EGEE

JOB DESCRIPTION LANGUAGE ATTRIBUTES SPECIFICATION
FOR THE GLITE WORKLOAD MANAGEMENT SYSTEM

Document identifier: WMS-JDL.odt
Date: 02/09/11
Activity: Document status:
Document link: https://edms.cern.ch/document/590869

Abstract: This document provides the specification of the Job Description Language attributes supported by the gLite Workload Management System (WMS)

Document Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Comment</th>
<th>Author/Partner</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>28/01/2005</td>
<td>Draft</td>
<td>Fabrizio Pacini</td>
</tr>
</tbody>
</table>
Abstract: This document provides the specification of the Job Description Language attributes supported by the gLite Workload Management System (WMS)

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Changes</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>05/05/2005</td>
<td>- Fix DAG attribute name retry =&gt; node_retry_count,</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add InputSandbox to partitionable job example.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add UserTags attribute description.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add OutputSandboxBaseDestURI attribute description.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add JobState attribute description.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add ListenerHost, ListenerPipeName attributes description.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add JobState attribute description</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>11/08/2005</td>
<td>- General review</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specify MyProxyServer inheritance for sub-jobs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add description of LBAddress attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add ExpiryTime attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Data requirements attributes clarified</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Review partitionable job description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added example on InputData</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Fix composition rules</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>14/12/2005</td>
<td>- Add DataRequirements attribute</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Fix DAG dependencies attribute description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add ShallowRetryCount attribute description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add AllowZippedISB attribute description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add PerusalFileEnable, PerusalTimeInterval and PerusalFilesDestURI attributes descriptions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Add NodesCollocation attribute description</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>04/01/2006</td>
<td>- Fix StdOutput description</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Clarify about HTTPS servers support</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>27/01/2006</td>
<td>- Clarify about wildcards support in Input and OutputSandbox</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Fix DataAccessProtocols</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>02/05/2006</td>
<td>- Update HLRLocation description</td>
<td>Fabrizio Pacini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Naming fix: max_nodes_running =&gt; max_running_nodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Clarify NodeName</td>
<td></td>
</tr>
</tbody>
</table>
Abstract: This document provides the specification of the Job Description Language attributes supported by the gLite Workload Management System (WMS)

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 0.8     | 25/05/2006 | - Retrycount/ShallowRetrycount is taken into account for nodes/sub-jobs  
- Added DefaultNodeRetryCount/DefaultNodeShallowRetryCount  
- Nodes/sub-jobs are also held in the TQ: added ExpiryTime at the DAG/Collection level for inheritance  
- Support for prologue/epilogue: added Prologue/PrologueArgs and Epilogue/EpilogueArgs attributes |
|         |      | Fabrizio Pacini |
| 0.9     | 09/11/07 | - PerusalTimeInterval default value changed  
- Checkpointable and Partitionable and Interactive jobs (and all related attributes) deleted because not supported  
- Short Deadline Jobs attributes added |
|         |      | Alessandro Maraschini |
| 1.0     | 11/24/10 | - JobType MPICH deprecated  
- NodeNumber attribute deprecated  
- CPUNumber attribute introduced  
- MPI jdl example updated  
- InputData and OutputData now supported  
- Checkpointable, Partitionable and Interactive Jobs deleted because no more supported  
- Other fixes |
|         |      | Sara Bertocco, Marco Cecchi, Massimo Sgaravatto |
|         | 11/03/11 | - Added three new parameters: smpgranularity, hostnumber, wholenodes  
- removed Nodescollocation attribute  
- added feedback matter |
|         |      | Alvise Dorigo |
| 02/09/11 | - Fixed example values in 3.21 section |
|         |      | Alvise Dorigo |
| 07/12/11 | - PerusalFilesDestURI removed |
|         |      | Marco Cecchi |
| 06/28/12 | - PerusalTimeInterval description changed (better description of the algorithm).  
- EnableWMSfeedback: replaced ‘enabled’ with ‘disabled’ |
|         |      | Marco Cecchi |
Content

1. INTRODUCTION ...................................................................................................................................................... 7
  1.1. PURPOSE .............................................................................................................................................................. 7
  1.2. REFERENCES .......................................................................................................................................................... 7
  1.3. TERMINOLOGY .............................................

2. REQUEST AND JOB TYPES ........................................................................................................................................... 10
  2.1. TYPE ..............................................................

3. JOB ATTRIBUTES DESCRIPTION ................................................................................................................................. 11
  3.1. JobType .................................................................................................................................................................
  3.2. EXECUTABLE ........................................................................................................................................................
  3.3. ARGUMENTS ...........................................................................................................................................................
  3.4. StdInput ...................................................................................................................................................................
  3.5. StdOutput ................................................................................................................................................................
  3.6. STDERR...................................................................................................................................................................
  3.7. INPUTSANDBOX ..............................................
  3.8. INPUTSANDBOXBASEURL ....................................... 14
  3.9. OUTPUTSANDBOX .......................................... 15
  3.10. OUTPUTSANDBOXDESTURL ...................................... 17
  3.11. OUTPUTSANDBOXBASEDESTURL .............................. 17
  3.12. PROLOGUE ....................................................... 18
  3.13. PROLOGUEARGUMENTS ........................................ 18
  3.14. EPILOGUE ......................................................... 19
  3.15. EPILOGUEARGUMENTS ......................................... 19
  3.16. ALLOWZIPPEDISB ................................................ 19
  3.17. ZIPPEDISB ........................................................ 20
  3.18. EXPIRYTIME ....................................................... 20
  3.19. ENVIRONMENT .................................................... 21
  3.20. PERUSALFILENAME ........................................... 21
  3.21. PERUSALTIMELIMIT ............................................ 22
  3.22. INPUTDATA ......................................................... 22
  3.23. STORAGEINDEX .................................................. 23
  3.24. DATACATALOG .................................................. 23
  3.25. DATAREQUIREMENTS ........................................... 24
  3.26. DATAACCESSPROTOCOL ...................................... 26
  3.27. OUTPUTSE ......................................................... 26
  3.28. OUTPUTDATA ....................................................... 26
  3.29. VIRTUALORGANISATION ......................................... 29
  3.30. RETRYCOUNT .........................................................
  3.31. SHALLOWRETRYCOUNT .........................................
  3.32. LBADDRESS .........................................................
  3.33. MYPROXYSERVER .................................................
  3.34. HLRLOCATION .....................................................
  3.35. JOBPROVENANCE ................................................
  3.36. NODENAME (DEPRECATED) .....................................
  3.37. SMPGRANULARITY .................................................
  3.38. WHOLENODES ....................................................
  3.39. HOSTNUMBER ....................................................
  3.40. CPUNUMBER ....................................................... 34
  3.41. REQUIREMENTS ...................................................
  3.42. RANK ............................................................... 35
  3.43. FUZZYRANK ....................................................... 36
  3.44. USERTAGS ........................................................
  3.45. SHORTDEADLINEJOB ...........................................
  3.46. ENABLEWMSFEEDBACK ...........................................

IST-2002-508833 PUBLIC 4 / 66
4. DAG ATTRIBUTES DESCRIPTION .............................................................................. 38
   4.1. TYPE.................................................................................................................. 40
   4.2. VIRTUALORGANISATION.................................................................................... 40
   4.3. MAX_RUNNING_NODES..................................................................................... 41
   4.4. HLRLOCATION................................................................................................ 41
   4.5. LBAADDRESS.................................................................................................. 41
   4.6. MYPROXYSERVER............................................................................................ 41
   4.7. JOBPROVENANCE............................................................................................ 41
   4.8. ALLOWZIPPEDISB............................................................................................ 42
   4.9. ZIPPEDISB....................................................................................................... 42
   4.10. EXPIRYTIME................................................................................................... 42
   4.11. PERUSALENABLE............................................................................................ 42
   4.12. USERTAGS....................................................................................................... 43
   4.13. REQUIREMENTS AND RANK......................................................................... 43
   4.14. INPUTSANDBOX AND INPUTSANDBOXBASEURI....................................... 43
   4.15. OUTPUTSANDBOXBASEDESTURI.................................................................. 44
   4.16. DEFAULTNODERETRYCOUNT....................................................................... 44
   4.17. DEFAULTNODESHALLOWRETRYCOUNT....................................................... 44
   4.18. NODES............................................................................................................ 44
   4.19. DEPENDENCIES............................................................................................. 45
   4.20. OUTPUTSANDBOX.......................................................................................... 46

5. PARAMETRIC JOB ATTRIBUTES DESCRIPTION ...................................................... 47
   5.1. PARAMETERS...................................................................................................... 48
   5.2. PARAMETERSTART........................................................................................... 49
   5.3. PARAMETERSTEP.............................................................................................. 49

6. JOBS COLLECTION ATTRIBUTES DESCRIPTION................................................... 50
   6.1. TYPE.................................................................................................................. 51
   6.2. VIRTUALORGANISATION.................................................................................... 51
   6.3. HLRLOCATION................................................................................................ 52
   6.4. LBAADDRESS.................................................................................................. 52
   6.5. MYPROXYSERVER............................................................................................ 52
   6.6. JOBPROVENANCE............................................................................................ 52
   6.7. ALLOWZIPPEDISB............................................................................................ 52
   6.8. ZIPPEDISB....................................................................................................... 53
   6.9. EXPIRYTIME................................................................................................... 53
   6.10. PERUSALENABLE............................................................................................ 53
   6.11. USERTAGS....................................................................................................... 53
   6.12. REQUIREMENTS AND RANK......................................................................... 53
   6.13. INPUTSANDBOX AND INPUTSANDBOXBASEURI....................................... 54
   6.14. OUTPUTSANDBOXBASEDESTURI.................................................................. 55
   6.15. DEFAULTNODERETRYCOUNT....................................................................... 55
   6.16. DEFAULTNODESHALLOWRETRYCOUNT....................................................... 55
   6.17. NODES............................................................................................................ 55
   6.18. OUTPUTSANDBOX.......................................................................................... 56

7. SPECIAL JDL EXPRESSIONS......................................................................................... 57
   7.1. GANG-MATCHING............................................................................................. 57

8. JDL EXAMPLES........................................................................................................... 58
   8.1. NORMAL JOB.................................................................................................... 58
   8.2. NORMAL JOB WITHOUT DATA REQUIREMENTS.......................................... 58
   8.3. NORMAL JOB WITH DATA REQUIREMENTS.................................................. 59
   8.4. JOB WITH OUTPUT DATA ............................................................................. 60
   8.5. JOB WITH INPUT AND OUTPUT DATA.......................................................... 61
   8.6. PARAMETRIC JOB............................................................................................ 62
8.7. MPI JOB

8.8. DAG

8.9. COLLECTION
1. INTRODUCTION

The Job Description Language (JDL) is a high-level, user-oriented language based on Condor classified advertisements (classads) for describing jobs and aggregates of jobs such as Direct Acyclic Graphs (DAG) and Collections. Being the JDL an extensible language the user is allowed to use whatever attribute for the description of a request without incurring in errors from the JDL parser. However, 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 (WMS) components in order to schedule and submit a job or the jobs of a complex request.

JDL attributes represent request-specific information and specify in some way actions that have to be performed by the WMS to schedule the job or the jobs of a complex request.

Some of the attributes in the JDL are mandatory. If the user does not specify them, the WMS cannot handle the request. For the other attributes the system may find a default value if they are necessary for processing the request.

Section 2 describes the requests/jobs types supported by the WMS and the way they can be composed. Sections 3 to 6 provide the complete list of the JDL attributes supported by the WMS together with their meaning and format and the rules to follow for building correct requests (i.e. Job, DAG and Collection) descriptions.

Before starting with the detailed attribute description please note that a request description is composed by entries that are strings having the format

\[ \text{attribute} = \text{expression}; \]

and are terminated by the semicolon character. The whole description has to be included between square brackets, i.e. \[ \text{<job descr.>} \]. 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 have a sharp character (#) or a double slash (//) at the beginning of each line. Comments spanning multiple lines or placed after the semicolon indicating the end of the attribute specification can be specified enclosing the text between “/*” and “*/”.

Moreover it is worth noting that the requirements and rank attribute expressions (see 3.44 and 3.45) that are evaluated by the WM during the match making process for selecting the CEs where the job will be sent, are built using the resources attributes, representing the characteristics and status of the resources in the grid. These resources attributes are not part of the predefined set of attributes for the JDL as their naming and meaning depends on the adopted Information Service schema. This independence of the JDL from the resources information schema allows targeting for the submission resources that are described by different Information Services without any changes in the job description language itself.

Please note that since the WMS (in particular the WMProxy component) exposes a publicly available WSDL interface, no assumption is made in the document (unless explicitly specified) about the client tool used to submit the JDL description of the job.

1.1. PURPOSE

The purpose of this document is to provide a detailed description of the syntax and semantics of the JDL attributes supported by the gLite WMS in order to help users in describing their applications to be submitted to the grid.

1.2. REFERENCES

[R1] EGEE: Enabling Grids for E-science in Europe

[R2] EGEE Middleware Architecture, EU Deliverable DJRA1.1,
https://edms.cern.ch/document/476451/>

[R3] EGEE Middleware Design, EU Deliverable DJRA1.2,
https://edms.cern.ch/document/487871/
[R4] 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.pdf

[R5] DataGrid Accounting System - Architecture v 1.0
http://www.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf


[R7] Job Description Language HowTo – DataGrid-01-TEN-0102-02
http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2.pdf

[R8] JDL Attributes - DataGrid-01-TEN-0142-0_2
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

[R9] 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

[R10] LB-API Reference Document – DataGrid-01-TED-0139-0_0


[R12] "Gang-Matching in EDG WMS" - DataGrid-01-TEN-014X-0_0

[R13] EGEE Middleware Architecture (revision), EU Deliverable DJRA1.4,
https://edms.cern.ch/document/594698/1.0

[R14] EGEE Middleware Design (revision), EU Deliverable DJRA1.5,
https://edms.cern.ch/document/606574/1.0

https://edms.cern.ch/document/674643/1

https://edms.cern.ch/document/572489/1

[R17] WMProxy Wiki pages

1.3. TERMINOLOGY

Glossary

class-ad Classified advertisement
CE Computing Element
CLI Command Line Interface
DGAS Datagrid Grid Accounting Service
DLI Data Location Interface
EDG European DataGrid
EGEE Enabling Grids for E-sciencE
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
ISM Information Super Market
job-ad    Class-ad describing a job
JA        Job Adapter
JC        Job Controller
JDL       Job Description Language
JRA       Joint Research Activity
LB        Logging and Bookkeeping
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
SD        Service Discovery Service
SE        Storage Element
SI00      Spec Int 2000
SMP       Symmetric Multi Processor
TBC       To Be Confirmed
TBD       To Be Defined
TQ        Task Queue
UI        User Interface
UUID      Universally Unique IDentifier
VO        Virtual Organisation
WM        Workload Manager
WMProxy   Workload Manager Proxy: the web service interface to the WMS
WMS       Workload Management System
WN        Worker Node
WP        Work Package

**Definitions**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condor</td>
<td>Condor is a High Throughput Computing (HTC) environment that can manage very large collections of distributively owned workstations</td>
</tr>
<tr>
<td>Globus</td>
<td>The Globus Toolkit is a set of software tools and libraries aimed at the building of computational grids and grid-based applications.</td>
</tr>
<tr>
<td>Request</td>
<td>An object that can be described through the JDL and has to be processed by the WMS. It can be a job (simple request) or a DAG or a Collection (complex requests).</td>
</tr>
</tbody>
</table>
2. REQUEST AND JOB TYPES

The JDL allows description of the following types of requests (the ones supported by the WMS):

- **Job** a simple job
- **DAG** a Direct Acyclic Graph of dependent jobs
- **Collection** a set of independent jobs

Although DAGs and Collections represent sets of jobs, they are described through a single JDL description and can hence be submitted in one shot to the WMS. Moreover upon submission of such kind of requests, the WMS (besides the ids of the single nodes) will provide the user with a collective identifier that will allow monitor and control of the whole set of jobs through a single handle.

As it is described in the following sections, the JDL structure for DAGs and Collections is in some way similar but they have been kept as separate request types as they represent sets of jobs with different kind of interrelation.

The WMS currently supports the following types for Jobs (this does not apply to DAGs and Collections):

- **Normal**: a simple batch job
- **MPICH (DEPRECATED)**: a parallel application using MPICH-P4 implementation of MPI.
- **Parametric**: a job whose JDL contains parametric attributes (e.g. Arguments, StdInput etc.) whose values can be made vary in order to obtain submission of several instances of similar jobs only differing for the value of the parameterized attributes

Analogously to DAGs and Collections, for job types whose submission results in the spawning of a set of jobs (i.e. Parametric jobs) the WMS (besides the ids of the single children) will provide the user with a collective identifier that will allow monitoring and control of the whole set of jobs through a single handle.

**Important Note**: Due to current limitations of the WMS the nodes of non single jobs (i.e. of a DAG/Collection) cannot be Parametric jobs.

As mentioned above the WMS supports different types of requests. The type of a submission request for the WMS is specified in the JDL through the “Type” attribute described in section 2.1.

2.1. TYPE

The **Type** attribute is a string representing the type of the request described by the JDL, e.g.

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

Possible values are:

- **Job**
- **DAG**
- **Collection**

The value for this attribute is case insensitive. If this attribute is not specified in the JDL description, the WMS will set it to “Job”.

- **Default**: "Job"
3. JOB ATTRIBUTES DESCRIPTION

This section reports the detailed description of the JDL attributes that can be specified for describing Job requests. A sub-section for each attribute is provided.

3.1. JOBTYPE

The JobType attribute is a string representing the type of the job described by the JDL, e.g.:

```
JobType = "Normal";
```

Possible values are:

- Normal
- MPICH (Deprecated)
- Parametric

This attribute only makes sense when the Type attribute equals to "Job". The value for this attribute is case insensitive. If not specified in the JDL the WMS will set it to "Normal"

- Default: "Normal"

3.2. EXECUTABLE

The Executable attribute is a string representing the executable/command name.

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

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

or equivalently

```
Executable = "$JAVA_HOME/bin/java";
```

The other possibility is to provide either an executable available on the local file system or an executable located on a remote gridFTP server accessible by the user (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN). In both cases the executable file will be staged from the original location to the WN.

In both cases only the file name has to be specified as executable. The URI of the executable should be then listed in the InputSandbox attribute expression to make it be transferred. E.g. respectively:

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

Or
Executable = "cms_sim.exe";
InputSandbox = {"gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe",
........};

Or

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

are accepted and interpreted as in the first case, i.e. the executable file is available on the local file system.

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. This attribute is mandatory.

- Mandatory: Yes
- Default: No

3.3. ARGUMENTS

The Arguments attribute 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’l 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
### 3.4. **STDINPUT**

The *StdInput* attribute is a string representing the standard input of the job. This means that the job is run as follows:

```
$ job < <standard input file>
```

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

```
StdInput = “/var/tpm/jobInput”;
```

or just a file name, e.g.

```
StdInput = “myjobInput”;
```

and this means that file needs to be made available on the WN where the job is run. Therefore the standard input file has to be added to the *InputSandbox* file list so that it will be copied to the WMS node and then downloaded on the WN by the WMS JobWrapper script. The same rules described for the *Executable* attribute apply to *StdInput*.

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

### 3.5. **STDOUTPUT**

The *StdOutput* attribute is a string representing the file name where the standard output of the job is saved on the WN.

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

```
StdOutput = “myjobOutput”;
StdOutput = “event1/myjobOutput”;
```

Wild cards are not allowed. The value specified for *StdError* can be the same as the one for *StdOutput* attribute: 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 glite-wms-job-output command [R15].

Alternatively the user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the *OutputSandboxDestURI* attribute expression. E.g.:

```
StdOutput = “myjobOutput”;
OutputSandbox = { “myjobOutput”, … };
OutputSandboxDestURI = {
    “gsiftp://fox.infn.it:5678/home/gftp/myjobOutput”,
    …
```
indicates that myjobOutput when the job has completed its execution has to be transferred on gsiftp://fox.infn.it:5678 in the /home/gftp directory.

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

### 3.6. STDERR

The *StdError* attribute is a string representing the file name where the standard error of the job is saved on the WN.

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

```plaintext
StdError = "myjobError";
StdError = "event1/myjobError";
```

Wild cards are not allowed. The value specified for *StdError* can be the same as the one for *StdOutput* attribute: 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 `glite-wms-job-output` command [R15].

Alternatively the user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the *OutputSandboxDestURI* attribute expression (see 3.5 for details).

The same rules as for the *StdOutput* apply to *StdError*.

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

### 3.7. INPUTSANDBOX

The *InputSandbox* attribute is a string or a list of strings identifying the list of files on the UI local file system or on an accessible gridFTP server (HTTPS servers are also supported but this requires to have the GridSite `htcp` client command installed on the WN) needed by the job for running. These files are transferred from the UI to the WMS node and then downloaded on the WN where the job is run by the WMS JobWrapper script. If located on a gridFTP server, they are directly downloaded on the WN where the job is run by the WMS JobWrapper script.

Wildcards and environment variables are admitted in the specification of this attribute only if the submission takes place through a client able to resolve them locally before passing the JDL to the WMProxy service. Admitted wildcard patterns are the ones supported by the Linux glob function. One can remove the special meaning of the characters '?', '*', and '[' by preceding them by a backslash.

File names can be provided as URI on a gridFTP/HTTPS server, simple file names, absolute paths and relative paths with respect to the current UI working directory. The *InputSandbox* file list cannot contain two or more files having the same name (even if in different paths) as when transferred in the job’s working directory on the WN they would overwrite each other.

LFNs cannot be given as *InputSandbox* file names; the *InputData* attribute (see 3.25.3) has to be used for that purpose.
This attribute is also used to accomplish executable and standard input staging to the WN where job execution takes place as explained above. The InputSandbox attribute meaning is strictly coupled with the value of the InputSandboxBaseURI defined in section 3.8 that specifies a common location on a gridFTP/HTTPS server where files in the InputSandbox not being already specified as URI are located.

Here below follows an example of InputSandbox setting:

```
InputSandbox = {
    "/tmp/ns.log",
    "mytest.exe",
    "myscript.sh",
    "data/event1.txt",
    "gsiftp://neo.datamat.it:5678/home/fpacini/cms_sim.exe ",
    "file:///tmp/myconf"
};
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

It means that:

- /tmp/log is located on the user local file system (accepted for backward compatibility)
- mytest.exe, myscript.sh and data/event1.txt are available on gsiftp://matrix.datamat.it:5432 in the /tmp directory
- /tmp/myconf is located on the user local file system (explicitly specified using the file:// prefix)

If the InputSandboxBaseURI is not specified than also mytest.exe, myscript.sh and data/event1.txt would be interpreted as located on the user local file system

- Mandatory: No
- Default: No

### 3.8. INPUTSANDBOXBASEURI

The InputSandboxBaseURI attribute is a string representing the URI on a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN) where the InputSandbox files that have been specified as simple file names and absolute/relative paths are available for being transferred on the WN before the job is started. E.g.

```
InputSandbox = {
    ...
    "data/event1.txt",
    ...
};
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```
makes the WMS consider

"gsiftp://matrix.datamat.it:5432/tmp/data/event1.txt"

for the transfer on the WN.

- Mandatory: No
- Default: No

3.9. OUTPUTSANDBOX

The OutputSandbox attribute is a string or a list of strings identifying the list of files generated by the job on the WN at runtime, which the user wants to retrieve. The listed files are transferred at job completion on the WMS node and can be downloaded on the UI local file system by mean of e.g. the glite-wms-job-output command [R15]. Wildcards are not admitted in the specification of this attribute.

This attribute can be combined with the OutputSandboxDestURI (described in section 3.10) or the OutputSandboxBaseDestURI (described in section 3.11) to have, upon job completion, the output directly copied to specified locations running a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN). Note that output files managed in this way are not retrieved by the glite-wms-job-output command.

File names can be provided as simple file names or relative paths with respect to the current working directory on the executing WN. The OutputSandbox file list should not contain two or more files having the same name (even if in different paths), as when transitorily transferred on the WMS machine they would overwrite each other unless different destination URI are explicitly specified in the OutputSandboxDestURI attribute.

Here below is provided an example of the OutputSandbox attribute:

OutputSandbox = {
  "myjobOutput",
  "myjobError",
  "run1/event1",
  "run1/event2",
};

This indicates that all the listed files (if actually generated by the job) will be available, after job completion, on the WMS node (if neither OutputSandboxDestURI nor OutputSandboxBaseDestURI have been specified) and will be available for retrieval using e.g. the glite-wms-job-output command.

- Mandatory: No
- Default: No
3.10. OUTPUTSANDBOXDESTURI

The `OutputSandboxDestURI` attribute is a string or a list of strings representing the URI(s) on a gridFTP/HTTPS server where the files listed in the `OutputSandbox` attribute have to be transferred at job completion.

The `OutputSandboxDestURI` list contains for each of the files specified in the `OutputSandbox` list the URI (including the file name) where it has to be transferred at job completion. E.g.

```plaintext
OutputSandbox = {
    "myjobOutput",
    "run1/event1",
    "myjobError"
};
OutputSandboxDestURI = {
    "gsiftp://matrix.datamat.it:5432/tmp/myjobOutput",
    "gsiftp://grid003.ct.infn.it:6789/home/cms/event1",
    "myjobError"
};
```

makes the WMS transfer respectively
- `myjobOutput` on `matrix.datamat.it` in the directory `/tmp`
- `event1` on `grid003.ct.infn.it` in the directory `/home/cms`
- `myjobError` on the WMS node

The `OutputSandboxDestURI` list must have the same cardinality as the `OutputSandbox` list, otherwise the JDL will be considered as invalid. Note that the file name specified in the `OutputSandbox` can be different from the corresponding destination file name specified in the `OutputSandboxBaseDestURI`.

The `OutputSandboxDestURI` attribute and the `OutputSandboxBaseDestURI` described in section 3.11 cannot be specified together in the same JDL.

If neither `OutputSandboxDestURI` nor `OutputSandboxBaseDestURI` are specified in the JDL then all the files listed in the `OutputSandbox` (if actually generated by the job) will be available, after job completion, on the WMS node and will be available for retrieval using e.g. the `glite-wms-job-output` command [R15]. Note that files whose URI points to a location different from the WMS node will be not retrieved by the `glite-wms-job-output` command.

- **Mandatory:** No
- **Default:** the job directory on the WMS node

3.11. OUTPUTSANDBOXBASEDESTURI

The `OutputSandboxBaseDestURI` attribute is a string representing the base URI on a gridFTP/HTTPS server, i.e., a directory on the server, where the files listed in the `OutputSandbox` attribute have to be transferred at job completion (HTTPS servers are also supported but this requires to have the GridSite `http` client command installed on the WN). All the `OutputSandbox` files are transferred to the location specified by the URI with the same names (only names in a flat directory) as the ones specified in the `OutputSandbox`. E.g.
OutputSandbox = {
    "myjobOutput",
    "run1/event1",
};
OutputSandboxBaseDestURI = "gsiftp://matrix.datamat.it:5432/tmp";

makes the WMS transfer both files in the /tmp directory of the gridFTP server matrix.datamat.it (note that event1 will go in /tmp and not in /tmp/run1).

The OutputSandboxBaseDestURI attribute and the OutputSandboxDestURI described in section 3.10 cannot be specified together in the same JDL.

If neither OutputSandboxDestURI nor OutputSandboxBaseDestURI are specified in the JDL then all the files listed in the OutputSandbox (if actually generated by the job) will be available, after job completion, on the WMS node and will be available for retrieval using e.g. the glite-wms-job-output command [R15].

− Mandatory: No
− Default: the job directory on the WMS node

3.12. PROLOGUE

The Prologue attribute is a string representing the executable/script name of the prologue.

The prologue is an executable run within the WMS job wrapper before the user job is started. It can be used for purposes ranging from application-specific checks that the job environment has been correctly set on the WN to actions like data transfers, database updates or MPI pre script. If the prologue fails the job wrapper terminates and the job is considered for resubmission. If the shallow resubmission is enabled the resubmission would be shallow, otherwise it would be deep (see 3.30 and 3.31).

The rules for specification of the Prologue attributes and its relationship with the InputSandbox attribute are exactly the same already described in section 3.2 for the Executable attribute.

Please note that the prologue is useful for testing job resubmission. For example the following specification:

    Prologue = "/bin/false";

will exercise the job shallow resubmission (deep if the shallow one has not been enabled).

− Mandatory: No
− Default: No

3.13. PROLOGUEARGUMENTS

The PrologueArguments attribute is a string containing all the prologue executable (see 3.12) command line arguments.

All the rules reported in the description of the Arguments attribute (see 3.3) also apply to the PrologueArguments attribute.
3.14. EPILOGUE
The Epilogue attribute is a string representing the executable/script name of the epilogue. The epilogue is an executable/script run within the WMS job wrapper after the user job completion. It can be used for purposes ranging from application-specific checks that the job performed correctly to actions like data transfers, database updates or MPI post script. If the epilogue fails the job is considered for deep resubmission (see 3.30).

The rules for specification of the Epilogue attributes and its relationship with the InputSandbox attribute are exactly the same already described in section 3.2 for the Executable attribute.

Please note that the epilogue is useful for testing job resubmission. For example the following specification:

```
Epilogue = "/bin/false";
```

will exercise the job deep resubmission.

- Mandatory: No
- Default: No

3.15. EPILOGUEARGUMENTS
The EpilogueArguments attribute is a string containing all the epilogue executable (see 3.14) command line arguments.

All the rules reported in the description of the Arguments attribute (see 3.3) also apply to the EpilogueArguments attribute.

- Mandatory: No
- Default: No

3.16. ALLOWZIPPEDISB
The AllowZippedISB attribute is a boolean attribute. When set to true, i.e.

```
AllowZippedISB = true;
```

makes the WMProxy client commands archive and compress all job input sandbox files into a single tar.gzipped file that is then transferred to the WMS.

Although the file compression rate could result in some cases negligible, this approach can be particularly useful when the job sandbox is composed by a large number of files and the cost of the authentication steps required by each single transfer call can have a big impact on the total amount of time for uploading the files.

This attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.
A default value for this attribute can be specified in the WMProxy client configuration file (AllowZippedISB parameter in glite_wms.conf)

Note that the AllowZippedISB is only used on the client side by the WMProxy glite-wms-job-submit command in order to know if it has to generate the tarball containing the job’s input sandbox files and fill the ZippedISB attribute (see 3.17) accordingly before sending the request to the WMProxy service. AllowZippedISB is not used at all by the WMProxy service.

Note that when job submission is done through the WMProxy client command glite-wms-job-submit if the JDL sets AllowZippedISB to true, then the ZippedISB attribute is set by the client command irrespective of what it contained (i.e. if ZippedISB is already present in the JDL its value is overridden).

- Mandatory: No
- Default: false (unless otherwise specified in the WMProxy client configuration)

3.17. ZIPPEDISB

The ZippedISB attribute is a string or a list of strings containing the file name of the compressed (gzip-ed) tarball containing input sandbox files for the job, e.g.:

    ZippedISB = "BossArchive_1_2_1.tgz";

If the ZippedISB attribute is set in the incoming JDL, the WMProxy service takes the specified archive (assumed to be located in the job’s input sandbox area on the WMS node) and explodes it in the right locations.

Note that this attribute MUST NOT be set when the submission is done through the WMProxy client commands and the AllowZippedISB attribute is set to true as upon that setting the glite-wms-job-submit command takes care of the creation of the compressed archive from the InputSandbox (3.7) files list, of the naming of the resulting tarball and of the appropriate setting of the ZippedISB attribute in the JDL.

Note moreover that when submitting through client tools other than the WMProxy client commands (e.g. WMProxy APIs, clients auto-generated from WSDL etc.), it must be made sure that all files contained in the compressed archive indicated by the ZippedISB attribute are also included in the InputSandbox attribute list. They would be otherwise ignored by the WMS job wrapper and not transferred on the job execution host.

Please refer to the WMProxy Wiki pages for details about the creation of the tarball and the JDL setting when submitting through the WMProxy API or through clients generated from the service WSDL.

- Mandatory: No
- Default: No

3.18. EXPIRYTIME

A job for which no compatible CEs have been found during the matchmaking phase is hold in the WMS Task Queue for a certain time (currently it is 1 day from job submission) so that it can be subjected again to matchmaking from time to time until a compatible CE is found. If no match is found after 1 day the job is aborted.
The *ExpiryTime* attribute is an integer representing the date and time (in seconds since epoch) until the job has to be considered valid by the WMS. E.g.

```
ExpiryTime = 1112339655;
```
tells that the job has to be considered valid for matchmaking until “2005-04-01 07:14:15”.
The *glite-wms-job-submit* command provides options (--valid, --to) to specify the value for the expiry time in a user-friendly format.
If the specified *ExpiryTime* exceeds one day from job submission then it is not taken into account by the WMS that will eventually hold the job in the TQ only for 1 day.

- **Mandatory:** No
- **Default:** one day after submission time.

### 3.19. ENVIRONMENT

The *Environment* attribute is a list of string representing environment settings that have to be performed on the execution machine and are needed by the job to run properly. The JobWrapper on the Worker Node performs these settings just before the job is started. 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

### 3.20. PERUSALFILEENABLE

The *PerusalFileEnable* attribute is a Boolean attribute that allows enabling the job file perusal support in the WMS. When this attribute is set to true, i.e.

```
PerusalFileEnable = true;
```
the user can inspect, while the job is running, the files generated by the job on the WN. This is achieved by uploading on a location on the WMS node (the WMS JobWrapper does it), at regular time intervals, chunks of the files generated by the job on the WN.

This can be done through the WMProxy job file perusal specific operations or through the *glite-wms-job-perusal* command.

The *PerusalFileEnable* attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

A default value for this attribute can be specified in the WMProxy client configuration file (*PerusalFileEnable* parameter in *glite_wms.conf*)
3.21. PERUSALTIMEINTERVAL

The PerusalTimeInterval attribute is a positive integer representing the difference in seconds between two subsequent saving (and upload on the WMS node) of the job files generated by the job on the WN. Specifying e.g.

```
PerusalTimeInterval = 1000;
```

makes the WMS JobWrapper save the job files specified through the --set option of the glite-wms-job-perusal command each 1000 seconds and upload them on the WMS node so that they can be inspected by the user.

If the JDL attribute is present, then a \( \text{max(jdl.PerusalTimeInterval, 1000 /* hard coded */)} \) is computed. If not, a \( \text{max(WmsConf.MinPerusalTimeInterval, 1000 /* hard coded */)} \) is calculated, where `WmsConf.MinPerusalTimeInterval` is the value of the `MinPerusalTimeInterval` parameter in the WM configuration file (default for `MinPerusalTimeInterval` is 5).

- **Mandatory:** No
- **Default:** `MinPerusalTimeInterval` parameter in the WMS configuration file (= 5)

3.22. INPUTDATA

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use `DataRequirements` attribute 3.25 instead.

The `InputData` attribute is a string or a list of strings representing Logical File Names (LFN), Grid Unique Identifiers (GUID), Logical Dataset (LDS) and/or generic queries. All of them are used by the WMS to query the related Data Catalog for getting back a list of Physical File names (PFN) that are needed by the job as input for processing. LFNs and LDSs have the form of a URI and a GUID is a 40 characters UUID (http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt). The format of generic queries is experiment/catalog specific.

All these data are stored in SEs and published in data catalogs.

As mentioned above the listed items are used by the WMS at matchmaking time to get the list of corresponding PFNs and then to find the CE having the greatest number of physical files on a close SE in order to schedule the job to run there.

Listed names have to be prefixed with "lfn:”, “guid:”, “lds” and “query:” to indicate that they are respectively LFNs, GUIDs, LDSs and generic queries. E.g.:

```
InputData = {
    "lfn:/EO/test.file",
    "si-lfn:/dm/test.file",
    "lds:cms.test.file",
    "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70",
    "query:select_my_DC1_files"
    "si-guid:456sde21-wer56-11d7-87e7-9d1-g8c8f33"
};
```
As the data could be published by different types of catalogs and the WMS currently supports three of them (i.e. the RLS, the StorageIndex and the DLI), the rule for choosing the catalog type to contact for resolving the file names according to the prefix is as follows:

- **lfn:, guid:** StorageIndex if the StorageIndex attribute (see 3.23) is also specified in the JDL. RLS otherwise and DLI upon failure on the RLS.
- **si-lfn:, si-guid:** StorageIndex
- **ldsl, query:** DLI

Wildcards are not admitted when specifying this attribute.

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

### 3.23. STORAGEINDEX

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use **DataRequirements** attribute 3.25 instead.

The **StorageIndex** attribute is a string representing the endpoint URI of the StorageIndex service to contact for resolving the file names specified in the InputData attribute list, e.g.

\[
\text{StorageIndex} = "https://data.glite.org:9443/StorageIndex";
\]

In case it is not specified and the InputData prefixes are **si-lfn:** or **si-guid:** then the VO default StorageIndex is considered and the endpoint URI is taken either from the configuration or through service discovery.

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

### 3.24. DATACATALOG

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use **DataRequirements** attribute 3.25 instead.

The **DataCatalog** attribute is a string representing the endpoint URI of the RLS or DLI service to contact for resolving the file names specified in the InputData attribute list, e.g.

\[
\text{DataCatalog} = "http://data.example.org/CMSDataSetCatalog";
\]

The specification of this attribute forces the WMS to resolve all LFNs and GUIDs in the Inputdata list that are not prefixed by **si-lfn:** or **si-guid:** using a "RLS" catalog and a "DLI" catalog upon failure.

- **Mandatory:** No
- **Default:** No
3.25. DATA REQUIREMENTS

The DataRequirements attribute is a list of classads representing the data requirements for a job. Each classad in the list contains three attributes (described in detail in the following sections 3.25.3, 3.25.1 and 3.25.2):

- *InputData*
- *DataCatalogType*
- *DataCatalog*

that represent respectively the list of input data needed by the job, the type of data catalog that has to be targeted to resolve logical names to physical names and lastly the URI of the data catalog if this is not the VO default one (endpoint known through service discovery or configuration).

The form of this attribute allows users to target experiment-specific catalogs for their jobs and to mix different input data types supported by different data catalogs in the same job description.

Note that it is possible to specify more than once the same catalog type e.g. for using the VO default one for some input data and a specific one for some other data.

Here below is reported an example of specification of the DataRequirements attribute:

```plaintext
DataRequirements = {
    [
        DataCatalogType = "DLI";
        DataCatalog = "https://cms.org:8877/dli";
        InputData = {
            "lfn:/my/test/data1",
            "guid:44rr44rr77hh77kkaa3",
            "lds:my.test.dataset",
            "query:my_query"};
    ],
    [
        DataCatalogType = "SI";
        DataCatalog = "https://glite.org:9443/StorageIndex";
        InputData = {
            "lfn:/eo/test.file",
            "guid:ddffrg5451"};
    ],
    [
        DataCatalogType = "RLS";
        DataCatalog = "https://eu-datagrid.org/RLS";
        InputData = {
            "lfn:/atlas/test.file",
            "guid:ggrgrg5656"};
    ],
    [
        DataCatalogType = "RLS";
        InputData = {
            "lfn:/myvo/test.file",
            "guid:adbdefgilm1234"};
    ],
    ....
};
```

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

### 3.25.1. DataCatalogType

The DataCatalogType attribute is a string representing the type of the data catalog to be targeted for resolving the input data logical names. Possible values, i.e. the data catalog interfaces currently supported by the WMS are:
RLS      LCG Replica Location Service  (lfn, guid)
SI       gLite Storage Index          (lfn, guid)
DLI      LCG Data Location Interface (lfn, guid, lds, query)

between parenthesis the corresponding supported data types.

- Mandatory: yes
- Default: No

3.25.2. DataCatalog
The DataCatalog attribute is a string representing the data catalog service endpoint URI to contact for resolving the file names specified in the InputData attribute list, e.g.

DataCatalog = "http://data.example.org/CMSDataSetCatalog";

It should be specified only if it is different from the VO default one.

- Mandatory: No
- Default: taken from WMS configuration or through service discovery

3.25.3. InputData
The InputData attribute is a string or a list of strings representing Logical File Names (LFN), Grid Unique ID-entifiers (GUID), Logical Dataset (LDS) and/or generic queries. All of them are used by the WMS to query the related Data Catalog for getting back a list of Physical File names (PFN) that are needed by the job as input for processing. LFNs and LDSs have the form of a URI and a GUID is a 40 characters UUID (http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt). The format of generic queries is experiment/catalog specific.

For more information please refer to [R3]. All these data are stored in SEs and published in data catalogs.

As mentioned above the listed items are used by the WMS at matchmaking time to get the list of corresponding PFNs and then to find the CE from which the specified files can be better accessed in order to schedule the job to run there.

Listed names have to be prefixed with “lfn:”, “guid:”, “lds” and “query:” to indicate that they are respectively LFNs, GUIDs, LDSs and generic queries. E.g.:

InputData = {
    "lfn:/EO.test.file",
    "lds:cms.test.file",
    "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70",
    "query:select_my_DC1_files"
};

As not all data catalogs support all types of input data, the specification of this attribute has to be cross-checked with the one of the DataCatalogType attribute described in section 3.25.1.
Wildcards are not admitted when specifying this attribute.

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

### 3.26. DATAACCESSPROTOCOL

The `DataAccessProtocol` attribute is a string or a 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 WMS matches indeed this attribute with the protocols supported by the SE, as published in the IS.

This is an example of the `DataAccessProtocol` attribute:

```plaintext
DataAccessProtocol = {
    "https",
    "gsiftp"
};
```

There is no restriction to the protocol names that can be specified through this attribute. Whether a given protocol is supported or not depends on the SE implementation. The list of SE access protocols supported in the Glue schema can be considered as the authoritative source for this information. Glue 1.2 supports `gsiftp`, `nfs`, `afs`, `rfio`, `gsirfio`, `dcap`, `gsidcap`, `root`, `https`, `other`.

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

### 3.27. OUTPUTSE

The `OutputSE` attribute 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 WMS to find a CE being “close” to this SE and schedule the job there. E.g.:

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

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

### 3.28. OUTPUTDATA

This attribute allows the user to ask for the automatic upload and registration to the Replica Catalog 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 classes composing OutputData. They cannot be specified independently in the job JDL.

Here below 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 DM 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, the DM service 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

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

### 3.28.1. 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
3.28.2. 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.

It is worth noting that this attribute is not taken into account by the WMS for the matchmaking, so the job could have run on a CE that is not close to the specified SE. Due to this it is suggested (unless the user has particular needs) either to omit the StorageElement specification so that the close SEs are automatically taken into account for the datasets upload or to keep its specification aligned with the one for the OutputSE attribute (see section 3.27) that is the one considered during the matchmaking.

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

3.28.3. 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 Data Management services.

- Mandatory: No
- Default: No (a GUID is assigned by DM services)
3.29. VIRTUALORGANISATION

The VirtualOrganisation attribute is a string representing the name of the VO the submitting user is currently working for.

The value for this attribute has to match with the VO specified within the credentials issued by VOMS or with the possible options provided by the WMProxy client commands (e.g. if the glite-wms-job-submit command is issued with the --vo option, then the value of this attribute is overwritten with the VO name specified on the command line). This is an example for this attribute:

VirtualOrganisation = “atlas”;

The value for this attribute is case insensitive.

- Mandatory: Yes
- Default: No

3.30. RETRYCOUNT

The RetryCount attribute is an integer representing the maximum number of deep job re-submissions to be done in case of failure due to some grid component (i.e. not to the job itself).

Job resubmission is defined “deep” when the user’s job has started running on the WN and then the job itself or the WMS JobWrapper has failed. It is instead defined “shallow” when the WMS JobWrapper has failed before starting the actual user’s job.

RetryCount has to be a number equal or greater 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 deep job re-submission mechanism it suffices specifying:

RetryCount = 0;

- Mandatory: No
- Default: 0, if no default is applied by the client

3.31. SHALLOWRETRYCOUNT

The ShallowRetryCount attribute is an integer representing the maximum number of shallow job re-submissions to be done in case of failure due to some grid component (i.e. not to the job itself).

Job resubmission is defined “deep” when the user’s job has started running on the WN and then the job itself or the WMS JobWrapper has failed. It is instead defined “shallow” when the WMS JobWrapper has failed before starting the actual user’s job.
ShallowRetryCount has to be a number equal or greater than 0 and the actual number of shallow submission retries for a job is represented by the minimum value between ShallowRetryCount itself and the value of the MaxShallowRetryCount parameter in the WM configuration file (default for MaxShallowRetryCount is 10). Specifying high values for the ShallowRetryCount is a recommended practice.

Note that the shallow retry count is reset to zero each time a deep resubmission of the job is done.

Hereafter follows an example for this attribute:

    ShallowRetryCount = 3;

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

    ShallowRetryCount = -1;

It is important to note the difference between setting the ShallowRetryCount attribute to –1 or to 0:

- -1 means that shallow resubmission is disabled and hence only deep resubmission is eventually considered
- 0 means that shallow resubmission is enabled but the maximum number of shallow resubmission is 0

As an example, consider the following JDL:

    [  
        Executable = "/bin/ls";  
        Prologue = "/bin/false";  
        ShallowRetryCount = x;  
        RetryCount = 2;  
        ...  
    ]

ShallowRetryCount set to –1 would imply that the job above is submitted once, resubmitted (deeply) twice, failing in all three cases and then aborted with reason “hit job retry count (2)”.
ShallowRetryCount set to 0 would imply that the job above is submitted only once and then aborted with reason “hit job shallow retry count (0)”.

- **Mandatory:** No
- **Default:** 0, if no default is applied by the client

### 3.32. LBADDRESS

The LBAddress attribute is a string representing the address (<host>[:<port>]) of the LB server where the WMS components have to store job information. E.g.:

    LBAddress = “tigerman.cnaf.infn.it:9000”;
If this attribute is not specified in the JDL the WMS gets the LB server address either querying the SD service or from the WMS configuration. If the port number is not specified, the WMS takes the default port, 9000.

- **Mandatory:** No
- **Default:** taken from WMS configuration

### 3.33. MYPROXYSERVER

The *MyProxyServer* attribute specifies the hostname of a MyProxy server where the user has registered her/his long-term proxy certificate.

A MyProxy server can be used to store a long-lived user certificate which can be used by the WMS to renew the lifetime of a standard user certificate proxy (usually valid only for 12 hours). Long-running jobs may run into this limit and fail due to expiration of the user proxy. To avoid this from happening, the user can store a certificate on a MyProxy server by issuing

```
$> myproxy-init -s <server hostname> -t <hours> -d -n
```

which will create a proxy valid for the specified number of hours on the given server (See [http://grid.ncsa.uiuc.edu/myproxy](http://grid.ncsa.uiuc.edu/myproxy) for more details about MyProxy). The very same hostname can be specified in the JDL MyProxyServer attribute, which will trigger the WMS proxy renewal process for this user’s jobs.

An example of the JDL setting is:

```
MyProxyServer = “skurut.cesnet.cz”;
```

If the port number is not specified, the WMS takes the default port, 7512.

In order to make sure that proxy renewal is disabled for a given job and no default MYProxy Server is taken from the configuration of your client tools, just set the *MyProxyServer* attribute to the empty string, i.e,

```
MyProxyServer = “”;
```

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

### 3.34. HLRLOCATION

The *HLRLocation* attribute is a string representing the user DGAS Home Location Register address in the format

```
<host fqdn>:<port>:[<X509contact string>]
```
HLR is the DGAS service responsible for managing the accounts of user and resources. The presence of the *HLRLocation* attribute in the JDL enables accounting for the job, i.e. 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 specified through the *HLRLocation* attribute.

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:** No

### 3.35. JOBPROVENANCE

The *JobProvenance* attribute is a string representing the endpoint URI of the Job Provenance service where data about the job have to be stored, e.g.

```
JobProvenance = "https://lindir.ics.muni.cz:10001";
```

The specification of this attribute in the job description makes the WMS feed the Job Provenance service with the job sandbox files.

This attribute is not mandatory. A default value for this attribute can be specified in the WMProxy client configuration file (*JobProvenance* parameter in glite_wms.conf).

In order to make sure that job provenance is disabled for a given job and no default JobProvenance Server is taken from the configuration of your client tools, just set the *JobProvenance* attribute to the empty string, i.e.,

```
JobProvenance = "";
```

- **Mandatory:** No
- **Default:** No (unless specified in the WMProxy client configuration)

### 3.36. NODENUMBER (DEPRECATED)

The *NodeNumber* attribute is an integer greater than 1 specifying the number of CPUs needed for a MPI job. This attribute is only allowed if the job type is MPICH (see 3.1). An example of the JDL setting is:

```
NodeNumber = 5;
```
The WMS 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:** No
- **Default:** No

### 3.37. SMPGRANULARITY

3.38.
The SMPGranularity attribute is an integer greater than 0 specifying the number of cores any host involved in the allocation has to dedicate to the considered job.

This attribute can’t be specified along with the Hostnumber attribute when WholeNodes is false.

This attribute is supported with CREAM CE v. >= 1.7 (glite-ce-cream v. >= 1.13)

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

### 3.39. WHOLENODES

3.40.
The WholeNodes attribute is a boolean that indicates whether whole nodes should be used exclusively or not.

This attribute is supported with CREAM CE v. >= 1.7 (glite-ce-cream v. >= 1.13)

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

### 3.41. HOSTNUMBER

3.42.
HostNumber is an integer indicating the number of nodes the user wishes to obtain for his job.

This attribute can’t be specified along with the SMPGranularity attribute when WholeNodes is false.
This attribute is supported with CREAM CE v. >= 1.7 (glite-ce-cream v. >= 1.13)

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

### 3.43. CPUNUMBER

The **CPUNumber** attribute is an integer greater than 1 specifying the number of CPUs needed for a MPI job. This attribute is only allowed if the job type is Normal. It must be used in place of the deprecated **NodeNumber** attribute (see 3.36) An example of the JDL setting is:

```
CPUNumber = 5;
```

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

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

### 3.44. REQUIREMENTS

The **Requirements** attribute is a Boolean ClassAd expression that uses C-like operators. It represents job requirements on resources. The Requirements expression can contain attributes that describe the CE in the IS which are prefixed with “other.”. This notation indicates at matchmaking time that for a given classAd (the job’s JDL) the expression has to be evaluated in the context of the counterpart classAd (the CE’s JDL).

All these attributes are reported in the Glue Schema for the CE.

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 WMS during the match making phase. This is an 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 classAd Member function described in [R7] returns true if the tag "IDL1.7" is a member of the other.GlueHostApplicationSoftwareRunTimeEnvironment list.

The **Requirements** attribute is mandatory in the JDL as it is a mandatory parameter for the matchmaking library.

If this attribute is not included in the JDL the **glite-wms-job-submit** command sets it to:

```
Requirements = other.GlueCEStateStatus == "Production";
```
as “Production” is the nominal working state for a CE. This automatic setting is however not insured if submission is performed using client tools other than `glite-wms-job-submit`.

Requirements can be set to `true` to express no constraints on the resources.

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

### 3.45. RANK

The `Rank` attribute 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 WMS will submit the job to the CE with the highest rank.

The `Rank` expression can contain attributes that describe the CE in the IS which are prefixed with “`other`.”. This notation indicates at matchmaking time that for a given classAd (the job’s JDL) the expression has to be evaluated in the context of the counterpart classAd (the CE’s JDL). See [R7] for details.

All these attributes are reported in the Glue Schema for the CE.

The evaluation of the rank expression is performed by the WMS during the match making phase. This is an example of Rank expression:

\[
\text{Rank} = \text{other.GlueCEPolicyMaxRunningJobs} - \text{other.GlueCEStateRunningJobs};
\]

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

The `Rank` attribute is mandatory in the JDL as it is a mandatory parameter for the matchmaking library.

If this attribute is not included in the JDL the `glite-wms-job-submit` command sets it to:

\[
\text{Rank} = -\text{other.GlueCEStateEstimatedResponseTime};
\]

that expresses a preference for those CEs having a shorter estimated time for traversing the local batch system queue. This automatic setting is however not insured if submission is performed using client tools other than `glite-wms-job-submit`.

Setting the Rank expression to a constant value (e.g. `Rank = 1;`) makes all CEs matching job’s requirements have the same ranking. In this case the CE for submitting the job is chose randomly by the WMS.

- **Mandatory:** Yes
- **Default:** No
3.46. FUZZYRANK

The FuzzyRank attribute is a Boolean attribute that enables fuzzyness in the ranking computation. In other words if this attribute is set to true, it 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. The selection probability is higher for higher rank values.

- Mandatory: No
- Default: false

3.47. USERTAGS

The UserTags attribute is a classad attribute that allows the user to specify user-defined key, value pairs (where the value must be a string) that are logged at submission time to the LB and are associated to the job in the LB database. The specified user tags can be then used to build conditions when querying the LB for the status of submitted jobs (see e.g. the --user-tag option of the glite-job-status command). The value of a user tag can be modified or a new key, value pair can be added after the job has been submitted using the LB producer API (see method edg_wll_LogUserTag) that is provided by the org.glite.lb.client component.

This is an example of user tags setting in the JDL:

UserTags = [
    color = "red";
    position = "12";
    prodId = "cms_1234";
];

- Mandatory: No
- Default: No

3.48. SHORTDEADLINEJOB

The ShortDeadlineJob is a boolean attribute that allows the user to enable the support for Short Deadline Jobs (SDJ). When this attribute is set to true, i.e.:

ShortDeadlineJob = true;

The WMProxy will append a condition on the Requirements of the JDL itself:

Requirements = ( [...] ) && RegExp("*sdj$", other.GlueCEUniqueID);

this expression allows the job to be sent to a SDJ dedicated Computing Element.

Please note that the value of the condition depends on the WMProxy configuration file.
When the ShortDeadlineJob is set to false (i.e. the default behaviour), the WMProxy will append a condition on the Requirements; this will avoid non-SDJ jobs to be executed on SDJ dedicated Computing Elements:

```
Requirements = ( [...] ) && !RegExp("*sdj$", other.GlueCEUniqueID);
```

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

### 3.49. ENABLEWMSFEEDBACK

Enables the WMS's feedback mechanism. This mechanism avoids a job to remain stuck for a long time in queue waiting to be assigned to a worker node for execution. All jobs in Scheduled state for more than a given predetermined time (REPLANGRACEPERIOD in the WMS configuration, see WMS admin guide) will be removed from their current queue and re-matched.

This mechanism only work with the CREAM CE, setting this parameter to true will automatically append in the requirements expression the checks to consider CREAM CEs only.

Each replan request is formally equivalent to a shallow resubmission, so this feature must have the shallow resubmission enabled to work. The maximum number of possible replans will then be determined by the value of ShallowRetryCount (which of course also accounts for other kinds of resubmission).

The feedback is a mechanism that requires the existence of a global synchronization token, as said this is the shallow resubmission token. For this reason, it must not be used with the deep resubmission enabled. To work properly, in fact, the feedback mandates that one and only one instance of a given job is running on the Grid.

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

**EXAMPLE:**

In the WMS configuration (section [WorkloadManager]):

```
ReplanGracePeriod = 1800; // seconds to wait for a job in Scheduled state before moving it to another queue
MaxReplansCount = 4; // max number of trials
```

**JDL:**

```
[ // usual job attributes
  EnableWmsFeedback = true;
  ShallowRetryCount = 5;
  // deep resubmission must be disabled
  RetryCount = -1;
];
```
4. DAG ATTRIBUTES DESCRIPTION

A DAG (directed acyclic graph) represents a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs. The jobs are nodes (vertices) in the graph and the edges (arcs) identify the dependencies.

Although not requiring a great number of new attributes, the structure of the JDL description for a DAG differs significantly from the one for a job and it is for sure more complex. Due to this we start by providing an example of DAG description so that the structure of the JDL in this case is clear from the beginning. The new attributes and the slight differences in the semantic for some of the known ones will be highlighted in the text.

It is important to note that upon submission of a DAG, besides the identifier associated with each node, the WMS assigns also to the DAG itself an identifier that has to be used as the handle for monitoring and controlling the whole DAG.

Let's consider the DAG in next Example 1:

Example 1 - DAG example

It can be described by the following JDL:

```plaintext
[ 
  Type = "dag";
  VirtualOrganisation = "EGEE";
  MyProxyServer = "skurut.cesnet.cz";
  HLRLocation =
    "eth.to.infn.it:5562:/O=CESNET/O=INFN To/CN=Andrea Guarise";
  InputSandbox = {
    "/tmp/foo/*.exe",
    "/home/gliteuser/bar",
    "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe",
  },
]```
"file:///tmp/myconf"
};
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
Rank = - other.GlueHostEstimatedTraversalTime;
Requirements = other.GlueCEStateStatus == "Production";
max_nodes_running = 5;
nodes = [
  nodeA = [
    description = [
      JobType = "Normal";
      Executable = "a.exe";
      InputSandbox = {
        "/home/data/myfile.txt",
        root.InputSandbox
      }
    ];
  ];
  mynode = [
    description = [
      JobType = "Normal";
      Executable = "b.exe";
      Arguments = "1 2 3";
      RetryCount = 3;
      Requirements =
        other.GlueCEInfoTotalCPUs > 2;
      Rank = other.GlueCEStateFreeCPUs;
      OutputSandbox = {"myoutput.txt",
                       "myerror.txt"};
      OutputSandboxDestURI =
        "gsiftp://neo.datamat.it:5432/tmp";
    ];
  ];
  nodeD = [
    description = [
      JobType = "Checkpointable";
      JobType = "Normal";
      Executable = "b.exe";
      Arguments = "1 2 3";
      RetryCount = 3;
      InputSandbox = {
        "file:///home/pippo",
        root.nodes.mynode.description.OutputSandbox[0]
      }
    ];
  ];
  nodeC = [
Example 2 - JDL representation for a DAG

As shown above the structure of a DAG description consists of a certain number of nested classads in the following hierarchy:

- The root classad containing the whole description with some attributes that are associated to the DAG as a whole since they are in some sense inherited by the nodes of the DAG (this mechanism will be explained in the following).
- a children classads named Nodes containing the description of all the nodes of the DAG and at the same level the dependencies attribute specifying all dependencies among the DAG sub-jobs.
- A set of classads (whose names are free) inside Nodes each one corresponding to a node and containing its description

The detailed description of the mentioned attributes is provided hereafter:

### 4.1. TYPE

See 2.1.

In this case the value for this attribute is “DAG”.

- Mandatory: Yes

### 4.2. VIRTUALORGANISATION

See 3.29.

The Virtual Organisation must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the VirtualOrganisation attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: Yes
- Default: No
4.3. **MAX_RUNNING_NODES**

The `max_running_nodes` attribute is an integer greater than zero representing the maximum number of
nodes of a DAG that can be submitted by DAGMan at a given time. E.g.

\[ \text{max_running_nodes} = 25; \]

This means that if at a certain time there are \( N \) nodes of the DAG that are free of dependencies, only
`max_running_nodes` of them can be submitted simultaneously whilst the remaining ones will be
submitted as soon as the already submitted one complete their execution.

This attribute is not mandatory and defaults to 10.

- **Mandatory**: No
- **Default**: 10

4.4. **HLRLOCATION**

See 3.34.

The HLR must be the same for the DAG and all its nodes. The value of this attribute is hence inherited
by all nodes descriptions. If a node already contains the `HLRLocation` attribute in its description, the
value of the attribute is overridden by the one specified for the DAG (if different).

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

4.5. **LBADDRESS**

See 3.31

The LB Server address must be the same for the DAG and all its nodes. The value of this attribute is hence inherited
by all nodes descriptions. If a node already contains the `LBAddress` attribute in its description, the
value of the attribute is overridden by the one specified for the DAG (if different).

- **Mandatory**: No
- **Default**: taken from WMS configuration

4.6. **MYPROXYSERVER**

See 3.33

The MyProxy Server must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the `MyProxyServer` attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

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

4.7. **JOBPROVENANCE**

See 3.35
The Job Provenance Server must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the JobProvenance attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

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

### 4.8. ALLOWZIPPEDISB

See 3.16.

For DAG requests this attribute is only take into account at this level. If the AllowZippedISB attribute is set to true, a single compressed archive is created for the input sandbox files of all DAG nodes.

The AllowZippedISB attributes (if any) specified within the nodes description are ignored by WMS.

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

### 4.9. ZIPPEDISB

See 3.17.

For DAG requests the archive file indicated through this attribute should contain the input sandbox files of all DAG nodes.

For DAG requests this attribute is only take into account at this level; the ZippedISB attributes (if any) specified within the nodes description are ignored by WMS.

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

### 4.10. EXPIRYTIME

See 3.18.

The value of this attribute is not considered for the DAG itself. All nodes that do not contain the ExpiryTime attribute in their descriptions inherit the value of this attribute from the one specified for the DAG.

- **Mandatory:** No
- **Default:** one day after submission time.

### 4.11. PERUSALFILEENABLE

See 3.20.

All nodes that do not contain the PerusalFileEnable attribute in their descriptions inherit the value of this attribute from the one specified for the DAG.

The PerusalFileEnable attribute is not mandatory.

- **Mandatory:** No
- **Default:** false
4.12. USERTAGS

See 3.47.

The key,value pairs specified for the DAG are only applied to the DAG as a whole and not to the nodes that can have their own tags specified within their descriptions.

- Mandatory: No
- Default: No

4.13. REQUIREMENTS AND RANK

See 3.44 and 3.45.

All nodes that do not contain the Requirements and/or Rank expressions in their descriptions inherit the value of these attributes from the one specified for the DAG. E.g. in Example 2, nodeD inherits Requirements and Rank from the DAG whilst mynode doesn’t.

- Mandatory: No
- Default: No

4.14. INPUTSANDBOX AND INPUTSANDBOXBASEURI

See 3.7 and 3.8.

All nodes that do not contain the InputSandbox and/or the InputSandboxBaseURI attributes in their descriptions inherit the value of these attributes from the one specified for the DAG. This rules applies also to the single attribute, i.e. a node that only contains the InputSandbox in its description, inherits the InputSandboxBaseURI (this is the case for nodeD in Example 2) from the DAG and vice-versa. Note however that the InputSandboxBaseURI is not applied to files already specified as URI or prefixed with the “file:” URI scheme. Nodes representing jobs without input sandbox have to contain the following specification in their description (empty InputSandbox list):

```
InputSandbox = {};
```

The inheritance of the InputSandbox attribute allows the introduction of the concept of the “shared sandbox”, i.e. a sandbox that is common to multiple jobs (some of the nodes of the DAG) and that needs to be transferred on the WMS node only once.

As it can be seen in Example 2 it is possible within the description of a node, to make a reference to an attribute either of the DAG or of another node. E.g. in nodeA the InputSandbox includes the input sandbox of the DAG, i.e. the resulting sandbox for the node is:

```
InputSandbox = {
  "/home/data/myfile.txt",
  "/tmp/foo",
  "/home/gliteuser/bar",
  "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
  "file:///tmp/myconf"
};
```

Another example of attribute reference is the InputSandbox of nodeD that refers to a file in the output sandbox of the node mynode. The resulting InputSandbox list for nodeD is:

```
InputSandbox = {
  "/home/data/myfile.txt",
  "/tmp/foo",
  "/home/gliteuser/bar",
  "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
  "file:///tmp/myconf"
};
```
Both attributes described in this section are:

- Mandatory: No
- Default: No

### 4.15. OUTPUTSANDBOXBASEDESTURI
See 3.11.
All nodes that contain neither the OutputSandboxDestURI nor the OutputSandboxBaseDestURI attributes in their descriptions inherit the value of OutputSandboxBaseDestURI from the one specified for the DAG.

- Mandatory: No
- Default: No

### 4.16. DEFAULTNODERETRYCOUNT
The DefaultNodeRetryCount attribute specifies value of the RetryCount attribute (See 3.30.) to be applied to all nodes of the DAG not specifying their own RetryCount.

- Mandatory: No
- Default: 0, if no default is applied by the client

### 4.17. DEFAULTNODESHALLOWRETRYCOUNT
The DefaultNodeShallowRetryCount attribute specifies value of the ShallowRetryCount attribute (See 3.31.) to be applied to all nodes of the DAG not specifying their own ShallowRetryCount.

- Mandatory: No
- Default: 0, if no default is applied by the client

### 4.18. NODES
The Nodes attribute is the core of the DAG description and it is used for specifying the nodes and their dependencies. It is a classad containing all the nodes of the DAG and the attribute named Dependencies. Each node corresponds to a classad attribute (nodeA, nodeB, nodeC, nodeD and mynode in the previous Example 2), whose name can be freely set by the user, which contains the description of the job representing the node. Such description can be provided through the following attributes:

#### 4.18.1. File
The File attribute is a string representing the absolute path (or relative to the home of the user) on the local file system to a file containing the JDL description of a Job. It is important to note that this kind
of representation can only be used when submitting to the WMS through a client (like e.g. `glite-wms-job-submit` (R15)) able to resolve the path locally and to expand the JDL with the full description before passing it to the WMS.

The job description provided within the file pointed by `File` is subjected to all rules reported in section 3. The only limitation that applies is on the `JobType` allowed for a DAG node that, as already reported in section 2, can’t be `Parametric`.

The `File` attribute cannot be specified together with the `Description` attribute (see 4.18.2) within the same node description.

- **Mandatory:** No (provided `Description` is specified)
- **Default:** No

### 4.18.2. Description

The `Description` attribute is a classad representing the JDL description of the job representing the node of the DAG.

The job description provided by this attribute is subjected to all rules reported in section 3. The only limitation that applies is on the `JobType` allowed for a DAG node that, as already reported in section 2, can’t be `Parametric`.

The `Description` attribute cannot be specified together with the `File` attribute (see 4.18.1) within the same node description.

- **Mandatory:** No (provided `File` is specified)
- **Default:** No

### 4.19. DEPENDENCIES

The `Dependencies` attribute is a list of lists representing the dependencies between the nodes of the DAG described in the `nodes` attribute. Each sub-list specifies dependencies between a sub set of nodes.

The `dependencies` specification (taking e.g. the one in Example 2) must be read as follows:

```plaintext
{ nodeA, nodeB } means that nodeB cannot start before nodeA has completed its execution successfully

{ nodeA, nodeC } means that nodeC cannot start before nodeA has completed its execution successfully

{ nodeA, mynode } means that mynode cannot start before nodeA has completed its execution successfully

{ { nodeB, nodeC, mynode }, nodeD } means that nodeD cannot start before nodeB, nodeC and mynode have completed their execution successfully
```

The form `{ { nodeB, nodeC, mynode }, nodeD }` is equivalent to:

```plaintext
{ nodeB, nodeD },
```
{ nodeC, nodeD },
{ mynode, nodeD }

and it is also equivalent to:

{ { nodeB, nodeC, mynode }, { nodeD } }

The description of a pipeline of jobs could be accomplished by specifying the following dependencies:

{ { nodeA, nodeB}, {nodeB, nodeC}, {nodeC, nodeD }, {nodeD, nodeE} … }

The dependencies attribute is mandatory for a DAG description although it can be an empty list (i.e. dependencies = {};) and can also be specified within the nodes classad.

- Mandatory: Yes
- Default: No

4.20. OUTPUTSANDBOX

The JDL description of a DAG must not contain any OutputSandbox attribute occurrence besides the ones in the descriptions of the nodes. The OutputSandbox of the DAG (i.e. the files retrieved e.g. using the glite-wms-job-output command [R15]) has indeed to be considered as the sum of the output sandboxes of all its nodes.
5. PARAMETRIC JOB ATTRIBUTES DESCRIPTION

We describe here in detail the JDL for a parametric job.

A **Parametric** job is a job having one or more parametric attributes in the JDL. The parametric attributes vary their values according to another attribute (named **Parameters**) also specified in the job description. The submission of a **Parametric** job results in the submission of a set of jobs having the same descriptions apart from the values of the parametric attributes. Both the parametric job and all jobs resulting from the submission of it are assigned by the WMS with an identifier so that it is possible to monitor and control each of them separately and as a single entity (through the parametric job handle).

To clarify the concept, let us consider the following **Parametric** job example:

```
[  
JobType = "Parametric";
Executable = "cms_sim.exe";
StdInput = "input_PARAM_.txt";
StdOutput = "myoutput_PARAM_.txt";
StdError = "myerror_PARAM_.txt";
Parameters = 10000;
ParameterStart = 1000;
ParameterStep = 10;
InputSandbox = {  
   "file:///home/cms/cms_sim.exe",
   "file:///home/cms/data/input_PARAM_.txt "
  };
OutputSandbox = {  
   "myoutput_PARAM_.txt",
   "myerror_PARAM_.txt"
  };
OutputSandboxDestURI = "gsiftp://neo.datamat.it:5432/tmp";
Requirements = other.GlueCEInfoTotalCPUs > 2;
Rank = other.GlueCEStateFreeCPUs;
]
```

**Example 3 – JDL representation for a Parametric Job**

As shown in the above Example 3, besides the attributes available for *normal* jobs (described in section 3), the JDL for a parametric job contains three additional specific attributes:

- **Parameters**
- **ParameterStart**
- **ParameterStep**

which represent respectively the set of values the parametric attributes should assume, the starting value for the variation (if the parameter is a number) and the step for each variation. Moreover the parametric attributes in the JDL, i.e. the ones whose values should vary, contain the "_PARAM_" instruction within their values.

The submission of the JDL in Example 3 will result in the generation of N jobs, where
\[ N = \frac{(\text{Parameters} - \text{ParameterStart})}{\text{ParameterStep}} \]

each one containing the same JDL but the parametric attributes for which the “\_PARAM\_” instruction is replaced with the actual value of the parameter, e.g. for the \textit{StdInput} attribute of Example 3 it would be respectively in the resulting jobs:

- \textit{StdInput} = "input1000.txt";
- \textit{StdInput} = "input1010.txt";
- \textit{StdInput} = "input1020.txt";
- ...  
- \textit{StdInput} = "input9990.txt";

The applied value of the parameter is exported to the job run time environment through the variable \textit{ParameterValue} set by the WMS \textit{JobWrapper} script before the job is started. E.g. the job related to the first bullet above would find the environment variable \textit{ParameterValue} set equal to 1000 when running. Moreover each sub-job that is generated by the submission of a parametric job is assigned by the WMS with the “node\_<P>” name where \( P \) is the value of the parameter for that given instance of the job (this can be seen as \textit{Node Name} field of the \texttt{glite-job-status} command output [R16]).

In the job description of Example 3 the \textit{Parameters} attribute is specified as a number to represent (coupled with the \textit{ParameterStart} attribute value, if any) a range for the parameterisation (note that the value of \textit{Parameters} is not included in the range, i.e. values vary form \textit{ParameterStart} to \textit{Parameters-1}). Alternatively the \textit{Parameters} attribute can be specified as a list of values that will be used for assigning values to the parametric attributes. E.g. a JDL as follows:

```
[
  ...
  Executable = "my_sim.exe";
  Arguments = "\_PARAM\_";
  Parameters = {alpha, beta, gamma};
  ...
]
```

results in the submission of three jobs only differing for the value of the argument (respectively “alpha”, “beta” and “gamma”) passed to the executable (see also 8.6). Note that when \textit{Parameters} is specified as a list of values, it does not make sense anymore to specify the \textit{ParameterStart} and \textit{ParameterStep} attributes.

Here below is provided the detailed description of each of the mentioned attributes.

### 5.1. PARAMETERS

The \textit{Parameters} attribute is either an integer representing the upper bound (not included) or the lower bound, in case it is negative, for the values that can be assumed by the parametric attributes (i.e. the ones containing the “\_PARAM\_” instruction), e.g.

\[ \text{Parameters} = 100000; \]

or a list of items representing the values on which the parametric attributes have to range, e.g.
Parameters = {“raw”, “d0”, “d1”, “d2”};

It is important to note that when the Parameters attribute is specified as a list, the values of the list must not have a “type” (e.g. have not to be enclosed within quotes if they are strings) as typing will be automatically enforced according to the type of the parametric attributes. Furthermore note that literals items must be double-quoted.

Obviously this attribute can only be set for parametric jobs.

− Mandatory: Yes
− Default: No

### 5.2. PARAMETERSTART

The ParameterStart attribute is an integer indicating the initial value to take into account for the parameter scanning of a Parametric job. E.g.

Parameters = 100000;
ParameterStart = 15000;

means that the parametric attributes values will range starting from 15000 up to 985000.

This attribute can only be set for a parametric job if the Parameters attribute has been specified as a number. If not provided by the user it is assumed to be 0.

− Mandatory: No
− Default: 0

### 5.3. PARAMETERSTEP

The ParameterStep attribute is an integer representing the size of each variation of the parameter (i.e. the difference between two subsequent values of the parameter) when ranging between the values of ParameterStart and Parameters. E.g.

Parameters = 100000;
ParameterStart = 15000;
ParameterStep = 1000;

means that for each generated job the value of the parameter will be increased of 1000 in the interval [15000, 100000).

This attribute can only be set for a parametric job if the Parameters attribute has been specified as a number. If not provided by the user it is assumed to be 1.

− Mandatory: No
− Default: 1
6. JOBS COLLECTION ATTRIBUTES DESCRIPTION

A job Collection is a set of independent jobs that for some reasons (known to the user) have to be submitted, monitored and controlled as a single request. As it happens for a DAG, upon submission, besides the identifiers for the sub-jobs, the collection is associated with a jobId that can be used as the unique handle for the whole set of jobs.

The JDL description for a Collection is quite simple as it basically consists of a list of classads (the sub-jobs) plus some attributes that analogously to the DAG are associated to the Collection as a whole.

Here below is reported an example of a job Collection description that we will use as a basis for providing further details:

```json
[
    Type = "collection";
    VirtualOrganisation = "EGEE";
    MyProxyServer = "skurut.cesnet.cz";
    HLRLocation = "eth.to.infn.it:5562:/O=CESNET/O=INFN To/CN=Andrea Guarise";
    InputSandbox = {
        "/tmp/foo",
        "/home/gliteuser/bar",
        "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
        "file:///tmp/myconf"
    },
    InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp"
    Rank = other.GlueHostEstimatedTraversalTime;
    Requirements = other.GlueCEStateStatus == "Production";
    nodes = {
        [
            JobType = "Normal"
            Executable = "a.exe"
            InputSandbox = {
                "/home/data/myfile.txt",
                root.InputSandbox
            }
        ],
        [
            JobType = "Normal"
            Executable = "b.exe"
            Arguments = "1 2 3"
            RetryCount = 3
            Requirements = other.GlueCEInfoTotalCPUs > 2;
            Rank = other.GlueCEStateFreeCPUs;
            OutputSandbox = {"myoutput.txt",
                "myerror.txt" }
            OutputSandboxDestURI = "gsiftp://neo.datamat.it:5432/tmp";
        ]
    }
]
Example 4 – JDL representation for a Collection

It can be easily noted that the structure of the JDL for a collections differs from the one for a DAG. The nodes attribute is indeed a list containing plain classads with no further nesting (i.e. the description attribute doesn’t exist for a collection). Moreover as the jobs of a collection are supposed to be independent, the dependencies attribute cannot be included in the collection description. Apart from this and the fact that the Type attribute has to be “Collection” in this case, all the attributes described for a DAG (see section ) preserve the same meaning for a collection and the same “inheritance” rules apply.

The attributes of a collection are described in detail in the following sub-sections.

6.1. TYPE

See 2.1.

In this case the value for this attribute is “Collection”.

– Mandatory: Yes

6.2. VIRTUALORGANISATION

See 3.29.

The Virtual Organisation must be the same for the Collection and all its jobs. The value of this attribute is hence inherited by all jobs descriptions. If a job already contains the VirtualOrganisation attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

– Mandatory: Yes
– Default: No
6.3. HLRLOCATION
See 3.34.
The HLR must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the HLRLocation attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).
  – Mandatory: No
  – Default: No

6.4. LBADDRESS
See 3.31.
The LB Server address must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the LBAddress attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).
  – Mandatory: No
  – Default: taken from WMS configuration

6.5. MYPROXYSERVER
See 3.33.
The MYProxy Server must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the MyProxyServer attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).
  – Mandatory: No
  – Default: No

6.6. JOBPROVENANCE
See 3.35.
The Job Provenance Server must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the JobProvenance attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).
  – Mandatory: No
  – Default: No

6.7. ALLOWZIPPEDISB
See 3.16.
For Collection requests this attribute is only take into account at this level. If the AllowZippedISB attribute is set to true, a single compressed archive is created for the input sandbox files of all Collection sub-jobs.
The AllowZippedISB attributes (if any) specified within the sub-jobs description are ignored by WMS.
  – Mandatory: No
6.8. ZIPPEDISB

See 3.17.
For Collection requests the archive file indicated through this attribute contains the input sandbox files of all collection sub-jobs.
For Collection requests this attribute is only take into account at this level; the ZippedISB attributes (if any) specified within the sub-jobs description are ignored by WMS.

- Mandatory: No
- Default: false

6.9. EXPIRYTIME

See 3.18.
The value of this attribute is not considered for the Collection itself. All sub-jobs that do not contain the ExpiryTime attribute in their descriptions inherit the value of this attribute from the one specified for the Collection.

- Mandatory: No
- Default: one day after submission time.

6.10. PERUSALFILEENABLE

See 3.20.
All nodes that do not contain the PerusalFileEnable attribute in their descriptions inherit the value of this attribute from the one specified for the Collection.
The PerusalFileEnable attribute is not mandatory.

- Mandatory: No
- Default: false

6.11. USERTAGS

See 3.47.
The key,value pairs specified for the Collection are only applied to the Collection as a whole and not to the sub-jobs that can have their own tags specified within their descriptions.

- Mandatory: No
- Default: No

6.12. REQUIREMENTS AND RANK

See 3.44 and 3.45.
All jobs that do not contain the Requirements and/or Rank expressions in their descriptions inherit the value of these attributes from the one specified for the Collection. E.g. in Example 4, the first and the third jobs in the nodes list inherit Requirements and Rank from the Collection whilst the second job doesn’t.
6.13. INPUTSANDBOX AND INPUTSANDBOXBASEURI

See 3.7 and 3.8.

All jobs that do not contain the InputSandbox and/or the InputSandboxBaseURI attributes in their descriptions inherit the value of these attributes from the one specified for the collection. This rule also applies to the single attribute, i.e. a job that only contains the InputSandbox in its description, inherits the InputSandboxBaseURI (this is the case for the third job in the nodes list in Example 4) from the collection. The same happens if only InputSandboxBaseURI is specified in the sub-jobs descriptions.

Note however that the InputSandboxBaseURI is not applied to files already specified as URI or prefixed with the “file://” URI scheme. Jobs of the collection without input sandbox have to contain the following specification in their description (empty InputSandbox list):

    InputSandbox = {};

so that the attribute is not inherited from the collection.

The inheritance of the InputSandbox attribute allows the introduction of the concept of the “shared sandbox”, i.e. a sandbox that is common to multiple jobs of the collection and that needs to be transferred on the WMS node only once.

As it can be seen in Example 4 it is possible within the description of a node, to make a reference to an attribute either of the collection or of another node. E.g. in the first job of the nodes list the InputSandbox attribute includes the input sandbox of the collection, i.e. the resulting sandbox for the node is:

    InputSandbox = {
        "/home/data/myfile.txt",
        "/tmp/foo",
        "/home/gliteuser/bar",
        "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe",
        "file:///tmp/myconf"
    };

Another example of attribute reference is the InputSandbox attribute of the third job of the nodes list that refers to the first file in the output sandbox of the second job of the nodes list (although this kind of reference is discouraged as the jobs of a collection are intended as independent and hence the user cannot assume that the output sandbox of a given job is ready before another job is started; DAGs have to be used at this purpose). The resulting InputSandbox list for the third job of the nodes list is:

    InputSandbox = {
        "file:///home/pippo",
        "gsiftp://neo.datamat.it:5432/tmp/myoutput.txt"
    };

Both attributes described in this section are:
6.14. OUTPUTSANDBOXBASEDESTURI

See 3.11.

All sub-jobs that contain neither the OutputSandboxDestURI nor the OutputSandboxBaseDestURI attributes in their descriptions inherit the value of OutputSandboxBaseDestURI from the one specified for the Collection.

- Mandatory: No
- Default: No

6.15. DEFAULTNODERETRYCOUNT

The DefaultNodeRetryCount attribute specifies value of the RetryCount attribute (See 3.30.) to be applied to all sub-jobs of the collection not specifying their own RetryCount.

- Mandatory: No
- Default: 0, if no default is applied by the client

6.16. DEFAULTNODESHALLOWRETRYCOUNT

The DefaultNodeShallowRetryCount attribute specifies value of the ShallowRetryCount attribute (See 3.31.) to be applied to all sub-jobs of the collection not specifying their own ShallowRetryCount.

- Mandatory: No
- Default: 0, if no default is applied by the client

6.17. NODES

The Nodes attribute is the core of the Collection description and it is used for specifying the jobs the collection is made of. It is a list containing all the classads describing the jobs.

The job descriptions provided in the nodes list are subjected to all rules reported in section 3.

The description of a node of the collection can be either provided directly in the classad (like e.g. the first job in the nodes list of Example 4) or through a pointer to a JDL file (like e.g. the last job in the nodes list of Example 4) by using the File attribute. A collection job can be assigned with a name by means of the NodeName attribute as it is done e.g. in the third job of the nodes list of Example 3. This is very useful when monitoring the collection as the user can easily associate the given names to the automatically assigned job identifiers.

The above-mentioned attributes are described here below:

6.17.1. File

The File attribute is a string representing the absolute path (or relative to the current working directory) on the local file system to a file containing the JDL description of a Job. It is important to note that this kind of representation can only be used when submitting to the WMS through a client (like e.g. glite-wms-job-submit [R15]) able to resolve the path locally and to expand the JDL with the full description before passing it to the WMS.
The job description provided within the file pointed by the *File* attribute is subjected to all rules reported in section 3.

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

### 6.17.2. **NodeName**

The *NodeName* attribute is a string representing the name that the user wants to associate to the job of a collection so that the job can be easily identified at job monitoring time (see e.g. the specification of the third job of the *nodes* list of Example 4).

For those jobs of a collection whose description does not contain this attribute, the *NodeName* is automatically assigned by the WMS and set equal to:

```
NodeName = "node<N>";
```

Where N is a number starting from 0 that indicates the position of the job classad in the *nodes* list. E.g. in the collection of Example 4, assuming that the description contained in `/home/test/c.jdl` does not specify the *NodeName* attribute, the job names would be: node0, node1, mysubjob, node3.

Note that when the *File* attribute is used in the collection description, then the *NodeName* can also be specified at the same level as the *File* attribute.

Note moreover that due to the attributes naming rules imposed by the Condor classad library, the *NodeName* string attribute cannot start with a number.

- **Mandatory:** No
- **Default:** node<N> assigned by WMS

### 6.18. **OUTPUTSANDBOX**

The JDL description of a Collection must not contain any *OutputSandbox* attribute occurrence besides the ones in the descriptions of the jobs of the collection. The *OutputSandbox* of the Collection (i.e. the files retrieved e.g. using the *glite-wms-job-output* command [R15]) must indeed to be considered as the sum of the output sandboxes of all its jobs.
7. 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.

7.1. 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 WMS 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 WMS 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 close 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 accommodated 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:

\[
\text{Requirements} = \text{anyMatch(} \text{other.storage.CloseSEs, target.GlueSAStateAvailableSpace > 200);}
\]

This makes indeed the WMS 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].
8. JDL EXAMPLES

In the following sections are reported simple examples of JDL describing different types of jobs and requests.

8.1. NORMAL JOB

```
JobType = "normal";
Executable = "jobExecutor";
Arguments = "1 100";
Environment = {
    "PATH=$PATH:/usr/local/bin",
    "LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib"};
InputSandbox = {"/home/glite/scripts/jobExecutor ",
                
    "/home/glite/scripts/envcheck.sh"};
StdOutput = "je.out";
StdError = "je.err";
OutputSandbox = {"je.out","je.err","je-1-100.out"};
OutputSandboxDestURI = {"je.out","je.err",
                        
    "https://trinity.datamat.it:7443/run/je-1-100.out"};
Rank = -other.GlueCEStateEstimatedResponseTime;
Requirements = RegExp(".*lal.in2p3.fr.*",
                    
    other.GlueCEUniqueID) &&
                    
    (other.GlueCEPolicyMaxCPUTime >= 100);
// If no resource is available retry matchmaking until 2006-05-26 18:30:00
ExpiryTime = 1148729400;
Prologue = "envcheck.sh";
RetryCount = 2;
ShallowRetryCount = 4;
MyProxyServer = "kuiken.nikhef.nl";
VirtualOrganisation = "infngrid";
PerusalFileEnable = true;
```

8.2. NORMAL 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"
};
OutputSandboxDestURI = {
```
"gsiftp://matrix.datamat.it:5432/tmp/sim.err",
"gsiftp://grid003.ct.infn.it:6789/home/cms/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 = {
    "file:///home/fpacini/GUI/sbin/script.sh"
};
rank = (other.GlueCEPolicyMaxRunningJobs-other.GlueCEStateRunningJobs);
// This is the default requirements expression
requirements = other.GlueCEStateStatus == "Production" ;
]

8.3. NORMAL JOB WITH DATA REQUIREMENTS
[
    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 deep re-submission in case of failure
    RetryCount = 0;
    ShallowRetryCount = 5;
    DataRequirements = {
        [
            InputData = {
                "lfn:/mydata/file1",  "lfn:/mydata/file2",
                "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"
            },
            DataCatalogType = "RLS";
            DataCatalog = "https://lcg.cern.ch/RLS";
        ],
        [
            InputData = {
                "lfn:/grid/egee/my_test/test_LFN",
                "lfn:/data/mydatafile2",
                "guid:34r57b2312-6a33-p45d9982-4e101f8f3b61"
            },
            DataCatalogType = "SI";
            // Do not specify this attribute if you want to use the VO default SI
        ]
    }
]
8.4. 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:/myout/data.1"
    ],
    [ 
      OutputFile = "dataset2.out";
      LogicalFileName = "lfn:/myout/data.2"
    ]
  }
]
8.5. JOB WITH INPUT AND OUTPUT DATA

```plaintext
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;
DataRequirements = {
    InputData = {
        "lfn:/mydata/file1",  "lfn:/mydata/file2",
        "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"
    },
    DataCatalogType = "RLS";
    DataCatalog = "https://lcg.cern.ch/RLS";
};
DataAccessProtocol = {"gsiftp", "file"};
OutputData = {
    OutputFile = "dataset1.out";
    StorageElement = "grid011.pd.infn.it";
    LogicalFileName = "lfn:/myout/data.1"
},
    OutputFile = "dataset2.out";
    LogicalFileName = "lfn:/myout/data.2"
};
```
8.6. PARAMETRIC JOB

[  
JobType = "Parametric";
Executable = "sim.exe";
StdOutput = "myoutput.txt";
StdError = "myerror.txt"
// The iteration is done on the values of this list
Parameters = {/DC1/test/data, /DC2/test/data,
             /DC3/test/data, /DC4/test/data};
DataRequirements = {
[  
    DataCatalogType = "RLS";
    // _PARAM_ assumes values from the Parameters list
    InputData = {"lfn:_PARAM_",
                 "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"};
  ]
};
InputSandbox = {
    "gsiftp://neo.datamt.it:3344/home/cms/sim.exe",
    
};
OutputSandbox = {
    "myoutput.txt",
    "myerror.txt" }
Requirements = other.GlueCEInfoTotalCPUs > 2;
Rank = other.GlueCEStateFreeCPUs;

See also Example 3.

8.7. MPI JOB

JobType = "Normal";
// This is the number of CPU needed by the job
CPUNumber = 6;
Executable = "mpi-start-wrapper.sh";
Arguments = "mpi-test OPENMPI";
StdOutput = "mpi-test.out";
StdError = "mpi-test.err";
InputSandbox = {
    "mpi-start-wrapper.sh",
    "mpi-hooks.sh",
    "mpi-test.c"
};
OutputSandbox = {
    "mpi-test.err",
    "mpi-test.out"
};
Requirements = 
Member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment) 
&& Member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment);

8.8. DAG

type = "dag";
VirtualOrganisation = "EGEE";
max_nodes_running = 10;
// shared by all nodes not specifying their own InputSandbox
InputSandbox = {
    "cdfSim.sh",
    "run_cdfSim.tcl"
};
requirements = other.GlueCEInfoLRMSType=="LSF";
rank = other.GlueCEInfoTotalCPUs ;
nodes = [
    nodeA = [
        file ="cdfSimA.jdl" ;
    ],
    nodeB = [
        file ="cdfSimB.jdl" ;
    ],
    nodeC = [
        file ="cdfSimC.jdl" ;
    ];
nodeD = [
    file = "cdfSimD.jdl" ;
];
nodeE = [
    file = "cdfSimE.jdl" ;
];
nodeF = [
    file = "cdfSimF.jdl" ;
];
nodeG = [
    file = "cdfSimG.jdl" ;
];
nodeH = [
    file = "cdfSimH.jdl" ;
];
nodeI = [
    file = "cdfSimI.jdl" ;
];
nodeL = [
    file = "cdfSimL.jdl" ;
];
nodeM = [
    file = "cdfSimM.jdl" ;
];
nodeN = [
    file = "cdfSimN.jdl" ;
];
nodeO = [
    description = [
        JobType = "cdfSim.sh";
        Executable = "13";
        Arguments = "cdfSim.out";
        StdOutput = "cdfSim.err";
        StdError = "run_cdfSim.tcl_13.log";
        OutputSandbox = { "cdfSim.out",
                         "cdfSim.err",
                         "run_cdfSim.tcl_13.log"};
        OutputSandboxDestURI = "gsiftp://gr1.infn.it:5432/tmp";]
];
nodeP = [
    file = "cdfSimP.jdl" ;
];
nodeQ = [
    file = "cdfSimQ.jdl" ;
];
nodeR = [
    file = "cdfSimR.jdl" ;
];
nodeS = [
    file = "cdfSimS.jdl" ;
];
nodeT = [
    file = "cdfSimT.jdl" ;
];

dependencies = {
    {nodeA, nodeB},
    {{nodeB, nodeC}, nodeD},
    {nodeD, nodeE},
    {nodeE, {nodeF, nodeG, nodeH, nodeI, nodeL, nodeM, nodeN, nodeO}},
See also Example 2.

8.9. COLLECTION

```plaintext
Type = "collection";
VirtualOrganisation = "EGEE";
MyProxyServer = "skurut.cesnet.cz";
InputSandbox = {
    "/tmp/foo",
    "/home/gliteuser/bar",
    "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe",
    "file:///tmp/myconf"
}; // inherited by all j*.jdl jobs
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
nodes = {

    [NodeName = "chkpt_job";
     JobType = "Normal";
     JobType = "Checkpointable";
     Executable = "hsub.exe";
     JobSteps = 10000000;
     CurrentStep = 1;
     Arguments = "gsiftp://lxde01.pd.infn.it/tmp/root_tst/";
     RetryCount = 3;
     InputSandbox = "file:///home/pippo";
     rank = (-other.GlueCEStateEstimatedResponseTime);
     requirements = other.GlueCEInfoLRMSType="pbs";
    ],
    [file = "/home/sim_test/j1.jdl"
    ],
    [file = "/home/sim_test/j2.jdl"
    ],
    [file = "/home/sim_test/j3.jdl"
    ],
    [file = "/home/sim_test/j4.jdl"
    ]

```
See also Example 4.