],
        [
            OutputFile = "dataset3.out"
        ],
        [
            OutputFile = "dataset4.out";
            StorageElement = "grid001.ct.infn.it"
        ]
    };
    OutputSE = "grid011.pd.infn.it";
    InputSandbox = {
        "/home/fpacini/JNI/test.sh",
        "/home/fpacini/DATA/file2",
        "/home/fpacini/DATA/sim.dat",
        "/home/fpacini/HandsOn-0409/WP1testA"
    };
    // Ranking is done according to cost for accessing InputData
    rank = other.DataAccessCost;
    requirements        =        (other.GlueCEStateFreeCPUs>=2)        &&
                                 (other.GlueCEInfoLRMSType=="lsf")
]
```

4.4. INTERACTIVE JOB

```
  [
    Type = "job";
    JobType = "interactive";
    VirtualOrganisation = "bio";
    Executable = "scriptint.sh";
    RetryCount = 1;
    // grid_console_shadow listens on this port. If not specified is
    // assigned by the OS
```

```
    ListenerPort = 6000;
    FuzzyRank = true;
    InputSandbox = {
        "/home/fpacini/JDL2/fox/scriptint.sh",
        "/home/fpacini/JDL2/fox/cpi",
        "/home/fpacini/DATA/sim.dat"
    };
    requirements = (other.GlueHostOperatingSystemRelease == "LINUX") &&
                    (other.GlueHostMainMemoryRAMSize >= 128)
    // this is needed for Interactive jobs. Don't need to specify it. It is
    // added automatically by UI
                    && (other.GlueHostNetworkAdapterOutboundIP);
    rank = other.GlueHostBenchmarkSF00
]
```

4.5. CHECKPOINTABLE JOB

```
[
    Type = "job";
    JobType = "checkpointable";
    VirtualOrganisation = "eo";
    // This is the total number of steps for the job. Not mandatory
    // if your job already knows it
    JobSteps = 10000000;
    CurrentStep = 1;
    Executable = "hsum";
    Arguments = "200000 gsiftp://lxde01.pd.infn.it/tmp/root_test/";
    StdOutput = "sim.out";
    StdError = "sim.err";
    InputData = {
        "lfn:wp1-test-file-01-lfn",
        "lfn:wp1-test-file-02-lfn",
        "lfn:wp1-test-file-04-lfn"
    };
    DataAccessProtocol = {"file", "rfio"};
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    RetryCount = 3;
    InputSandbox = {
        "/home/fpacini/GUI/sbin/hsum"
    };
```

```
    // This is the default rank expression
    rank = -other.GlueCEStateEstimatedResponseTime;
    // semicolon ";" can be omitted for last attribute specification
    requirements = other.GlueCEInfoLRMSType=="pbs"
]
```

4.6. MPI JOB

```
[
    Type = "job";
    JobType = "mpich";
    VirtualOrganisation = "iteam";
    // This is the minimum number of CPU needed by the job
    NodeNumber = 6;
    Executable = "cpi";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    // This attribute triggers the proxy-renewal mechanism
    MyProxyServer = "skurut.cesnet.cz";
    RetryCount = 3;
    InputSandbox = {
        "/home/fpacini/JDL2/fox/cpi"
    };
   requirements      =      other.GlueHostNetworkAdapterOutboundIP      &&
     Member("IDL2.1",other.GlueHostApplicationSoftwareRunTimeEnvironment);
   rank = other.GlueCEStateFreeCPUs;;
]
```