DataGrid

JOB SUBMISSION USER INTERFACE MAN PAGES

Document identifier:	**DataGrid-01-TEN-0101-0_1**
Date:	**21/05/2001**
Work package:	**WP1**
Partner:	**Datamat SpA**
Document status	**DRAFT**
Deliverable identifier:	

Abstract: This note provides the man pages of the Job Submission User Interface commands

Delivery Slip

	Name	Partner	Date	Signature
From	Fabrizio Pacini	Datamat SpA	21/05/2001	
Verified by	Stefano Beco	Datamat SpA	21/05/2001	
Approved by				

Document Log

Issue	Date	Comment	Author
0_0	18/04/2001	First draft	Fabrizio Pacini
0_1	21/05/2001	draft	Fabrizio Pacini

Document Change Record

Issue	Item	Reason for Change
0_1	General corrections	Take into account comments from WP1 members and addition of new commands.

Files

Software Products	User files
Word 97	DataGrid-01-TEN-0101-0_1-Document.doc
Acrobat Exchange 4.0	DataGrid-01-TEN-0101-0_1-Document.pdf

CONTENT

1. INTRODUCTION

This note provides a description of the Job Submission User Interface commands in a Unix man-page style in order to explain the main functionality supplied to the user and to highlight the interactions with the other components of the DataGrid system. It has to be intended as a preliminary effort that will be subjected to iterated refinements according to the received comments and suggestions.

In the commands synopsis the mandatory arguments are showed between angle brackets (<*arg*>) whilst the optional ones between square brackets (*[arg]*).

1.1. OBJECTIVES OF THIS DOCUMENT

Aim of this note is to provide a starting basis for the understanding of the actual user needs from the point of view of the job submission interface and to help in the identification of the interactions between the Workload Manager (WP1) components.

1.2. APPLICATION AREA

WP1 and WP8-9-10.

1.3. APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] DATAGRID WP1 Task 1.2 Job Description Language HowTo, 07/03/2001
(https://lindir.ics.muni.cz/datagrid/wp1/drafts/ClassAd-HowTo/v1.pdf)

[A2] DATAGRID - WP1 Job Submission User Interface for PM9 (revised presentation), 23/03/2001
(https://lindir.ics.muni.cz/datagrid/wp1/meetings/milano_03_2001/datamat-revised.pdf)

[A3] WP1 meeting - CESNET presentation in Milan, 20-21/03/2001
(https://lindir.ics.muni.cz/datagrid/wp1/meetings/milano_03_2001/cesnet.pdf)

[A4] Logging and Bookkeeping Service, 0705/2001
(http://lindir.ics.muni.cz/dg_public/lb_draft.pdf)

Reference documents

[R1]

1.4. DOCUMENT EVOLUTION PROCEDURE

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

- comments received from WP1 and/or other WPs partners,
- changes/evolutions/additions to the Job Submission User Interface.

1.5. TERMINOLOGY

Definitions

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

Glossary

class-ad	Classified advertisement
CE	Computing Element
DB	Data Base
GIS	Grid Information Service
job-ad	class-ad describing a job
JDL	Job Description Language
JSS	Job Submission Service
LB	Logging and Bookkeeping Service
LRMS	Local Resource Management System
MPI	Message Passing Interface
PM	Project Month
RB	Resource Broker
SMP	Symmetric Multi Processor
TBD	To Be Defined
UI	User Interface
WP	Work Package

2. EXECUTIVE SUMMARY

The purpose of the Job Submission User Interface is to provide easy access to grid scheduling and submission services for the DataGrid application users.

In this document we describe syntax and behavior of the commands made available by the UI to allow job submission, monitoring and control.

3. USER INTERFACE

Before beginning with the description of the Job Submission User Interface commands it is important to introduce the concept of UI session. Starting a session is the first **needed** step to allow the exploitation of the UI services. This step can be accomplished by means of the **dg-startUI** command that is described in the next section 3.1 of this document.

Main tasks performed by the **dg-startUI** command are:

- Setting of needed environment variables
- Creation of the UI configuration files in the run-time configuration space
- Creation of a proxy certificate for the user, which can be used for authentication

An option of the **dg-startUI** command allows the user to start the UI session with particular settings according to the “working group” he/she belongs to.

A UI session can be terminated by using the **dg-stopUI** command described in the next section 3.1 of this document, anyway to change the UI session a **dg-startUI** can be reissued without needing a previous call of the **dg-stopUI**.

3.1. COMMANDS DESCRIPTION

– *dg-startUI*

Allows the user to start a User Interface session.

SYNOPSIS

dg-startUI [-help] [-version]

dg-startUI [-config *group_name*]

DESCRIPTION

dg-startUI starts a job submission UI session. It sets the needed environment variables and stages the configuration files in the UI run-time configuration space. Moreover it creates a proxy certificate for the user that can be used for authentication without having to re-enter the protecting pass-phrase for each operation. At this aim the user is asked for the certificate password when issuing this command.

The *-config* option allows the user to specify the group or experiment he/she belongs to when submitting the jobs. When issued with this option the **dg-startUI** command performs "group specific" environment initialisations and loads "group specific" configuration files.

OPTIONS

-help

displays command usage.

-version

displays UI version.

-config *group_name*

performs environment settings and creates configuration files specific to the group specified by *group_name*.

EXIT STATUS

dg-startUI exits with a status value of 0 (zero) upon success, and a >0 value upon failure.

EXAMPLES

1. $> **dg-startUI**

```
Enter pass phrase: ********
UI >>
```

starts a UI session with default settings

2. $> **dg-startUI** –config *CMS*

```
Enter pass phrase: ********
UI – CMS >>
```

starts a UI session with specific settings for the CMS experiment.

– *dg-stopUI*

Terminates a User Interface Session.

SYNOPSIS

dg-stopUI [-help] [-version]

DESCRIPTION

dg-stopUI closes a job submission UI session. It resets UI environment settings and destroy the user certificate proxy created by **dg-startUI.**

OPTIONS

-help

displays command usage

-version

displays UI version.

EXIT STATUS

dg-stopUI exits with a status value of 0 (zero) upon success, and a >0 value upon failure.

EXAMPLES

```
UI >> dg-stopUI
UI session has been successfully closed
$>
```

– *dg-job-submit*

Allows the user to submit a job for execution on remote resources in a grid.

SYNOPSIS

dg-job-submit [-help] [-version] [-template]

dg-job-submit <*job_description_file*> [-resource *res_id*] [-verbose] [-notify *e_mail_address*]

DESCRIPTION

dg-job-submit allows the user to run a job at one or several remote resources. It is the command for submitting jobs to the grid. **dg-job-submit** requires as input a job description file in which job characteristics and requirements are expressed by means of a Condor class-ad.

The job description file is syntactically checked and default values are assigned to some of the mandatory attributes not provided in the job description file. The resulting job-ad is sent to the Resource Broker that finds the job best matching resource (*match making*) and submits the job to it.

This command returns the *dg_jobID* of the job (a string that identifies unambiguously the job in the whole DataGrid), generated by the User Interface, that can be later used as a handle to monitor the job status (see **dg-job-status** described later in this document). Format of the *dg_jobId* is still TBD, anyway it will for sure contain hostname, current date and time, process identifier and used RB and LB addresses (e.g. *<hostname>__<date>_<time>_<PID>_<RB address>_<LB address>*).

The *-resource* option can be used to target the job submission to a specific known resource identified by the provided *res_id* (returned by **dg-list-job-match** described later in this document). For PM9 a resource will be either a queue of an underlying LRMS, assuming that this queue represents a set of "homogeneous" resources or a "single" node. The *res_id* is a string, provided by WP4 services, univocally identifying the resources published in the GIS.

When this option is specified, the job-ad to be sent to the RB is completed with the *ResourceId* attribute valued with the provided *res_id* and this makes the Resource Broker skip the match making process and directly submit the job to the requested resource.

It is important to note that the user can specify the number of time the JSS must resubmit a job to a resource in case the submission fails (e.g. the resource is temporary unavailable) by means of the JDL attribute *RetryCount*. Only when all requested re-submissions fail the job is aborted. If the user does not provide the value of RetryCount it is assigned with a default value by the UI.

When **dg-job-submit** is used with the *–notify* option the job-ad that is sent to the RB is completed with the *UserContact* attribute valued with the provided *e_mail_address* and the following schema is used to notify the user of job status changes:

- an e-mail notification is sent to *e_mail_address* when the matchmaking process has finished and the job is ready to be submitted to JSS (READY status)

- an e-mail notification is sent to *e_mail_address* when the job starts running on the CE (RUNNING status)
- an e-mail notification is sent to *e_mail_address* when the job has finished (ABORTED or DONE status).

The notification message will contain basic information on the job such as the job identifier and a brief description of its status.

The **dg-job-submit** command has a particular behaviour when the job description file contains the *InputSandbox* attribute whose value is a list of file paths on the UI machine local disk. Let's suppose to have a job that needs for the execution a certain set of files having a small size and available on the submitting machine. Let's also suppose that for performance reasons it is preferable not going through the WP2 data transfer services for the staging of these files on the executing node. Then the user can use the *InputSandbox* attribute to specify the files that have to be staged from the submitting machine to the executing resource. All of them are indeed transferred at job submission time together with the job class-ad to the RB that will store them temporarily on its local disk. The JSS will then perform the staging of these files on the executing node. The size of files to be transferred to the RB should be small since overfull of RB local storage means that no more job of this type can be submitted.

Since the *InputData* and *InputSandbox* expressions, can consist respectively of a great number of file names and logical collection names, it is admitted the use of wildcards to specify the value of these attributes. Syntax and allowed wildcards are described in Appendix 4.3.

If at submission time the user has also specified the *OutputSandbox* attribute (also representing a list of file names), then it is possible, when the job has finished, to copy all files, resulting from the job execution and listed in *OutputSandbox,* on the UI machine issuing the **dg-get-job-output** command described later in this document.

This mechanism can be also used to stage a job executable available locally on the UI machine to the executing CE. Indeed through the *Executable* JDL attribute (see also Table 2) the user can either specify an executable that already resides on the remote CE, prefixing it with an appropriate environment variable (e.g. $(CMS)) or provide a local executable name that has to be transferred to the remote CE. In the latter case the executable name shall also be repeated in *InputSandbox* attribute expression.

JOB DESCRIPTION FILE

A job description file contains a description of job characteristics and constraints in a class-ad style. Details on the class-ad language are reported in the document [A1] also available at the following URL http://www.mi.infn.it/~prelz/grid/classad-howto.pdf.

This file must be edited by the user to insert relevant information on the job that is later needed by the RB to perform the match making. A template of the job description file, containing a basic set of attributes can be obtained by calling the **dg-job-submit** command (described in this document) with the *-template* option. Job description file entries are strings having the format *attribute* = *expression* and are terminated by the semicolon character. If the entry spans more than one line, the end of line has to be indicated with a "\" character. Comments must be preceded by a sharp character (#) at the beginning of each line.

Being the class-ad an extensible language, it doesn't exist a fixed set of admitted attributes, i.e. the user can insert in the job description file whatever attribute he believes meaningful to describe its jobs, anyway only the attributes that can be in some way related with the resource ones published in the GIS are taken into account by the Resource Broker for the match making process. Unrelated attributes are simply ignored (at least until these resources properties will not be published in the GIS). The attributes taken into account by the RB together with their meaning are reported in Table 2 of Annexes.

There is a small subset of class-ad attributes that are compulsory, i.e. that have to be present in a job class-ad before it is sent to the Resource Broker in order to make possible the performing of the match making process.

They can be grouped in three categories: some of them **must** be provided by the user whilst some other, if not provided, are filled by the UI with configurable default values. Lastly there are some whose filling is reserved to the UI.

The following Table 1 summarises what just stated.

Attribute	**Mandatory**	**Mandatory with default value (*default value*)**	**Reserved to UI**
dg_jobId			✓
Executable	✓		
Requirements		✓ (TRUE)	
Rank		✓ (-EstimatedTraversalTime)	
RetryCount		✓ (3)	
CertificateSubject			✓
ReplicaCatalog	✓		

Table 1 Mandatory Attributes

OPTIONS

-help

displays command usage.

-version

displays UI version.

-template

-t

creates in the current working directory a file named *jobDescr.jdl* containing a template of the job description file.

-verbose

-v

displays on the standard output the job class-ad (resulting from the job description file checking and completion) that is sent to the Resource Broker.

-resource *res_id*

-r *res_id*

if the commands is launched with this option, the job-ad sent to the RB contains a line of the type *ResourceId = res_id* and the job is submitted by the Resource Broker to the resource identified by *res_id*.

-notify *e_mail_address*

-n *e_mail_address*

when a job is submitted with this option an e-mail message containing basic information pertaining the job identification and status is sent to the specified *e_mail_address* when the job enters one of the following status:

- READY
- RUNNING
- ABORTED or DONE

EXIT STATUS

dg-job-submit exits with a status value of 0 (zero) upon success, and a >0 value upon failure.

EXAMPLES

1. $> **dg-job-submit** myjob1.jdl –resource *firefox.esrin.esa.it:2119/jobmanager-condor*

   where *myjob1.jdl* is as follows:

```
###############################################
#
# -------- Job description file ----------
#
###############################################
Executable   = "$(CMS)/fpacini/exe/sum.exe";
InputData    = "http://ramses.esrin.esa.it/rams/dataset1";
RetryCount   = 10;
Rank         = other.MaxCpuTime;
Requirements = other.ResourceManagementType =="Condor" && \
               other.Architecture =="INTEL" && other.OpSys=="LINUX" && \
               other.FreeCpus >= 4;
```

   submits *sum.exe* to the resource identified by *res_id*, whose LRMS is Condor. The command returns a job handle (dg_*jobID*) to the user:

   $> Your job identifier is: datagrid.esrin.esa.it__20010511_163007_2033_RB2_LB1

2. $> **dg-job-submit** myjob2.jdl –notify *fpacini@datamat.it*

   submits the job described by *myjob2.jdl* , returns a job handle (*dg_jobId*) to the user and sends a notification by e-mail at well defined job status changes to fpacini@datamat.it.

SEE ALSO

[A1], [A2], **dg-list-job-match**.

– *dg-get-job-output*

This command requests the RB for the job output files (specified by the *OutputSandbox* attribute of the job-ad) and stores them on the submitting machine local disk.

SYNOPSIS

dg-get-job-output [-help] [-version]

dg-get-job-output <*dg_jobId*> [-dir *directory_path*]

DESCRIPTION

The **dg-get-job-output** command has to be used to retrieve the output files of a job that has been submitted through the **dg-job-submit** command with a job description file including the *InputSandbox* and *OutputSandbox* attributes. After the submission, when the job has terminated its execution, the user can load the files generated by the job and temporarily stored on the RB machine as specified by the *OutputSandbox* attribute issuing the **dg-get-job-output** with as input the *dg_jobId* returned by the **dg-job-submit**.

It is possible to specify the local directory path on the UI machine where these files have to be stored by mean of the –*dir* option, otherwise they are put in a default location.

OPTIONS

-help

displays command usage.

-version

displays UI version.

-dir *directory_path*

retrieved files (previously listed by the user through the *OutputSandbox* attribute of the job description file) are stored in the location indicated by *directory_path.*

dg_jobId

job identifier returned by **dg-job-submit**.

EXIT STATUS

dg-get-job-output exits with a status value of 0 (zero) upon success, and a non-zero value upon failure.

EXAMPLES

Let us consider the following command:

```
$> dg-get-job-output firefox.esrin.esa.it__20010514_163007_21833_RB1_LB3 –dir
/home_firefox/fpacini/data
```

it retrieves the files listed in the *OutputSandbox* attribute from the RB and stores them locally in */home_firefox/fpacini/data*.

– *dg-list-job-match*

Returns the list of resources fulfilling job requirements.

SYNOPSIS

dg-list-job-match [-help] [-version]

dg-list-job-match <*job_description_file*> [-verbose] [-output *output_file*]

DESCRIPTION

dg-list-job-match displays the list of identifiers of the resources accessible by the user and satisfying the job requirements included in the job description file. The resources identifiers are returned either on the standard output or in a file according to the chosen command option and are strings univocally identifying the resources published in the GIS.

dg-list-job-match requires a job description file in which job characteristics and requirements are expressed by means of a Condor class-ad. The job description file is first syntactically checked and then used as the main command-line argument to dg-list-job-match. The Resource Broker is only contacted to find job compatible resources; the job is never submitted.

See the **dg-job-submit** section and in particular Table 1 for general rules for building the job description file.

The option *-verbose* of the **dg-list-job-match** command can be used to obtain on the standard output the complete class-ad sent to the RB.

JOB DESCRIPTION FILE

See **dg-job-submit** for details.

OPTIONS

-help

displays command usage.

-version

displays UI version.

-verbose

-v

displays on the standard output the job class-ad that is sent to the Resource Broker.

-output *output_file*

-d *output_file*

returns the queue identifiers list in the file specified by *output_file* instead of the standard output. *output_file* can be either a simple name or an absolute path (on the submitting machine). In the first case the file *output_file* is created in the current directory from which the command is launched.

EXIT STATUS

dg-list-job-match exits with a status value of 0 (zero) upon success, and a non-zero value upon failure.

EXAMPLES

Let us consider the following command:

$> **dg-list-job-match** myjob.jdl

where the job description file *myjob.jdl* looks like:

```
#################################
#
# Sample Job Description File
#
#################################

Executable   = "/home_firefox/fpacini/exe/sum.exe";
InputData    = "http://ramses.esrin.esa.it/rams/dataset1";
InputSandbox = "/home_firefox/fpacini/exe/sum.exe";
RetryCount   = 4;
Rank         = other.MaxCpuTime;
Requirements = other.ResourceManagementType =="Condor" &&
               other.Architecture =="INTEL" && other.OpSys=="LINUX" &&
               other.FreeCpus >= 2;
```

In this case the job requires queues being Condor Pools of INTEL LINUX machines with at least 2 free Cpus. Moreover the *Rank* expression states that queues with higher maximum Cpu time allowed for jobs are preferred.

The response of such a command is represented by a list of *ResourceIds.*

SEE ALSO

[A1],[A2], **dg-job-submit**.

– *dg-job-cancel*

Cancels one or more submitted jobs.

SYNOPSIS

dg-job-cancel [-help] [-version]

dg-job-cancel < *dg_jobId*$_1$ *….* *dg_jobId*$_n$ | *-all* >

DESCRIPTION

This command cancels a job previously submitted using **dg-job-submit**. Before cancellation, it prompts the user for confirmation. The cancel request is sent to the Resource Broker that fulfills it.

dg-job-cancel can remove one or more jobs: the jobs to be removed are identified by one or more job identifiers (dg_jobIds returned by **dg-job-submit**) provided as arguments to the command and separated by a blank space. The result of the cancel operation is reported to the user for each specified dg_jobId.

If the *-all* option is specified, all the jobs owned by the user submitting the command are removed. The IDs of the cancelled job are printed on the standard output. When the command is launched with the *-all* option, no dg_jobId can be specified. It has to be remarked that for PM9 only the owner of the job can remove the job.

OPTIONS

-help

displays command usage.

-version

displays UI version.

-all

cancels all job owned by the user submitting the command. This option can not be used if a dg_jobIds list has been already specified in the command

dg_jobId

job identifier returned by **dg-job-submit**.

EXIT STATUS

dg-job-cancel exits with a status value 0 if all the specified jobs were cancelled successfully

>0 otherwise

EXAMPLES

1. $> **dg-job-cancel** dg_jobId1 dg_jobId2

displays the following confirmation message:

```
Are you sure you want to remove dg_jobId1 dg_jobId2? [Y/N] Y
   dg_jobId1 successfully removed
   dg_jobId2 not found
   $>
```

2. $> **dg-job-cancel** –all

displays the following confirmation message:

```
Are you sure you want to remove all jobs owned by user fpacini?
[Y/N] Y
   dg_jobId1 successfully removed
   dg_jobId2 successfully removed
   dg_jobId3 successfully removed
   $>
```

SEE ALSO

[A2], **dg-job-submit**.

– *dg-job-status*

Displays bookkeeping information about submitted jobs.

SYNOPSIS

dg-job-status [-help] [-version]

dg-job-status < *dg_jobId$_1$ …. dg_jobId$_n$* | *-all* > [-full]

DESCRIPTION

This command prints the status of a job previously submitted using **dg-job-submit**. The job status request is sent to the LB that provides the requested information.

dg-job-status can monitor one or more jobs: the jobs to be checked are identified by one or more job identifiers (dg_jobIds returned by **dg-job-submit**) provided as arguments to the command and separated by a blank space.

If the *-all* option is specified, information about all the jobs owned by the user submitting the command are printed on the standard output. When the command is launched with the *–all* option, no dg_jobId can be specified.

The job information displayed to the user encompasses (bookkeeping information):

- dg_jobId
- jobStatus (the job current status)
- CertificateSubject
- Executable
- InputData
- OutputData
- ResourceId (if the job has been already scheduled)
- submissionTime (when the job has been submitted from the UI; SUBMITTED status)
- scheduledTime (when the job has been submitted to the resource; SCHEDULED status)
- startRunningTime (when the job has started its execution; RUNNING status)
- StopRunningTime (when the job has completed its execution; DONE or ABORTED status)

If the *-full* option is specified, **dg-job-status** displays a long description of the queried jobs by printing in addition the whole job class-ad (job-ad) plus, if available, some dynamic job information like CPU time, memory and disk consumption.

The *jobStatus* possible values are reported in Appendix 4.2. Details on the Job Status Diagram can be found in [A4].

OPTIONS

-help

displays command usage.

-version

displays UI version.

-all

displays status information about all job owned by the user submitting the command. This option can not be used if a dg_jobIds list has been already specified in the command

-full

displays a long description of the queried jobs

dg_jobId

job identifier returned by **dg-job-submit**

EXIT STATUS

dg-job-status exits with a status value 0 if job status could be retrieved >0 otherwise.

EXAMPLES

$> **dg-job-status** dg_jobId2

displays the following lines:

```
****************************************************************************************************
dg_jobID              =  firefox.esrin.esa.it__20010514_163007_21833_RB1_LB3
Certificate Subj      =  /C=IT/O=ESA/OU=ESRIN/CN=Fabrizio Pacini/Email=fpacini@datamat.it
JobStatus             =  RUNNING
Executable            =  $(ATLAS)/exe/sum.exe
InputData             =  http://ramses.esrin.esa.it/rams/dataset1
ResourceId            =  firefox.esa.it:2119/jobmanager-condor
submissionTime        =  10:24:32 05-06-2001 GMT
scheduledTime         =  10:25:45 05-06-2001 GMT
startRunningTime      =  10:26:01 05-06-2001 GMT

****************************************************************************************************
```

SEE ALSO

[A1], [A2], [A4], **dg-job-submit**.

– *dg-get-logging-info*

Displays logging information about submitted jobs.

SYNOPSIS

dg-get-logging-info [-help] [-version]

dg-get-logging-info < *dg_jobId$_1$ …. dg_jobId$_n$* | *-all* > [-from *T1*] [-to *T2*] [-**output** *output_file*] [-full]

DESCRIPTION

This command queries the LB persistent DB for logging information about jobs previously submitted using **dg-job-submit**. The job logging information are stored permanently by the LB service and can be retrieved also after the job has terminated its life-cycle, diffrently from the bookkeeping information that are in some way "consumed" by the user during the job existence.

The **dg-get-logging-info** request is sent to the LB service that queries the DB and returns the retrieved information. Content of the logging information is the complete job class-ad plus

- jobStatus
- ResourceId (the resource or list of resources where the job run)
- submissionTime
- scheduledTime
- startRunningTime
- StopRunningTime

If the **dg-get-logging-info** command has been issued with the *–full* option then some additional information is provided, such as the identifiers of the RB, JSS and LB "used" by the job together with some job dynamic information (e.g. CpuTime, WallClockTime, memory and disk consumption) if available.

It is important to note that the format of the returned information depends on the LB model. Indeed it distinguishes between two types of collected information, namely cumulative (e. g. CPU time) and non-cumulative (e. g. memory consumption). For cumulative properties not all the values are stored in the database but the corresponding cell is overwritten with the most actual value (naturally time-stamped). Non-cumulative properties are instead stored as sent in order to allow e. g. creation of appropriate profiles.

Data on several jobs can be queried by specifying a list of job identifiers separated by a blank space as arguments of the command.

If the *-all* option is used, logging information about all the jobs owned by the user submitting the command are printed on the standard output.

When the command is launched with the *-all* option, no *dg_jobId* can be specified.

The user can also specify a time range he is interested to by using the *-from* and *-to* options. These options take as input timestamps in the format *hh:mm:ss DDMMYYYY* (GMT) and make the command retrieve job logging information only for the specified time interval. When a time range is not specified the query retrieves information logged in the current day, i.e. default value for the *-from* option is *00:00:00* of current day and for the *-to* option is current time. Anyway these default values are configurable locally.

The *-output* option can be used to have the retrieved information written in the file identified by *output_file* instead of the standard output. If the *-full* option is specified, **dg-get-logging-info** returns the whole information available for the given job.

OPTIONS

-help

displays command usage.

-version

displays UI version.

-all

retrieves logging information about all job owned by the user submitting the command. This option can not be used if a dg_jobIds list has been already specified in the command

-full

retrieves the whole information available in the DB for the specified job.

-output *output_file*

-d *output_file*

writes the logging information in the file specified by *output_file* instead of the standard output. *output_file* can be either a simple name or an absolute path (on the submitting machine). In the first case the file *output_file* is created in the current directory from which the command is launched.

-from *hh:mm:ss* [*DDMMYYYY*]

get information for the time since the specified date. If *DDMMYYYY* is not provided, input time is considered in the current day.

-to *hh:mm:ss* [*DDMMYYYY*]

get information for the time up to the specified date. If *DDMMYYYY* is not provided, input time is considered in the current day.

dg_jobId

job identifier returned by **dg-job-submit**

EXIT STATUS

dg-get-logging-info exits with a status value 0 if job logging info could be retrieved >0 otherwise.

EXAMPLES

1. $> **dg-get-logging-info** –all –from *12:15:00 04052001* –to *10:00:00 05052001* –**output** *mylog.txt*

writes in file *mylog.txt* in the current working directory logging information about my jobs for the time since 12:15 on 5 May 2001 up to 10 o'clock on 6 May 2001.

2. $> **dg-get-logging-info** dg_jobId1 –from 11:35:00

writes in the standard output the logging information of job identified by *dg_jobId1* from today at 11:35 up to now.

SEE ALSO

[A2], [A4], **dg-job-submit**.

4. ANNEXES

4.1. JDL ATTRIBUTES

The following Table 2 and Table 3 report the recommended set of class-ad attributes to be used to write the job description file when submitting a request through the UI commands **dg-job-submit** and **dg-list-job-match**. Table 2 contains job specific attributes, i.e. the ones to be used to describe job characteristics, whilst Table 3 contains resource specific attributes that can be used to express job constraints and preferences, i.e. to build the *Requirements* and *Rank* expressions.

The JDL language is fully extensible, hence it is allowed to use whatever attribute in the job description file, anyway the user should base the specification of job constraints and preferences only on the attributes advised in Table 3 (at least for PM9) since they are the ones that are related with those published in the GIS and can then be taken into account by the RB for the match making process. Those attributes whose support is not yet assured for PM9 delivery are greyed.

Attribute	**Meaning**
dg_jobId	Job unique identifier; inserted by the UI
CertificateSubject	Subject of the user certificate; Inserted by the UI
Executable	Executable/command name. The user can specify an executable that lies already on the remote CE. Name of this executable should start with some environment variable indicating its location (e.g. $(CMS)). The other possibility is to provide a local executable name, which will be staged on the CE. In this case the executable should be listed also in the *InputSandbox* attribute expression.
InputData	- a logical collection, a list logical collections and/or - a list of logical files and/or - a list of physical files
StdInput	Standard input
StdOutput	Standard output
StdError	Standard error
OutputSE	URI of Storage Element where to store the output
InputSandbox	List of files on the UI local disk needed by the job for running

Attribute	Meaning
OutputSandbox	List of files generated by the job
RetryCount	Number of job submission retrial made by JSS
ReplicaCatalog	Replica Catalogue Identifier, i.e. something in the following format: `<protocol>://<full hostname> :<port>/<Replica Catalog DN>.`
UserContact	e-mail address for notification
Rank	a ClassAd Floating-Point expression that states how to rank queues that have already met the Constraints expression. Essentially, rank expresses preference. A higher numeric value equals better rank. The RB will give the job the queue with the highest rank.
Requirements	Boolean ClassAd expression which uses C-like operators. It represents job requirements on resources. In order for a job to run on a given queue, this Constraint expression must evaluate to true on the given queue.
IsInteractive	Boolean. If true indicates that the job is interactive.
JobPriority	Job priority
OutputData	- logical collection, a list logical collections and/or - a list of logical files and/or - a list of physical files

Table 2 class-ad job specific attributes

Attribute	Meaning
ResourceId	Unique resource identifier
GlobusResourceContactString	Globus resource manager contact string.
QueueName	Queue name
ResourceManagementType	Defines the type of resource management system (LSF/Condor/…).
ResourceManagementVersion	The version of the local resource management system.
GRAMVersion	the GRAM version.
Architecture	the architecture of the machine or of the machines associated to the queue (we assume that all the machines “belonging” to the queue have the same architecture).
OpSys	the operating system of the machine or of the machines associated to the queue (assuming that all these machines run the same operating system).
PhysicalMemory	Minimum available physical memory (in bytes) associated to the queue.
LocalDisk	Local disk footprint
TotalCPUs	the number of total CPUs associated to the resource.
FreeCPUs	the total number of free processors associated to the resource, processors able to run, in that moment, jobs submitted to the resource.
NumSMPs	number of SMP processors associated to the resource.
TotalJobs	the number of jobs submitted to the resource, jobs that have not already been completed.
RunningJobs	The number of jobs submitted to the resource that are currently running.
IdleJobs	the number of jobs submitted to the resource, jobs that are not running since they are waiting for available resources.

Attribute	Meaning
MaxTotalJobs	the maximum number of jobs (running and idle) allowed for the resource.
MaxRunningJobs	the maximum number of running jobs allowed for the resource.
WorstTraversalTime	Worst traversal time
EstimatedTraversalTime	Scaled value of the last traversal time
Status	the status of the resource. For a queue if it is ready or not to dispatch jobs to the executing machines.
RunWindows	the time windows that define when the resource is active, (for a queue: ready to dispatch jobs to the executing machines).
Priority	the priority of the resource.
MaximumCPUTime	the maximum CPU time allowed for jobs submitted to the resource.
MaximumWallClockTime	the maximum wall clock time allowed for jobs submitted to the resource.
MinSI00	Min SpecInt2000
MaxSI00	Max SpecInt2000
AvgSI00	Average SpecInt2000
RunTimeEnvironments	List of “tag(s)” identifying the appropriate test-bed SW installation (e.g. CMSVersion+CMSRoot or a list of RPM names/hashes)
AFSAvailable	Boolean indicating if AFS is available at the CE
OutboundIP	Boolean indicating that the resource has access to the internet
InboundIP	Boolean indicating that the resource can be accessed from the internet
StorageElements	List of SE URI’s that are close “enough” to the resource (e.g. on the same LAN)
AuthorizationPolicies	Grid map file entries (TBD)

Table 3 class-ad Resource specific attributes

4.2. JOB STATUS DIAGRAM

The following Figure 1 reports the status that a job can assume during its life cycle.

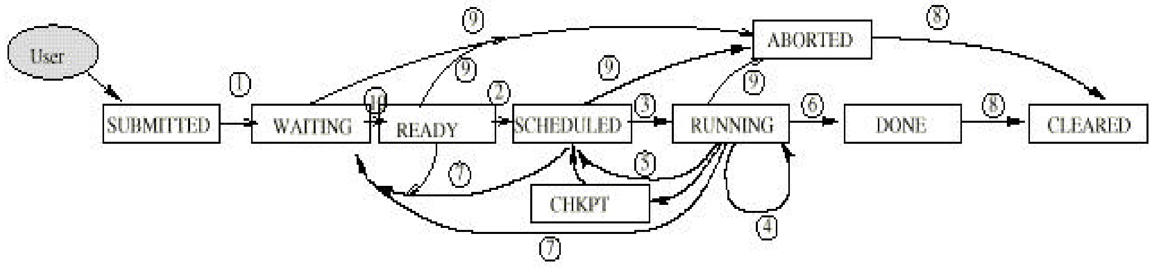

Figure 1 Job Life Cycle

Job status in Figure 1 are briefly described hereafter (see [A4] for further details):

STATUS:

- **SUBMITTED**: job is submitted but not yet received by the RB (i.e. it is waiting in the UI).
- **WAITING**: job is waiting in the queue in the RB for various reason (no appropriate CE (cluster) found; required dataset is not available, dependency on other job).
- **READY**: appropriate CE found; job is transferred to the CE.
- **SCHEDULED**: job is waiting in the queue on the CE.
- **RUNNING**: job is running.
- **CHKPT**: job is check-pointed and is waiting for restart; this is a system checkpointing of jobs running on a CE, independently from Application Checkpointing.
- **DONE**: job successfully exited.
- **ABORTED**: job was aborted for various reasons (waiting in the queue in RB,JSS or CE for too long, over-use of quotas, expiration of user credentials, etc.).
- **CLEARED**: output files were transferred to the user, job is removed from bookkeeping database.

4.3. WILDCARD PATTERNS

The wildcard patterns that can be included in the *InputData* and *InputSandbox* attributes expressions are used by the UI and the RB to perform file name "globbing" in a fashion similar to the UNIX csh shell. The result of the "globbing" is a list of the files whose names match any of the specified *patterns*.

The admitted special characters together with their meaning are listed hereafter:

- *****	wildcard for any string
- **?**	wildcard for any single character
- **[***chars* **]**	delimits a wildcard matching any of the enclosed characters. If *chars* contains a sequence of the form ***a-b*** then any character between ***a*** and ***b*** (inclusive) will match. Such an expression can be negated by means of the special character "**!**" (**[!***chars***]** matches any character not in *chars*).
- **{***a,b,…..* **}**	alternation of comma-separated alternatives; thus, \`foo{baz,qux}' would be read as \`foobaz' or \`fooqux'

EXAMPLES

Consider a directory where "ls –F" gives:

```
1file     a1.f      apple.o   bob.o     h4374.f   john.o
2files    ab        apps/     foo.c     h4374.o   mydir/
ABS       ab.f      bob       foo.f     john      stuff/
a1        apple.f   bob.f     gh        john.f
```

That is to say some files and directories. The examples below show the way the mentioned wildcards are expanded (the notation **=>** indicates the result of typing the command).

1) Every two letter file name:

   `echo ??` **=>** a1 ab gh

2) Every two character name starting with "a":

   `echo a?` **=>** a1 ab

3) Every file starting with j, o, h, or n:

   `echo [john]*` **=>** h4374.f h4374.o john john.f john.o

4) Include a range, e.g. everything starting with an upper case letter or a digit:

   `echo [A-Z0-9]*` **=>** 1file 2files ABS

5) Negate a range:
   `echo [!john]*.f =>` a1.f ab.f apple.f bob.f foo.f

6) Every file starting in "a" and ending in .f:
   `echo a*.f =>` a1.f ab.f apple.f