# Contents

1 User Guide  
  1.1 Getting Started .................................................. 3  
  1.2 Web Portal Reference ............................................. 11  
  1.3 Web Portal User guide ............................................ 37  
  1.4 Commands Reference ............................................. 52  
  1.5 Tutorials ............................................................. 72  

2 Administrator Guide  
  2.1 DIRAC Setup Structure ........................................... 93  
  2.2 Getting Started .................................................... 94  
  2.3 DIRAC Server Installation ........................................ 96  
  2.4 Install WebAppDIRAC ............................................. 105  
  2.5 System Administrator Console .................................. 113  
  2.6 DIRAC Configuration .............................................. 118  
  2.7 DIRAC Systems .................................................... 159  
  2.8 Managing DIRAC Sites ........................................... 240  
  2.9 Multi-VO DIRAC .................................................. 245  
  2.10 Administrator Command Reference ............................ 256  

3 Developer Guide  
  3.1 Development Model ............................................... 297  
  3.2 Development Environment ....................................... 311  
  3.3 Architecture overview .......................................... 320  
  3.4 Coding Conventions .............................................. 325  
  3.5 Developing DIRAC components .................................. 327  
  3.6 Documenting your developments ................................ 361  
  3.7 Testing (VO)DIRAC ............................................... 361  
  3.8 Framework Overview ........................................... 372  
  3.9 REST Interface .................................................. 379  
  3.10 WebAppDIRAC .................................................. 381  

4 Documentation sources ............................................. 399  

5 Indices and tables .................................................. 401
The DIRAC (Distributed Infrastructure with Remote Agent Control) project is a complete Grid solution for a community of users such as the LHCb Collaboration. DIRAC forms a layer between a particular community and various compute resources to allow optimized, transparent and reliable usage.

A high level overview gives a general idea of the DIRAC functionality. A more detailed description of the DIRAC system can be found at this location.

The DIRAC Workload Management system realizes the task scheduling paradigm with Generic Pilot Jobs (or Agents). This task scheduling method solves many problems of using unstable distributed computing resources which are available in computing grids. In particular, it helps the management of the user activities in large Virtual Organizations such as LHC experiments. In more details the DIRAC WMS with Pilot Jobs is described here.
This page is the work in progress. More material will come soon.

A number of DIRAC tutorials is collected in the DIRAC project GitHub repository.

1.1 Getting Started

1.1.1 Installing DIRAC client

The DIRAC client installation procedure consists of several steps.

Install script

Download the `dirac-install` script from:

```
chmod +x dirac-install
```

Choose the directory where you want to install the DIRAC software and run the `dirac-install` script from this directory giving the appropriate version of the DIRAC release:

```
dirac-install -r v5r9
```

This installs the software and you should get the following directories and files created:

```
drw-r-xr-x 8 atsareg lhcb 2048 Mar 25 2010 Linux_x86_64_glibc-2.5
-dw-r-xr-x 16 atsareg lhcb 2048 Oct 12 12:13 DIRAC
-rw-r-xr-x 1 atsareg lhcb 21224 Oct 12 13:37 dirac-install
-dw-r-xr-x 2 atsareg lhcb 10240 Oct 12 17:11 scripts
-rw-r--r-- 1 atsareg lhcb 998 Oct 12 17:15 bashrc
```

Instead of the `Linux_x86_64_glibc-2.5` directory there can be another one that corresponds to the binary platform of your installation. The `scripts` directory contains command line tools. The `DIRAC` directory has all the software. Finally, the `bashrc` script is there to easily set up the environment for your DIRAC installation:

```
source bashrc
```

Think of adding the above line to your login scripts.
Installing with VO defaults

In most cases you are installing the DIRAC client to work as a member of some particular user community or, in other words, Virtual Organization. The managers of your Virtual Organization can prepare default settings to be applied for the DIRAC client installation. In this case the installation procedure reduces to the following assuming the name of the Virtual Organization *dirac*:

```
chmod +x dirac-install
dirac-install -V formation
source bashrc
dirac-proxy-init
dirac-configure defaults_formation.cfg
```

The *dirac_defaults.cfg* file contains the Virtual Organization default settings. It is downloaded as part of the installation procedure. Check with your Virtual Organization managers if this mode of installation is available.

Configuring client

Once the client software is installed, it should be configured in order to access the corresponding DIRAC services. The minimal necessary configuration is done by the following command:

```
```

where -S option is specifying the DIRAC Setup name within which the client will be working. The -C option is to define the URL of the Configuration Service that the client will contact to discover all the DIRAC services. The exact values of the command options are specific for a given user community, ask the group administrators for the details. Typically, a single community specific installation scripts are provided which are including all the necessary specifications.

Updating client

The client software update when a new version is available is simply done by running again the *dirac-install* command giving the new version value.

1.1.2 Getting User Identity

To start working with the Grid in general and with DIRAC in particular, the user should join some grid Virtual Organization and obtain a Grid Certificate. The procedure to obtain the Grid Certificate depends on the user’s national Certification Authority (CA). The certificate is usually obtained via a Web interface and is downloaded into the user’s Web Browser. To be used with the Grid client software, the certificate should be exported from the Browser into a file in p12 format. After that the certificate should be converted into the pem format and stored in the user home directory. If the DIRAC client software is available, the conversion can be done with the following DIRAC command:

```
$ dirac-cert-convert.sh <cert_file.p12>
```

The user will be prompted for the password used while exporting the certificate and for the pass phrase to be used with the user’s private key. Do not forget it!

Registration with DIRAC

Users are always working in the Grid as members of some User Community. Therefore, every user must be registered with the Community DIRAC instance. You should ask the DIRAC administrators to do that, the procedure can be different for different communities.
Once registered, a user becomes a member of one of the DIRAC user groups. The membership in the group determines the user rights for various Grid operations. Each DIRAC installation defines a default user group to which the users are attributed when the group is not explicitly specified.

**Proxy initialization**

Before a user can work with DIRAC, the user’s certificate proxy should be initialized and uploaded to the DIRAC ProxyManager Service. This is achieved with a simple command:

```
proxy-init
```

In this case the user proxy with the default DIRAC group will be generated and uploaded. If another non-default user group is needed, the command becomes:

```
proxy-init -g <user_group>
```

where “user_group” is the desired DIRAC group name for which the user is entitled.

### 1.1.3 User Jobs

Here is a brief description of how to submit and follow simple user jobs in DIRAC

**Job command line tools**

In order to submit a job, it should be described in a form of JDL. An example JDL for a simple job is presented below:

```plaintext
Executable = "/bin/cp";
Arguments = "my.file my.copy";
InputSandbox = {"my.file"};
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err","my.copy"};
CPUTime = 10;
```

This job will take a local file “my.file”, put it into the Input Sandbox and then copy it to the “my.copy” file on the Grid. In the Output Sandbox the new copy will be returned together with the job standard output and error files. To submit the job one should execute:

```
> dirac-wms-job-submit job.jdl
JobID = 11758
```

where the job.jdl file contains the job JDL description. The command returns the JobID which is a unique job identifier within the DIRAC Workload Management System. You can now follow the status of the job by giving:

```
> dirac-wms-job-status 11758
JobID=11758 Status=Waiting; MinorStatus=Pilot Agent Submission; Site=CREAM.CNAF.it;
```

In the output of the command you get the job Status, Minor Status with more details, and the site to which the job is destinated.

Once the job in its final Status ( Done or Failed ), you can retrieve the job outputs by:

```
> dirac-wms-job-get-output 11702
Job output sandbox retrieved in 11702/
```

This will retrieve all the files specified in the job Output Sandbox into the directory named after the job identifier.
Web Job Launchpad

The Job Launchpad is a web application available in the DIRAC Web Portal which allows to formulate and submit simple jobs.

The job parameters the same as in the job JDL description are entered in the corresponding fields. Use the Add parameters menu to add fields for more parameters. Add any number of files to ship in the job input sandbox by just finding them in your local file system.

Once the job description is complete, press the Submit button to launch the job. You can modify any parameter and submit a new job without restarting from scratch.

Jobs with DIRAC Python API

The DIRAC API is encapsulated in several Python classes designed to be used easily by users to access a large fraction of the DIRAC functionality. Using the API classes it is easy to write small scripts or applications to manage user jobs and data.

While it may be exploited directly by users, the DIRAC API also serves as the interface for the Ganga Grid front-end to perform distributed user analysis for LHCb, for example.

The DIRAC API provide several advantages for the users, those advantages are enumerated below:

- Provides a transparent and secure way for users to submit jobs to the grid.
- Allow to debug locally the programs before be submitted to the Grid.
- A simple, seamless interface to Grid resources allows to run single applications or multiple steps of different applications.
• The user can perform an analysis using understandable Python code.
• Using local job submission the job executable is run locally in exactly the same way (same input, same output) as it will do on the Grid Worker Node. This allows to debug the job in a friendly local environment.
• Using local submission mode the user can check the sanity of the job before submission to the Grid.
• All the DIRAC API commands may also be executed directly from the Python prompt.
• Between others advantages.

Creating a DIRAC Job using API

The API allows creating DIRAC jobs using the Job object, specifying job requirements:

```python
from DIRAC.Interfaces.API.Job import Job
from DIRAC.Interfaces.API.Dirac import Dirac

dirac = Dirac()
j = Job()

j.setCPUTime(500)
j.setExecutable('/bin/echo hello')
j.setExecutable('/bin/hostname')
j.setExecutable('/bin/echo hello again')
j.setName('API')

jobID = dirac.submit(j)
print 'Submission Result:', jobID
```

In this example, the job has three steps from different applications: echo, hostname and echo again.

Submitting jobs

To submit the job is just send the job using the script:

```
$ python testAPI-Submission.py
2010-10-20 12:05:49 UTC testAPI-Submission.py/DiracAPI INFO: <====DIRAC v5r10-pre2====>
2010-10-20 12:05:49 UTC testAPI-Submission.py/DiracAPI INFO: Will submit job to WMS
{'OK': True, 'Value': 196}
```

The script output must return the jobID, this is useful for keeping track of your job IDs.

Job Monitoring

Once you have submitted your jobs to the Grid, a little script can be used to monitor the job status:

```
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job
import sys
dirac = Dirac()
jobid = sys.argv[1]
print dirac.status(jobid)
```

Run it like this:
python Status-API.py <Job_ID>

$python Status-API.py 196 {'OK': True, 'Value': {196: {'Status': 'Done', 'MinorStatus': 'Execution Complete', 'Site': 'LCG.IRES.fr'}}}

The script output is going to return the status, minor status and the site where the job was executed.

Job Output

When the status of the job is done, the outputs can be retrieved using also a simple script:

```python
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job
import sys
dirac = Dirac()
jobid = sys.argv[1]
print dirac.getOutputSandbox(jobid)
```

And, executing the script:

```bash
python Output-API.py <Job_ID>

$ python Output-API.py 196
```

The job output is going to create a directory with the jobID and the output files will be stored inside this directory.

Job Description Language Reference

In this section all the attributes that can be used in the DIRAC JDL job descriptions are presented.
The basic JDL parameters

These are the parameters giving the basic job description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>Name of the executable file</td>
<td>Executable = &quot;/bin/ls&quot;;</td>
</tr>
<tr>
<td>Arguments</td>
<td>String of arguments for the job executable</td>
<td>Arguments = &quot;-ltr&quot;;</td>
</tr>
<tr>
<td>StdError</td>
<td>Name of the file to get the standard error stream of the user application</td>
<td>StdError = &quot;std.err&quot;;</td>
</tr>
<tr>
<td>StdOutput</td>
<td>Name of the file to get the standard output stream of the user application</td>
<td>StdOutput = &quot;std.out&quot;;</td>
</tr>
<tr>
<td>InputSandbox</td>
<td>A list of input sandbox files</td>
<td>InputSandbox = [&quot;jobScript.sh&quot;];</td>
</tr>
<tr>
<td>OutputSandbox</td>
<td>A list of output sandbox files</td>
<td>OutputSandbox = [&quot;std.err&quot;,&quot;std.out&quot;];</td>
</tr>
</tbody>
</table>

Job Requirements

These parameters are interpreted as job requirements

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUTime</td>
<td>Max CPU time required by the job in HEPSPEC06 seconds</td>
<td>CPUTime = 18000;</td>
</tr>
<tr>
<td>Site</td>
<td>Job destination site</td>
<td>Site = [&quot;EGICPPM.fr&quot;];</td>
</tr>
<tr>
<td>BannedSites</td>
<td>Sites where the job must not go</td>
<td>BannedSites = [&quot;EGILAPP.fr&quot;,&quot;EGIM3PEC.fr&quot;];</td>
</tr>
<tr>
<td>Platform</td>
<td>Target Operating System</td>
<td>Platform = &quot;Linux_x86_64_glibc-2.5&quot;;</td>
</tr>
</tbody>
</table>

Data

Describing job data

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputData</td>
<td>Job input data files</td>
<td>InputData = [&quot;/dirac/user/a/atsareg/data1&quot;,&quot;/dirac/user/a/atsareg/data2&quot;];</td>
</tr>
<tr>
<td>OutputData</td>
<td>Job output data files</td>
<td>OutputData = [&quot;output1&quot;,&quot;output2&quot;];</td>
</tr>
<tr>
<td>OutputPath</td>
<td>The output data path in the File Catalog</td>
<td>OutputPath = [&quot;/myjobs/output&quot;];</td>
</tr>
<tr>
<td>OutputSE</td>
<td>The output data Storage Element</td>
<td>OutputSE = [&quot;DIRAC-USER&quot;]</td>
</tr>
</tbody>
</table>

Parametric Jobs

Bulk submission parameters

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>Number of parameters or a list of values</td>
<td>Parameters = 10;</td>
</tr>
<tr>
<td>ParameterStart</td>
<td>Value of the first parameter</td>
<td>ParameterStart = 0;</td>
</tr>
<tr>
<td>ParameterStep</td>
<td>Parameter increment</td>
<td>ParameterStep = 0.1; (default 0.)</td>
</tr>
<tr>
<td>ParameterFactor</td>
<td>Parameter multiplier</td>
<td>ParameterFactor = 1.1; (default 1.)</td>
</tr>
</tbody>
</table>
1.1.4 User Data

Users are managing their data in the distributed computing environment by uploading it to and downloading it from the Storage Elements, replicating files to have redundant copies. The data is accessed from the user jobs, and new data files are created while the job execution. All the files are registered in the File Catalog to be easily discoverable. The basic DIRAC commands to manipulate data are described in this section.

File upload

The initial data file upload to the Grid Storage Element is performed by the following example command:

```
dirac-dms-add-file <LFN> <FILE PATH> <SE> [GUID]
```

where <LFN> is the Logical File Name which will uniquely identify the file on the Grid. <FILE PATH> is the full or relative path to the local file to be uploaded. <SE> is the name of the Storage Element where the file will be uploaded. Optionally <GUID> - unique identifier - can be provided. For example:

```
dirac-dms-add-file /dirac/user/u/username/user.file user.file DIRAC-USER
```

will upload local file `user.file` to the `DIRAC-USER` Storage Element. The file will be registered in the File Catalog with the LFN `/dirac/user/u/username/user.file`

File download

To download a file from the Grid Storage Element one should do:

```
dirac-dms-get-file <LFN>
```

giving the file LFN as the command argument. This will discover the file on the Grid and will download the file to the local current directory.

File replication

To make another copy of the file on a new Storage Element, the following command should be executed:

```
dirac-dms-replicate-lfn <LFN> <SE>
```

This will make a new copy of the file specified by its LFN to the SE Storage Element. For example:

```
dirac-dms-replicate-lfn /dirac/user/u/username/user.file DIRAC-USER
```

You can see all the replicas of the given file by executing:

```
dirac-dms-lfn-replicas <LFN>
```

Finding Storage Elements

You can find all the Storage Elements available in the system by:

```
dirac-dms-show-se-status
```

This will show the Storage Elements together with their current status which will help you to decide which ones you can use.
Data in user jobs

To access data files from the user jobs and make the system save the files produced in the jobs on the Grid, the job description should contain InputData and OutputData parameters. In case of using job JDL description, the JDL can look like the following:

```sh
Executable = "/bin/cp";
Arguments = "my_data.file my_data.copy";
InputData = {"/dirac/user/a/atsareg/my_data.file"};
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err","my.copy"};
OutputData = {"my_data.copy"};
OutputSE = "DIRAC-USER";
CPUTime = 10;
```

For this job execution the input data file with LFN /dirac/user/a/atsareg/my_data.file will be put into the working directory of the user executable. The job will produce a new data file my_data.copy which will be uploaded to the DIRAC-USER Storage Element and registered with LFN (example) /dirac/user/a/atsareg/0/19/my_data.copy. The LFN is constructed using the standard DIRAC user LFN convention (/<vo>/user/<initial>/<username>/) and the job ID to avoid clashes of files with the same name coming from different jobs.

### 1.2 Web Portal Reference

This page is the work in progress. See more material here soon!

#### 1.2.1 Browse Remote Configuration

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Text Actions**
- **Operations**

**Description**

Show Remote Configuration allows the users navigate in a friendly way through the configuration file currently managed by the DIRAC Configuration System.

**Text Actions**

Text actions are provided in the left-side panel, the actions available are:

- **View configuration as text**
  
  This action shows the configuration file in a pop-up window in text format.

- **Download configuration**
  
  Users can use this option to download the configuration file into their local machines.
Operations

In the right side panel the configuration file is exposed using a schema or folders metaphor, in this way the users can expand or collapse folders and sub folders to see the respective attributes and values.

1.2.2 Data Logging Monitor

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- Description
- Selectors
- Columns

Description

The Data Logging Monitor provide information about data logs currently managed by the DIRAC Data Management System. It shows details of the selected files and allows certain logs selection.

Selectors

A text box is located in the text field to introduce the LFN of the file to be showed in the central panel.

Columns

The information on the selected LFN is presented in the central panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

- Status
  - DATA Status
- Minor Status
  - This status complements Status of the file.
- Status Time
- Status
- Source
  - Data source directory

1.2.3 Error Console

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- Description
- Selectors
- Columns
Description

Error Console provide information about Errors reported by DIRAC services and managed by Framework System Logging Report. Details of found errors are showed, also this information can be refined using the available selectors in the left side panel.

Selectors

Selector widgets are provided in the left-side panel. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets. Available selectors in this case are:

Start Date
Date to start Logs selection

Final Date
Date until logs must be showed.

Columns

The information on selected logs is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

The following columns are provided:

Components
DIRAC Component related with the error.

SubSystem
UNKNOWN??

Error
Brief error description.

LogLevel
Log Level associated with the fault, this help to determinate the importance of the error

<table>
<thead>
<tr>
<th>Log Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>The DEBUG Level is a fine-grained event used to debug the service or agent</td>
</tr>
<tr>
<td>INFO</td>
<td>The INFO Level is a coarse-grained event used to show application process</td>
</tr>
<tr>
<td>WARN</td>
<td>The WARN Level show warns about future possible errors in the service or agent</td>
</tr>
<tr>
<td>ERROR</td>
<td>The ERROR Level show errors occurred, the services or agents can still run</td>
</tr>
<tr>
<td>FATAL</td>
<td>The FATAL Level show errors than makes service or agent stop</td>
</tr>
</tbody>
</table>

SiteName
Site names associated with the error.

Example
Shows one error log entry.

**OwnerDN**

Distinguish name of the entity associated with the error.

**OwnerGroup**

DIRAC group associated with the error.

**IP**

Server IP Address associated with the error.

**Message Time**

UTC time stamp in the log file when the error was reported.

**Number of errors**

Number of error occurrences.

### 1.2.4 Job Monitoring

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Selectors**
- **Columns**
- **Operations**
- **Actions**

**Description**

The Job Monitoring is the most accessed page of the DIRAC Web Portal, provide information about User Jobs managed by the DIRAC Workload Management System. It shows details of the selected Jobs and allows certain Job selections.

**Selectors**

Selector widgets are provided in the accordion menu left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

The following selectors are available:

**Site**

The Pilot Job destination site using DIRAC nomenclature.

**Status**
Currently status of the job. The following values of status are possible:

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waiting</td>
<td>Job is accepted for DIRAC WMS</td>
</tr>
<tr>
<td>Scheduled</td>
<td>Job is assigned to a Site</td>
</tr>
<tr>
<td>Running</td>
<td>Job has started running in the CE</td>
</tr>
<tr>
<td>Done</td>
<td>Job finished successfully</td>
</tr>
<tr>
<td>Deleted</td>
<td>Job deleted by the user</td>
</tr>
<tr>
<td>Killed</td>
<td>Job killed by the user</td>
</tr>
</tbody>
</table>

**Minor Status**

Minor status complement the Job status, creating a complete sentence to have a better comprehension of the status.

<table>
<thead>
<tr>
<th>Minor Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Finished with Error</td>
<td>Job finished but with errors during application execution.</td>
</tr>
<tr>
<td>Execution Complete</td>
<td>Job successfully finished.</td>
</tr>
<tr>
<td>Marked for Termination</td>
<td>Job marked by the user for termination.</td>
</tr>
<tr>
<td>Maximum of Rescheduling reached</td>
<td>Job can be rescheduled a number of predefined times and this number was reached.</td>
</tr>
<tr>
<td>Pilot Agent Submission</td>
<td>Job is Waiting until a pilot job being available.</td>
</tr>
<tr>
<td>Matched</td>
<td>Job is assigned to a pilot job.</td>
</tr>
</tbody>
</table>

**Application Status**

With this information the user can know what kind of problem occurs during execution of the application.

<table>
<thead>
<tr>
<th>Application Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed Input Sandbox Download</td>
<td>Job failed to download Input Sandbox.</td>
</tr>
<tr>
<td>Unknown</td>
<td>Job failed by a unknown reason.</td>
</tr>
</tbody>
</table>

**Owner**

The Job Owner. This is the nickname corresponding to the Owner grid certificate distinguish name.

**JobGroup**

The Job Owner group using during job submission.

**JobID**

Number or list of numbers, of jobs selected.

**Global Sort**

This option is available in the accordion menu in the left panel. Allow users to sort jobs information showed in the right side panel. Available possibilities are:

- JobID Ascending
- JobID Descending
- LastUpdate Ascending
- LastUpdate Descending
- Site Ascending
Current Statistics

This option is available in the accordion menu in the left panel, and show statistics of jobs selected, as status and number, in a table in the same panel. The columns presented are:

**Status**

Job status, in this case: Done, Failed, Killed, Waiting.

**Number**

Total number of jobs in the related status.

Global Statistics

This option is available in the accordion menu in the left panel, and show statistics of all of jobs in the system, as status and number, in a table in the same panel.

**Status**

Job status, in this case: Done, Failed, Killed, Waiting.

**Number**

Number of total jobs.

Columns

The information on the selected Jobs is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

The following columns are provided:

**JobID**

JobID in DIRAC nomenclature.

**Status**

Job status.

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waiting</td>
<td>Job is accepted for DIRAC WMS</td>
</tr>
<tr>
<td>Scheduled</td>
<td>Job is assigned to a pilot job to be executed.</td>
</tr>
<tr>
<td>Running</td>
<td>Job was started and is running into CE</td>
</tr>
<tr>
<td>Done</td>
<td>Job finished successfully</td>
</tr>
<tr>
<td>Deleted</td>
<td>Job marked by the user for deletion</td>
</tr>
<tr>
<td>Killed</td>
<td>Job is marked for kill</td>
</tr>
</tbody>
</table>

Minor Status
Complement Job Status.

<table>
<thead>
<tr>
<th>Minor Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Finished with Error</td>
<td>Job finished but with errors during execution.</td>
</tr>
<tr>
<td>Execution Complete</td>
<td>Job successfully finished.</td>
</tr>
<tr>
<td>Marked for Termination</td>
<td>Job marked by the user for termination.</td>
</tr>
<tr>
<td>Maximum of Rescheduling reached</td>
<td>Job can be rescheduled a number of predefined times.</td>
</tr>
<tr>
<td>Pilot Agent Submission</td>
<td>Job is Waiting until a pilot job be available.</td>
</tr>
<tr>
<td>Matched</td>
<td>Job is assigned to a pilot job.</td>
</tr>
</tbody>
</table>

Application Status.

Site

The Job destination site in DIRAC nomenclature.

JobName

Job Name assigned by the User.

Owner

Job Owner. This is the nickname of the Job Owner corresponding to the users certificate distinguish name.

LastUpdateTime

Job last status update time stamp (UTC)

LastSingofLife

Time stamp (UTC) of last sign of life of the Job.

SubmissionTime

Time stamp (UTC) when the job was submitted.

Operations

Clicking on the line corresponding to a Job, one can obtain a menu which allows certain operations on the Job. Currently, the following operations are available.

JDL

Job JDL into DIRAC nomenclature.

Attributes

Job Attributes associated with the job, owner, priority, etc.

Parameters

Parameters of the site where the job ran or is running.

LoggingInfo

Get Job information in a pop-up panel about each status where the job has been.

PeekStandartOutput

Get the standard output of the Job in a pop-up panel.
 actions that the user can perform over their jobs are showed below:

<table>
<thead>
<tr>
<th>Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Restart the Job</td>
</tr>
<tr>
<td>Kill</td>
<td>Kill the Job selected</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete the job</td>
</tr>
</tbody>
</table>

### 1.2.5 Manage Proxies

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Columns**
- **Operations**

**Description**

Manage Proxies provide information about User Proxies currently managed by the DIRAC Framework System. Users have a different proxy for each group than him/her belong. This pages shows all the details associated to each user proxy.

**Columns**

The information is deployed in the main panel in a form of a table. The columns available in this page are:

- **User**
  User nickname following DIRAC nomenclature.

- **DN**
  User certificate distinguish name.

- **Group**
  DIRAC user group associated with the proxy.

- **ExpirationDate(UTC)**
  Date until user certificate is valid.

- **Persistent**
  Show if a proxy is persistent (value=true) or not (value=false).
You can choose to display the proxies by group or grouping by field choosing them in the menu, activated by pressing on a menu button.

**Operations**

The only operation that the user can perform over proxies is to delete them. The user can select one or more proxies into the main panel or using the top bar check box to select all of them, and after click in the delete button. Also is available the option *select none* proxy.

**1.2.6 Manage Remote Configuration**

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Text Actions**
- **Modification Actions**
- **Operations**

**Description**

Show Remote Configuration allows administrators navigate in a friendly way through the configuration file, the configuration of the servers is managed by DIRAC Configuration System.

**Text Actions**

Text actions are provided in the left-side panel, in this moment just two options are available:

- **View configuration as text**
  
  This action shows the configuration file in text format in a pop-up window.

- **Download configuration**
  
  This action permit download the configuration file to local machine.

**Modification Actions**

Modification actions are provided in the left-side panel, the available modifications are:

- **Re-download configuration data from server**
  
  Allows DIRAC administrators to update or download again, depending of the case, the configuration used the server into the web browser.

- **Show differences with server**
  
  This option shows the differences between file loaded into web browser and the file used currently by the server.

- **Commit configuration**
  
  Allow DIRAC Administrator to commit a new configuration file into the server.
Operations

In the right side panel the configuration file is exposed using a schema or folder metaphor, this metaphor allows DIRAC Administrators to expand or collapse each folder and sub folders in order to look at, add, remove or change the attributes and respective values into the configuration file. After any modification of the configuration file is mandatory to commit the configuration file, executing this action the new configuration file is copied to the server, the service is restarted and loaded into the system.

1.2.7 Overview

DIRAC Web Portal is a Web application which provides access to all the aspects of the DIRAC distributed computing system. It allows to monitor and control all the activities in a natural desktop application like way. In order to reach this goal DIRAC Web Portal is built using GUI elements to mimic desktop applications, such as toolbars, menus, windows buttons and so on.

Description

All pages have two toolbars, one on the top and another at the bottom of the pages that contain the main navigation widgets. The top toolbar contains the main menu and reflects the logical structure of the Portal. It also allows to select active DIRAC setup. The bottom toolbar allows users to select their active group and displays the identity the user is connected with.

The mostly used layout within our Web Portal is a table on the right side of the page and a side bar on the left. Almost all data that needs to be displayed can be represented as two-dimensional matrix using a table widget. This widget has a built-in pagination mechanism and is very customizable. As a drawback, it is a bit slow to load the data into the table. On an average desktop hardware, tables with more than 100 elements can be slow to display the data.

1. Main Menu: This menu offers options for systems, jobs, tools and help.
2. Selections: Shows a set of selectors than permits generate customs selections.
3. Buttons to open/collapse panels: Permit open or collapse left menu.
4. Actions to perform for job(s): These actions permits select all, select none, reset, kill or submit
5. Menu to change DIRAC setup: Users can change between different setups.
6. Current location: Indicates where the user is located inside the portal.
7. Buttons to submit or reset the form: After options are selected its possible to submit and execute the selection or reset the selectors.
8. Pagination controls: Permits navigate between the pages, and also show in which page the user is navigating.
9. Refresh table: Reload the page without loose the previous selection and show the new status.
10. Items per page: This option allow the users to specify how many items are going to be displayed by page.
11. User DIRAC login: Login assigned to the user connected to DIRAC web portal.
12. DIRAC Group: The user could belong to different groups and perform actions depending of the group previously selected.
13. Certificate DN: Web portal shows the distinguish name of user certificate what is being used to realize the connection.
14. Index items displayed: Display the range of items displayed in the page.

Note: Some options are not displayed in all Web Portal pages, as selections.
Fig. 1.1: DIRAC Web Portal
Functionalities

DIRAC Web Portal is a Web based User Interface than provide several actions depending of each group and privileges of the user into DIRAC. Actions by user privileges are showed below:

- **Users**: Track jobs and data, perform actions on jobs as killing or deleting.
- **Production Managers**: Can define and follow large data productions and react if necessary starting or stopping them.
- **Data Managers**: Allows to define and monitor file transfer activity as well as check requests set by jobs.
- **Administrators**: Can manage, browse, watch logs from servers.

1.2.8 Pilot Monitor

This is part of the DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Selectors**
- **Columns**
- **Operations**

Description

The Pilot Monitor is providing information about the Pilot Jobs currently managed by the DIRAC Workload Management System. It shows details of the selected Pilot Jobs and allows certain Pilot Job manipulations.

Selectors

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

The following Selectors are available:

- **Status**  The Pilot Job Status. The following Status values are possible:
  - **Submitted**: Pilot Job is submitted to the grid WMS, its grid status is not yet obtained
  - **Ready**: Pilot Job is accepted by the grid WMS
  - **Scheduled**: Pilot Job is assigned to a grid site
  - **Running**: Pilot Job has started running at a grid site
  - **Stalled**: Pilot Job is stuck in the grid WMS without further advancement, this is typically an indication of the WMS error
  - **Done**: Pilot Job is finished by the grid WMS
  - **Aborted**: Pilot Job is aborted by the grid WMS
  - **Deleted**: Pilot Job is marked for deletion

- **Site**  The Pilot Job destination site in DIRAC nomenclature.

- **ComputingElement**  The end point of the Pilot Job Computing Element.

- **Owner**  The Pilot Job Owner. This is the nickname of the Pilot Job Owner corresponding to the Owner grid certificate DN.

- **OwnerGroup**  The Pilot Job Owner group. This usually corresponds to the Owner VOMS role.

- **Broker**  The instance of the WMS broker that was used to submit the Pilot Job.
**Time Span**  The Time Span widget allows to select Pilot Jobs with Last Update timestamp in the specified time range.

**Columns**

The information on the selected Pilot Jobs is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button ( small triangle ) in any column title field.

The following columns are provided:

- **PilotJobReference**  Pilot Job grid WMS reference.
- **Site**  The Pilot Job destination site in DIRAC nomenclature.
- **ComputingElement**  The end point of the Pilot Job Computing Element.
- **Broker**  The instance of the WMS broker that was used to submit the Pilot Job.
- **Owner**  The Pilot Job Owner. This is the nickname of the Pilot Job Owner corresponding to the Owner grid certificate DN.
- **OwnerDN**  The Pilot Job Owner grid certificate DN.
- **OwnerGroup**  The Pilot Job Owner group. This usually corresponds to the Owner VOMS role.
- **CurrentJobID**  The ID of the current job in the DIRAC WMS executed by the Pilot Job.
- **GridType**  The type of the middleware of the grid to which the Pilot Job is sent
- **Benchmark**  Estimation of the power of the Worker Node CPU which is running the Pilot Job. If 0, the estimation was not possible.
- **TaskQueueID**  Internal DIRAC WMS identifier of the Task Queue for which the Pilot Job is sent.
- **PilotID**  Internal DIRAC WMS Pilot Job identifier
- **ParentID**  Internal DIRAC WMS identifier of the parent of the Pilot Job in case of bulk (parameteric) job submission
- **SubmissionTime**  Pilot Job submission time stamp
- **LastUpdateTime**  Pilot Job last status update time stamp

**Operations**

Clicking on the line corresponding to a Pilot Job, one can obtain a menu which allows certain operations on the Pilot Job. Currently, the following operations are available.

- **Show Jobs**  Pass to a Job Monitor and select jobs attempted to be executed by the given Pilot Job
- **PilotOutput**  Get the standard output of the finished Pilot Job in a pop-up panel. Note that only successfully finished Pilot Jobs output can be accessed.

**1.2.9 Pilot Summary**

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Selectors**
- **Statistics**
• Columns
• Operations

Description

Pilot summary present a table with statistics of all pilots assigned by sites and sites efficiency this information give to the user the possibility to choose sites to submit their jobs according this values. This service is currently managed by the DIRAC Workload Management System.

Selectors

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

Sites

Allows the user to select one or various sites.

Statistics

General statistics are provided in the left-side panel, statistics showed are a summary over all the sites where DIRAC can run pilots jobs.

Scheduled

Number of pilot jobs in status scheduled in all the sites.

Status

Summary status of all the sites.

Aborted_Hour

Number of pilot jobs aborted in all the sites in the last hour.

Waiting

Number of pilot jobs in status waiting in all the sites.

Submitted

Total number of pilots submitted last hour.

PilotsPerJob

Number of pilots required to run a user job.

Ready

Total number of pilots in status ready.

Running

Total number of pilots running over all the sites.

PilotJobEff(%)

Percentage of pilots jobs finished whose status is done.

Done
Total number of pilot jobs whose status is done.

**Aborted**
Total number of pilot jobs aborted.

**Done_Empty**
Total number of pilot jobs in status done but **without output**.

**Total**
Total number of pilots.

**Columns**

The information on the selected sites is presented in the right-side panel in a form of a table.

**Site**
Site Name in DIRAC nomenclature.

**CE**
Site Computing Element name.

**Status**
General status of the site depending of pilot effectiveness.

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Site effectiveness less than 25% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Poor</td>
<td>Site effectiveness less than 60% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Fair</td>
<td>Site effectiveness less than 85% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Good</td>
<td>Site effectiveness more than 85% of pilot jobs executed successfully</td>
</tr>
</tbody>
</table>

**PilotJobEff(%)**
Percentage of pilots successful ran in the site.

**PilotsPerJob**
Number of pilot jobs required to execute an User Job.

**Waiting**
Number of pilot jobs waiting to be executed.

**Scheduled**
Number of pilot jobs scheduled in a particular site.

**Running**
Number of pilot jobs running in the site.

**Done**
Number of pilot jobs executed successfully in the site.

**Aborted_Hour**
Number of pilots aborted the last hour in the site.
Operations

Clicking on the line corresponding to a Site, one can obtain a menu which allows certain operations on Site Pilots Jobs. Currently, the following operations are available.

Show Pilots

Show in the right side panel all the Pilots Jobs related with the site.

Show Value

Show the value of the cell in a pop-up window.

1.2.10 Production Monitor

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- Description
- Selectors
- Current Statistics
- Global Statistics
- Columns
- Operations

Description

Production Monitoring, provide information about Productions managed by the DIRAC *Workload Management System*. It shows details of the selected production and allows users to refine certain selections.

Selectors

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

Status

Allow select production depending of status, the possible status of selections are:
Current Statistics

This option is available in the left panel, shows production statistics based on currently selected productions, resultant information is showed in a table in the same panel.

Global Statistics

This option is available in the left panel, and shows global statistics about all productions in a table in the same panel.

Columns

The information on the selected productions is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

<table>
<thead>
<tr>
<th>ID</th>
<th>DIRAC Production ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Production Status.</td>
</tr>
<tr>
<td>Agent Type</td>
<td>How the agent was submit: Automatic or Manual</td>
</tr>
<tr>
<td>Type</td>
<td>Production Type, by example: MCSimulation.</td>
</tr>
<tr>
<td>Group</td>
<td>DIRAC group of the user than submit the production.</td>
</tr>
<tr>
<td>Name</td>
<td>Production name.</td>
</tr>
<tr>
<td>Files</td>
<td>Number of files required to run the production.</td>
</tr>
</tbody>
</table>
| Processed(%) | }
Percentage of completeness of the production. It can be 0 in case the production can be extended.

**Files Processed**
Number of files processed until now.

**Files Assigned**
Number of files to be processed.

**Files Problematic**
??

**Files Unused**
Number of failed files in case production fail, it was sent but not processed.

**Created**
Number of jobs created to run the production.

**Submitted**
Number of jobs submitted to different sites.

**Waiting**
Number of jobs in status waiting.

**Running**
Number of jobs running.

**Done**
Number of jobs in status done.

**Failed**
Number of jobs failed.

**Stalled**
Number of jobs stalled.

**InheritedFrom**
?? production ID

**GroupSize**

**FileMask**

**Plugin**

**EventsPerJob**

**MaxNumberOfJobs**
Maximum number of jobs to be submitted for the selected production.

### Operations

Clicking on the line corresponding to a Production, one can obtain a menu which allows certain operations on the production. Currently, the following operations are available.

**Show Jobs**
Show associated jobs with the selected production.

**LoggingInfo**
Show logging info for the selected production.

**FileStatus**
Show Details
Details about the production selected

**Actions**
Actions can be done using the selectors and buttons in the title field, the options are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Start the production</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop the production</td>
</tr>
<tr>
<td>Flush</td>
<td>Flush the production</td>
</tr>
<tr>
<td>Clean</td>
<td>Clean</td>
</tr>
</tbody>
</table>

**Show Value**
Show value of selected cell.

### 1.2.11 Proxy Action Logs

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Columns**
- **Filters**

**Description**

Proxy Action Logs page present on a table each operation related with proxies, related with users, hosts or services, into DIRAC system.

**Columns**

**Timestamp (UTC)**
Time stamp (UTC) when the operation was executed.

**Action**
Describe the action executed using the proxy, by example: store proxy, download voms proxy, set persistent proxy.

**IssuerDN**
Certificate Distinguish Name of the entity who request perform the operation.

**IssuerGroup**
DIRAC group associated with IssuerDN who is requesting the operation.

**TargetDN**
Distinguish Name of Certificate entity who request to perform the operation.

**TargetGroup**

DIRAC group associated with the TargetDN over whom the operation is performed.

**Filters**

Filters allow the user to refine logs selection according to one or more attributes. Filters are available as a combination of a menu that appears clicking into a log row and options available in the bottom field, filters available are described below:

The menu shows options are:

- **Filter by action**
  - Depending on the value of the log that was clicked, a filter will be created.

- **Filter by issuer DN**
  - Depending on the value of the log that was clicked, a filter will be created.

- **Filter by target DN**
  - Depending on the value of the log that was clicked, a filter will be created.

- **Filter by target group**
  - Depending on the value of the log that was clicked, a filter will be created.

At the bottom field appears the following items:

- **Page Manager**
  - Allow the user to navigate through all the log pages.

- **Refresh button**
  - This button user to refresh the page in fly time and apply the filter to the logs.

- **Items displaying per page**
  - Deploy a menu that presents options of 25, 50, 100, 150 actions by page

- **After**
  - Show the logs actions performed after the date selected.

- **Before**
  - Show the logs actions performed before the date selected.

- **Filters**
  - Show selected filters to perform the action.

- **Clear Filters**
  - This button clears the filters used in the previous selection.

**NOTE:** To perform any filtering action must be pressed the refresh button in the bottom field.
1.2.12 RAW Integrity

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Selectors**
- **Global Sort**
- **Current Statistics**
- **Global Statistics**
- **Columns**
- **Operations**

**Description**

The RAW Integrity provide information about files currently managed by the DIRAC Data Management System. It shows details of the selected files and allows certain file selection.

**Selectors**

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

The following Selectors are available:

- **Status**
  Status of the file.
- **Storage Element**
  Name of Storage Element.
- **Time Start**
  Time Start to look stored files
- **Time End**
  Time end to look stored files
- **LFN**
  Logical file name.

**Global Sort**

This selector allows the users sort the files using one of the options showed below:

- Start Time
- End Time
- Status Ascending
• Status Descending
• Storage Ascending
• Storage Descending
• LFN

Current Statistics

Show status and numbers of selected files. The possible values of status are:

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td></td>
</tr>
<tr>
<td>Failed</td>
<td></td>
</tr>
</tbody>
</table>

Global Statistics

Show status and numbers in a global way. The possible values of status are:

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td></td>
</tr>
<tr>
<td>Failed</td>
<td></td>
</tr>
</tbody>
</table>

Columns

The information on the selected file is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

The following columns are provided:

LFN
Logical file name.

Status
Status of the file.

Site
Site name using DIRAC convention.

Storage Element
Storage Element name using DIRAC convention where the file is stored.

Checksum
Value of the checksum file which is also calculated at the original write time at the Online storage. If the two checksums match the integrity of the file in CASTOR can be assumed.

PFN
Physical File name.
Start Time (UTC)
End Time (UTC)
GUI

Operations

Clicking on the line corresponding to a file, one can obtain a menu which allows certain operations on the raw integrity. Currently, the following operations are available:

Logging Info

- Shows information about the file selected.
  - Status:
  - Minor Status:
  - Start Time: Start time
  - Source: File directory source.

Show Value

- Show the value of the cell.

1.2.13 History of Server Changes

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- Description
- Selectors
- Columns
- Operations

Description

This page provide information to DIRAC administrators about changes made to the server configuration file, showing historical files and who commit each file.

Selectors

Show differences between selected

- Show the differences between two configuration files selected from right side panel in an pop-up window.

RollBack to “TO” version

- Change server configuration to the configuration file selected into column TO/RB right side panel.
Columns

The information of historical configuration files is presented in the right-side panel in a form of a table. Available columns are:

- **From**
  - Selector button
- **TO/RB**
  - Selector button
- **Version**
  - Configuration file version number using DIRAC nomenclature.
- **Committer**
  - User who commit the configuration file.

Operations

Operations available into this page are:

- **Show Configuration File**
  - Show the selected configuration file in a pop-up window.
- **Show Value**
  - Show the value of selected cell.

1.2.14 Sites Summary

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- **Description**
- **Selectors**
- **Columns**
- **Operations**

Description

Site Summary provide information about Sites managed by the DIRAC *Workload Management System*. It shows details of the selected Sites and allows certain selections.

Selectors

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.
GridType
MaskStatus
Country

Columns

Tier
Show the Tier associated with the site.

GridType
Grid type of the site, by example: DIRAC, gLite.

Country
Country where the site is located.

MaskStatus
Mask status of the site, it can take two values: Allowed or Banned

Efficiency (%)
Site percentage of efficiency, the values associated are:

<table>
<thead>
<tr>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad</td>
<td>Site effectiveness less than 25% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Poor</td>
<td>Site effectiveness less than 60% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Fair</td>
<td>Site effectiveness less than 85% of pilot jobs executed successfully</td>
</tr>
<tr>
<td>Good</td>
<td>Site effectiveness more than 85% of pilot jobs executed successfully</td>
</tr>
</tbody>
</table>

Received
Number of Pilots Jobs such status is Received in the site.

Checking
Number of Pilots Jobs such status is Checking in the site.

Staging
Number of Pilots Jobs such status is Staging in the site.

Waiting
Number of Pilots Jobs such status is Waiting in the site.

Matched
Number of Pilots Jobs such status is Matched in the site.

Running
Number of Pilots Jobs such status is Running in the site.

Completed
Number of Pilots Jobs such status is Completed in the site.

Done
Number of Pilots Jobs such status is Done in the site.
Stalled

Number of Pilots Jobs such status is Stalled in the site.

Failed

Number of Pilots Jobs such status is Failed in the site.

Operations

1.2.15 Storage Directory Summary

This is part of DIRAC Web Portal project. For the description of the DIRAC Web Portal basic functionality look here.

- Description
- Selectors
- Usage
- Columns

Description

Storage Directory Summary provide information about Storage Directories of users currently managed by the DIRAC Data Management. It shows details of the selected storage directories.

Selectors

Selector widgets are provided in the left-side panel. These are drop-down lists with values that can be selected. A single or several values can be chosen. Once the selection is done press Submit button to refresh the contents of the table in the right-side panel. Use Reset button to clean up the values in all the selector widgets.

The following Selectors are available:

Production

Production name to be selected.

FileType

File type

Directory

Directory where the user storage the files.

SEs

List of Storage Elements that the user has available for use.

Usage

Usage in the left-side panel shows storage information as:

SE

Name of Storage element used to store the file in DIRAC convention.
Replicas

Number of file replicas.

Size

Size of files stored.

Columns

The information on the selected Storage Directory Summary is presented in the right-side panel in a form of a table. Note that not all the available columns are displayed by default. You can choose extra columns to display by choosing them in the menu activated by pressing on a menu button (small triangle) in any column title field.

The following columns are provided:

Directory Path

Directory path where the user files are stored.

Replicas

Number of replicas of the file.

Size

File size.

1.3 Web Portal User guide

The DIRAC Web portal is a user friendly interface allowing users to interact with the DIRAC services. It can be easily extended by particular VO or it can be integrated into some other portal.

1.3.1 Terms:

Application

A web page called application in the new portal, for example: Monitoring, Accounting, Production Management.

Desktop

It is a container of different applications. Each application opens in a desktop. The desktop is your working environment.

State

The State is the actual status of an application or a desktop. The State can be saved and it can be reused. A saved State can be shared within the VO or between users.

Theme

It is a graphical appearance of the web portal. DIRAC provides two themes: Desktop and Tab themes. Both themes provide similar functionalities. The difference is the way of how the applications are managed. The “Desktop theme” is similar to Microsoft Windows. It allows to work with a single desktop. The “Tab theme” is similar to web browser. Each desktop is a tab. The users can work with different desktops at the same time.
1. **Concepts:**

Two protocols are allowed: **http** and **https**. **http** protocol is very restricted. It only allows to access limited functionalities. It is recommended to the site administrators. The state of applications or desktops can not be saved. **https** protocol allows to access all functionalities of DIRAC depending on your role (DIRAC group). The state of the application is not saved in the URL. The URL only contains the name of application or desktop. For example: 

```
https://lhcb-portal-dirac.cern.ch/DIRAC/s:LHCb-Production/g:lhcb_prmgr/?view=tabs&theme=Grey&url_state=1|AllPlots
```

**Format of the URL**

- **Tab theme:**

  **Format of the URL when the Tab theme is used:**

  1. **https://** protocol
  2. lhcb-portal-dirac.cern.ch/DIRAC/: host.
  3. s:LHCb-Production: DIRAC setup.
  4. g:lhcb_prmgr : role
  5. view=tabs : it is the theme. It can be desktop and tabs.
  6. theme=Grey: it is the look and feel.
  7. &url_state=1: it is desktop or application.
  8. AllPlots : it is the desktop name. the default desktop is Default.
  9. The state is a desktop: AllPlots
  10. The state is an application: LHCbDIRAC.LHCbJobMonitor.classes.LHCbJobMonitor:AllUserJobs,

For example: desktop and application: AllPlots,*LHCbDIRAC.LHCbJobMonitor.classes.LHCbJobMonitor:AllUserJobs,*

- **Desktop theme**

  **For example:**

  ```
  https://lhcb-portal-dirac.cern.ch/DIRAC/s:LHCb-Production/g:lhcb_prmgr/?view=desktop&theme=Grey&url_state=1|AllPlots
  ```

**Note:** If you have a state saved under Desktop theme, you can open using Tab theme. This works the other way round as well.

A video tutorial is available at [https://www.youtube.com/watch?v=vKBpED0IyLc](https://www.youtube.com/watch?v=vKBpED0IyLc) link.
Tab theme

In this section a detailed description of the Tab theme is presented:

- **Main panel**
- **Menu structure**
- **Manage application and desktop**
- **Share application and desktop**
- **Settings panel**

Main panel

The main panel consists of two widget:

1. Menu
2. Desktop

Menu

It contains three main menu:

1. It is the Intro panel
2. It is the Main panel
3. You can found more information about DIRAC.

The default is 2. You can change by clicking on the icons.

Desktop

It is a container which contains various applications on different desktops.

Menu structure

Menu consists of two widgets:

1. Desksops&Applications
2. Settings
Desktop&Applications

You can manage your applications and desktops. The menu structure:

- Web: it contains external links.
- Tools: You can found DIRAC specific applications.
- Applications: You can found DIRAC and VO specific applications.
- OldPortal: It is link to the old portal.
- DIRAC: it is an external link to DIRAC portal.
- My Desktops: it is contains all saved desktops. You can see a Default desktop which contains all applications which belongs to the Default desktop.
- Shared: It contains all Shared desktops and applications.

Manage application and desktop

You can manage the state of applications and desktops by clicking to the following menu.
Desktop

The Desktop menu item contains:

- New Desktop: You can create an empty desktop.
- Save: You can save the desktop
- Save As: you can duplicate your desktop.
- Delete: You can delete different desktops.

If you click on the delete menu item, a pop up window will appear:

You can select the desktops to be deleted.

Application

The Application menu item contains:

- Save
- Save As
- Delete

These menu items have the same functionalities as the Desktop menu items.

Context menu

You have another possibility to manage applications and desktops. You have to right click on the application/desktop what you want to modify.
You have few additional menu items:

- Make public: Used to make public an application/desktop to everyone.
- Share desktop: Used to share the desktop within a specific user.
- Share application: Used to share the application within a specific user.
- Make private: Revoke the access to the desktop/application.
- Switch to presenter view: The applications will be open in a single desktop.
- Switch to tab view: The applications opened in different tabs.

Presenter view

The application which belongs to a desktop will be opened in a single tab. You can change the layout of the desktop using the buttons in the right corner of the panel (The buttons are in the red rectangle).

Tab view

The applications within a desktop will be opened in different tab.

In the right corner of the Tab theme you can see two icons.

First icon You can access to a specific application by clicking on the first icon. This is very useful when you have lot of application open in a desktop.
Second icon You can write help to the current application.

Share application and desktop

The applications/desktops can be shared. You can share an application/desktop by right click on the application/desktop what you want to share (more information above in the Manage application and desktop).

Share an application/desktop

You have to do the following steps to share an application/desktop:

1. right click on the desktop/application what you want to share.
2. choose the menu item: Share desktop or Share Application.
3. copy the text (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and click OK on the pop up window:
4. send the text (desktop|zmathe|lhcb_prmgr|JobMonitorAll) to the person

Load a shared application or desktop

You have to use the State Loader menu item:
The State Loader widget is the following:

You have to provide the Shared State (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and a name (for example: newName). You have three different ways to load a shared state:

1. Load
2. Create Link
3. Load & Create Link

**Load**

If you click on Load, you load the shared desktop/application to your desktop. The name of the application will be the provided name. For example: newName.

**Create Link**

This saves the application/desktop *Shared* menu item. Which means it keeps a pointer (reference) to the original desktop/application. This will not load the application/desktop into your desktop.

**Load & Create Link**

The desktop/application will be loaded to your desktop and it is saved under the *Shared* menu item.

**Delete shared applications/desktops**

You have to click on the menu *Manage application and desktop* and then select application or desktop depending what you want to delete. For example: Let’s delete the *newName* shared desktop.
You have to select what you want to delete state or a link. As it is a shared desktop what we want to delete we have to select *Links*. You have to click on the Delete button.

**Settings panel**

In the settings panel you can set up your portal. You have to click on the *Settings* widget:

You can define the following:

- Group you can change the role
- Setup: you can switch between different setups.
- Theme you can change the look and feel and also you can switch between Tab and Desktop themes.

**We have 3 look and feels:**

1. Grey it is the default
2. Neptune
3. Classic

You can automatically change the applications using *Automatic tab change* Note: After you set it you have to save the desktop. Consequently, you can not have automatic tab change in the *Default* desktop.

**Grey**

**Neptune**

**Classic**

### Desktop theme

In this section a detailed description of the Desktop theme is presented:

- *Main widget*
- *Menu structure*
- *Manage application and desktop*
- *Share application and desktop*

**Main widget**

When you open the web portal you will get an empty desktop.
In the left corner you can see an icon, which is the menu.

In the right corner you can see the settings.

You can define the following:

- You can switch between Tab and Desktop themes.
- Group you can change the role
- Setup: you can switch between different setups.

Menu structure

The menu structure:

- Web : it contains external links
- Tools : You can found DIRAC specific applications.
- Applications: You can found DIRAC and VO specific applications.
- OldPortal: It is link to the old portal.
- DIRAC it is an external link to DIRAC portal
- State Loader: It is used to load a state.
The states of the applications are available when you click on the application name.

The end of the list you can see the shared states of the selected application (You can see in the previous picture, indicated by red rectangle).

There is an other context menu which is available by right click on the desktop or on the task bar.
Manage application and desktop

Applications

You can manage the applications in two different ways.

**First way:** Each application has an associated menu:

- **First icon:**
  1. Load state: We can apply a state to the open application.
  2. Save: We can save the application.
  3. Save As...: We can duplicate the application.
  4. Refresh states: We can refresh the states.
  5. Manage states... We can delete the state or shared states.

- **Second icon:** We can pin and unpin an application. It is used to create a customized desktop.

- **Third icon:** We can hide the application

- **Fourth icon:** You can write help to the current application. The rest icons are the usual icons: minimize, maximize and exit.

**Second way:** We have to click on the application icon which is on the task bar.
The menu is equivalent to previous menu.

**Desktops**

You have to right click on the task bar to manage the desktops. The menu items have similar functionality than the application described above.

**Share application and desktop**

**Share an application/desktop**

You have to open the main menu more details: *Menu structure*

You have to do:

1. click on the menu item: Share
2. copy the text (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and click OK on the pop up window:
3. send the text (desktop|zmathe|lhcb_prmgr|JobMonitorAll) to the person

**Load a shared application or desktop**

You have to use the *State Loader* menu item more details:*Menu structure*

The State Loader widget is the following:
You have to provide the Shared State (for example: desktop/lzmathelhc_prmgrJobMonitorAll) and a name (for example: newName). You have three different ways to load a shared state:

1. Load
2. Create Link
3. Load & Create Link

Load

If you click on Load, you load the shared desktop/application to your desktop. The name of the application will be the provided name. For example: newName.

Create Link

This saves the application/desktop Shared menu item. Which means it keeps a pointer (reference) to the original desktop/application. This will not load the application/desktop into your desktop.

Load & Create Link

The desktop/application will be loaded to your desktop and it is saved under the Shared menu item.

Delete shared applications/desktops
You have to click on the Manage states... menu more details Manage application and desktop and then select application or desktop depending what you want to delete. For example: Let’s delete the newName shared desktop.

![Image of Manage states dialog]

You have to select what you want to delete state or a link. As it is a shared desktop what we want to delete we have to select Links. You have to click on the Delete button.

## 1.4 Commands Reference

This page is the work in progress. See more material here soon!

### 1.4.1 Data Management Command Reference

In this subsection the Data Management commands are collected.

**dirac-dms-add-file**

Upload a file to the grid storage and register it in the File Catalog

Usage:

```
dirac-dms-add-file [option|cfgfile] ... LFN Path SE [GUID]
```

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFN</td>
<td>Logical File Name</td>
</tr>
<tr>
<td>Path</td>
<td>Local path to the file</td>
</tr>
<tr>
<td>SE</td>
<td>DIRAC Storage Element</td>
</tr>
<tr>
<td>GUID</td>
<td>GUID to use in the registration (optional)</td>
</tr>
</tbody>
</table>

Usage:

```
dirac-dms-add-file [option|cfgfile] ... LocalFile
```

Arguments:
LocalFile: Path to local file containing all the above, i.e.:

lfn1 localfile1 SE [GUID1]
lfn2 localfile2 SE [GUID2]

Example:

```bash
$ dirac-dms-add-file LFN:/formation/user/v/vhamar/Example.txt Example.txt DIRAC-USER
{'Failed': {},
 'Successful': {'/formationes/user/v/vhamar/Example.txt': {'put': 0.70791220664978027,
 'register': 0.61061787605285645}}}
```

**dirac-dms-catalog-metadata**

Get metadata for the given file specified by its Logical File Name or for a list of files contained in the specified file

Usage:

```bash
dirac-dms-catalog-metadata <lfn | fileContainingLfns> [Catalog]
```

Example:

```bash
$ dirac-dms-catalog-metadata /formation/user/v/vhamar/Example.txt
FileName Size GUID Status Checksum
/formation/user/v/vhamar/Example.txt 34 EDE6DDA4-3344-3F39-A993-8349BA41EB23 1 eed20d47
```

**dirac-dms-change-replica-status**

Change status of replica of a given file or a list of files at a given Storage Element

Usage:

```bash
dirac-dms-change-replica-status <lfn | fileContainingLfns> <SE> <status>
```

**dirac-dms-check-file-integrity**

**dirac-dms-clean-directory**

Clean the given directory or a list of directories by removing it and all the contained files and subdirectories from the physical storage and from the file catalogs.

Usage:

```bash
dirac-dms-clean-directory <lfn | fileContainingLfns> <SE> <status>
```

Example:

```bash
$ dirac-dms-clean-directory /formation/user/v/vhamar/newDir
Cleaning directory /formation/user/v/vhamar/newDir ... OK
```
**dirac-dms-create-replication-request**

Create a DIRAC transfer/replicateAndRegister request to be executed by the DMS Transfer Agent

*Usage:*

```
dirac-dms-create-replication-request [option|cfgfile] ... DestSE LFN ...
```

*Arguments:*

- **DestSE**: Destination StorageElement
- **LFN**: LFN or file containing a List of LFNs

*Options:*

- **-m --Monitor** : Monitor the execution of the Request (default: print request ID and exit)

**dirac-dms-data-size**

Get the size of the given file or a list of files

*Usage:*

```
dirac-dms-data-size <lfn | fileContainingLfns> <SE> <status>
```

*Options:*

- **-u: --Unit=** : Unit to use [default GB] (MB,GB,TB,PB)

*Example:*

```
$ dirac-dms-data-size /formation/user/v/vhamar/Example.txt
-----------------------------
Files | Size (GB)
-----------------------------
   1 | 0.0
-----------------------------
```

**dirac-dms-filecatalog-cli**

Launch the File Catalog shell

*Usage:*

```
dirac-dms-filecatalog-cli [option]
```

*Options:*

- **-f: --file-catalog=** : Catalog client type to use (default FileCatalog)

*Example:*

```
$ dirac-dms-filecatalog-cli
Starting DIRAC FileCatalog client
File Catalog Client $Revision: 1.17 $Date: 
FC://help
Documented commands (type help <topic>):
```
===
add chmod find guid ls pwd replicate rmreplica user
chown get id meta register rm size
chgrp exit group lcd mkdir replicas rmdir unregister

Undocumented commands:
help

FC:/>

dirac-dms-fts-monitor

Monitor the status of the given FTS request

Usage:

dirac-dms-fts-monitor <lfn|fileOfLFN> sourceSE targetSE server GUID


dirac-dms-fts-submit

Submit an FTS request, monitor the execution until it completes

Usage:

dirac-dms-fts-submit [option|cfgfile] ... LFN sourceSE targetSE

Arguments:

LFN: Logical File Name or file containing LFNs
sourceSE: Valid DIRAC SE
targetSE: Valid DIRAC SE


dirac-dms-get-file

Retrieve a single file or list of files from Grid storage to the current directory.

Usage:

dirac-dms-get-file [option|cfgfile] ... LFN ...

Arguments:

LFN: Logical File Name or file containing LFNs

Example:

$ dirac-dms-get-file /formation/user/v/vhamar/Example.txt
{'Failed': {},
 'Successful': {'/formation/user/v/vhamar/Example.txt': '/afs/in2p3.fr/home/h/hamar/Tests/DMS/Example.txt'}}
**dirac-dms-lfn-accessURL**

Retrieve an access URL for an LFN replica given a valid DIRAC SE.

Usage:

```
dirac-dms-lfn-accessURL [option|cfgfile] ... LFN SE
```

Arguments:

- **LFN**: Logical File Name or file containing LFNs
- **SE**: Valid DIRAC SE

Example:

```
$ dirac-dms-lfn-accessURL /formation/user/v/vhamar/Example.txt DIRAC-USER
{'Failed': {},
 'Successful': {'/formation/user/v/vhamar/Example.txt': 'dips://dirac.in2p3.fr:9148/DataManagement/StorageElement /formation/user/v/vhamar/Example.txt'}}
```

**dirac-dms-lfn-logging-info**

Retrieve logging information for a given LFN

Usage:

```
dirac-dms-lfn-logging-info [option|cfgfile] ... LFN ...
```

Arguments:

- **LFN**: Logical File Name or file containing LFNs

**dirac-dms-lfn-metadata**

Obtain replica metadata from file catalogue client.

Usage:

```
dirac-dms-lfn-metadata [option|cfgfile] ... LFN ...
```

Arguments:

- **LFN**: Logical File Name or file containing LFNs

Example:

```
$ dirac-dms-lfn-metadata /formation/user/v/vhamar/Example.txt
{'Failed': {},
 'Successful': {'/formation/user/v/vhamar/Example.txt': {'Checksum': 'eed20d47',
 'ChecksumType': 'Adler32',
 'CreationDate': datetime.datetime(2011, 2, 11, 14, 52, 47),
 'FileID': 250L,
 'GID': 2,
 'GUID': 'EDE6DDA4-3344-3F39-A993-8349BA41EB23',
 'Mode': 509,
 'ModificationDate': datetime.datetime(2011, 2, 11, 14, 52, 47),
 'Owner': 'vhamar',
 'OwnerGroup': 'dirac_user',
 ...}}}
```
dirac-dms-lfn-replicas

Obtain replica information from file catalogue client.

Usage:

dirac-dms-lfn-replicas [option|cfgfile] ... LFN ...

Arguments:

LFN: Logical File Name or file containing LFNs

Options:

-a --All: Also show inactive replicas

Example:

$ dirac-dms-lfn-replicas /formation/user/v/vhamar/Test.txt

{'Failed': {},
'Successful': {'/formation/user/v/vhamar/Test.txt': {'M3PEC-disk': 'srm://se0.m3pec.u-bordeaux1.fr/dpm/m3pec.u-bordeaux1.fr/home/formation/user/v/vhamar/Test.txt'}}}

dirac-dms-pfn-accessURL

Retrieve an access URL for a PFN given a valid DIRAC SE

Usage:

dirac-dms-pfn-accessURL [option|cfgfile] ... PFN SE

Arguments:

PFN: Physical File Name or file containing PFNs
SE: Valid DIRAC SE

dirac-dms-pfn-metadata

Retrieve metadata for a PFN given a valid DIRAC SE

Usage:

dirac-dms-pfn-metadata [option|cfgfile] ... PFN SE

Arguments:

PFN: Physical File Name or file containing PFNs
SE: Valid DIRAC SE
dirac-dms-remove-catalog-files

Remove the given file or a list of files from the File Catalog

Usage:

```
dirac-dms-remove-catalog-files <LFN | fileContainingLFNs>
```

Example:

```
$ dirac-dms-remove-catalog-files /formation/user/v/vhamar/1/1134/StdOut
Successfully removed 1 catalog files.
```

dirac-dms-remove-catalog-replicas

Remove the given file replica or a list of file replicas from the File Catalog

Usage:

```
dirac-dms-remove-catalog-replicas <LFN | fileContainingLFNs>
```

dirac-dms-remove-files

Remove the given file or a list of files from the File Catalog and from the storage

Usage:

```
dirac-dms-remove-files <LFN | fileContainingLFNs>
```

Example:

```
$ dirac-dms-remove-files /formation/user/v/vhamar/Test.txt
```

dirac-dms-remove-lfn-replica

Remove replica of LFN from specified Storage Element and File catalogs.

Usage:

```
dirac-dms-remove-lfn-replica [option|cfgfile] ... LFN SE
```

Arguments:

- **LFN**: Logical File Name or file containing LFNs
- **SE**: Valid DIRAC SE

**Note**: Use `dirac-dms-remove-lfn-replica` to remove a replica from a specific Storage Element (SE) and the File Catalogs.

**Note**: Use `dirac-dms-remove-lfn` to remove an LFN and all associated replicas from Storage Elements and File Catalogs.

**Note**: Use `dirac-dms-remove-files` to remove a file or a list of files from the File Catalog and from the storage.

**Note**: Use `dirac-dms-remove-catalog-files` to remove a file or a list of files from the File Catalog.

**Note**: Use `dirac-dms-remove-catalog-replicas` to remove a file replica or a list of file replicas from the File Catalog.
LFN: Logical File Name or file containing LFNs

Example:

$ dirac-dms-remove-lfn /formation/user/v/vhamar/0/16/StdOut

{'Failed': {},
'Successful': {'/formation/user/v/vhamar/0/16/StdOut': {'FileCatalog': True}}}

dirac-dms-remove-replicas

Remove the given file replica or a list of file replicas from the File Catalog and from the storage.

Usage:

dirac-dms-remove-replicas <LFN | fileContainingLFNs> SE [SE]

Example:

$ dirac-dms-remove-replicas /formation/user/v/vhamar/Test.txt IBCP-disk
Successfully removed DIRAC-USER replica of /formation/user/v/vhamar/Test.txt

dirac-dms-replica-metadata

Get the given file replica metadata from the File Catalog

Usage:

dirac-dms-replica-metadata <LFN | fileContainingLFNs> SE

dirac-dms-replicate-lfn

Replicate an existing LFN to another Storage Element

Usage:

dirac-dms-replicate-lfn [option|cfgfile] ... LFN Dest [Source [Cache]]

Arguments:

LFN: Logical File Name or file containing LFNs

Dest: Valid DIRAC SE

Source: Valid DIRAC SE

Cache: Local directory to be used as cache

Example:

$ dirac-dms-replicate-lfn /formation/user/v/vhamar/Test.txt DIRAC-USER

{'Failed': {},
'Successful': {'/formation/user/v/vhamar/Test.txt': {'register': 0.50833415985107422, 'replicate': 11.878520965576172}}

1.4. Commands Reference 59
**dirac-dms-set-replica-status**

Set the status of the replicas of given files at the provided SE

**Usage:**

```
dirac-dms-set-replica-status [option|cfgfile] ... <LFN|File> SE Status
```

**Arguments:**

- **LFN:** LFN
- **File:** File name containing a list of affected LFNs
- **SE:** Name of Storage Element
- **Status:** New Status for the replica

**dirac-dms-show-fts-status**

Usage:

```
dirac-dms-show-fts-status.py (<options>|<cfgFile>)*
```

**Options:**

```
-p --problematic : show only problematic channels
```

**dirac-dms-show-se-status**

Get status of the available Storage Elements

**Usage:**

```
dirac-dms-show-se-status [<options>]
```

**Example:**

```
$ dirac-dms-show-se-status
Storage Element Read Status Write Status
DIRAC-USER Active Active
IN2P3-disk Active Active
IPSL-IPGP-disk Active Active
IRES-disk InActive InActive
M3PEC-disk Active Active
ProductionSandboxSE Active Active
```

**dirac-dms-user-lfns**

Get the list of all the user files.

**Usage:**

```
dirac-dms-user-lfns [option|cfgfile] ...
```

**Options:**
-D: --Days= : Match files older than number of days [0]
-M: --Months= : Match files older than number of months [0]
-Y: --Years= : Match files older than number of years [0]
-w: --Wildcard= : Wildcard for matching filenames [*]
-b: --BaseDir= : Base directory to begin search (default /[vo]/user/[initial]/[username])
-e --EmptyDirs : Create a list of empty directories

Example:

$ dirac-dms-user-lfns
/formation/user/v/vhamar: 14 files, 6 sub-directories
/formation/user/v/vhamar/newDir2: 0 files, 0 sub-directories
/formation/user/v/vhamar/testDir: 0 files, 0 sub-directories
/formation/user/v/vhamar/0: 0 files, 6 sub-directories
/formation/user/v/vhamar/test: 0 files, 0 sub-directories
/formation/user/v/vhamar/meta-test: 0 files, 0 sub-directories
/formation/user/v/vhamar/1: 0 files, 4 sub-directories
/formation/user/v/vhamar/0/994: 1 files, 0 sub-directories
/formation/user/v/vhamar/0/20: 1 files, 0 sub-directories
/formation/user/v/vhamar/0/998: 1 files, 0 sub-directories
/formation/user/v/vhamar/0/45: 1 files, 0 sub-directories
/formation/user/v/vhamar/0/16: 0 files, 0 sub-directories
/formation/user/v/vhamar/0/11: 1 files, 0 sub-directories
/formation/user/v/vhamar/1/1004: 1 files, 0 sub-directories
/formation/user/v/vhamar/1/1026: 1 files, 0 sub-directories
/formation/user/v/vhamar/1/1133: 1 files, 0 sub-directories
/formation/user/v/vhamar/1/1134: 0 files, 0 sub-directories
22 matched files have been put in formation-user-v-vhamar.lfns

**dirac-dms-user-quota**

Get the currently defined user data volume quotas

Usage:

dirac-dms-user-quota [options]

Example:

$ dirac-dms-user-quota
Current quota found to be 0.0 GB

**1.4.2 Workload Management Command Reference**

In this subsection all the Dirac workload management commands available are explained.

**dirac-wms-cpu-normalization**

Determine Normalization for current CPU. Used by jobs.

Usage:
### dirac-wms-cpu-normalization

Options:

- `-U` --Update : Update dirac.cfg with the resulting value

#### dirac-wms-get-normalized-queue-length

Report Normalized CPU length of queue

**Usage:**

```bash
dirac-wms-get-normalized-queue-length [option|cfgfile] ... Queue ...
```

**Arguments:**

- **Queue**: GlueCEUniqueID of the Queue (ie, juk.nikhef.nl:8443/cream-pbs-lhcb)

**Example:**

```bash
$ dirac-wms-get-normalized-queue-length cclcgceli03.in2p3.fr:2119/jobmanager-bqs-long cclcgceli03.in2p3.fr:2119/jobmanager-bqs-long 857400.0
```

### dirac-wms-get-queue-normalization

Report Normalization Factor applied by Site to the given Queue

**Usage:**

```bash
dirac-wms-get-queue-normalization [option|cfgfile] ... Queue ...
```

**Arguments:**

- **Queue**: GlueCEUniqueID of the Queue (ie, juk.nikhef.nl:8443/cream-pbs-lhcb)

**Example:**

```bash
$ dirac-wms-get-queue-normalization cclcgceli03.in2p3.fr:2119/jobmanager-bqs-long cclcgceli03.in2p3.fr:2119/jobmanager-bqs-long 2500.0
```

### dirac-wms-job-attributes

Retrieve attributes associated with the given DIRAC job

**Usage:**

```bash
dirac-wms-job-attributes [option|cfgfile] ... JobID ...
```

**Arguments:**

- **JobID**: DIRAC Job ID

**Example:**
DIRAC Documentation, Release integration

```plaintext
$ dirac-wms-job-attributes 1
{'AccountedFlag': 'False',
 'ApplicationNumStatus': '0',
 'ApplicationStatus': 'Unknown',
 'CPUTime': '0.0',
 'DIRACSetup': 'EELA-Production',
 'DeletedFlag': 'False',
 'EndExecTime': '2011-02-14 11:28:01',
 'FailedFlag': 'False',
 'HeartBeatTime': '2011-02-14 11:28:01',
 'ISandboxReadyFlag': 'False',
 'JobGroup': 'NoGroup',
 'JobID': '1',
 'JobName': 'DIRAC_vhamar_602138',
 'JobSplitType': 'Single',
 'JobType': 'normal',
 'KilledFlag': 'False',
 'LastUpdateTime': '2011-02-14 11:28:11',
 'MasterJobID': '0',
 'MinorStatus': 'Execution Complete',
 'OSandboxReadyFlag': 'False',
 'Owner': 'vhamar',
 'OwnerDN': '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
 'OwnerGroup': 'eela_user',
 'RescheduleCounter': '0',
 'RescheduleTime': 'None',
 'RetrievedFlag': 'False',
 'RunNumber': '0',
 'Site': 'EELA.UTFSM.cl',
 'StartExecTime': '2011-02-14 11:27:48',
 'Status': 'Done',
 'SubmissionTime': '2011-02-14 10:12:40',
 'SystemPriority': '0',
 'UserPriority': '1',
 'VerifiedFlag': 'True'}
```

**dirac-wms-job-delete**

Delete DIRAC job from WMS, if running it will be killed

Usage:

```plaintext
dirac-wms-job-delete [option|cfgfile] ... JobID ...
```

Arguments:

- **JobID**: DIRAC Job ID

Example:

```plaintext
$ dirac-wms-job-delete 12
Deleted job 12
```

**dirac-wms-job-get-input**

Retrieve input sandbox for DIRAC Job
Usage:

dirac-wms-job-get-input [option|cfgfile] ... JobID ...

Arguments:

JobID: DIRAC Job ID

Options:

-D: --Dir= : Store the output in this directory

Example:

$ dirac-wms-job-get-input 13
Job input sandbox retrieved in InputSandbox13/

dirac-wms-job-get-jdl

Retrieve the current JDL of a DIRAC job

Usage:

dirac-wms-job-get-jdl [option|cfgfile] ... JobID ...

Arguments:

JobID: DIRAC Job ID

Example:

$ dirac-wms-job-get-jdl 1
{'Arguments': '-ltrA',
'CPUTime': '86400',
'DIRACSetup': 'EELA-Production',
'Executable': '/bin/ls',
'JobID': '1',
'JobName': 'DIRAC_vhamar_602138',
'JobRequirements': '{
  OwnerDN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar;
  OutputSandbox': ['std.out', 'std.err'],
  'Owner': 'vhamar',
  'OwnerDN': '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
  'OwnerGroup': 'eela_user',
  'OwnerName': 'vhamar',
  'Priority': '1'}

dirac-wms-job-get-output-data

Retrieve the output data files of a DIRAC job

Usage:

dirac-wms-job-get-output-data [option|cfgfile] ... JobID ...

Arguments:

JobID: DIRAC Job ID

Options:


**dirac-wms-job-get-output**

Retrieve output sandbox for a DIRAC job

Usage:

`dirac-wms-job-get-output [option|cfgfile] ... JobID ...`

Arguments:

JobID: DIRAC Job ID or a name of the file with JobID per line

Options:

-D: --Dir= : Store the output in this directory

Example:

```
$ dirac-wms-job-get-output 1
Job output sandbox retrieved in 1/
```

**dirac-wms-job-kill**

Issue a kill signal to a running DIRAC job

Usage:

`dirac-wms-job-kill [option|cfgfile] ... JobID ...`

Arguments:

JobID: DIRAC Job ID

Example:

```
$ dirac-wms-job-kill 1918
Killed job 1918
```

Consider that:

- jobs will not disappear from JobDB until JobCleaningAgent has deleted them
- jobs will be deleted “immediately” if they are in the status ‘Deleted’
- USER jobs will be deleted after a grace period if they are in status Killed, Failed, Done

What happens when you hit the “kill job” button: - if the job is in status ‘Running’, ‘Matched’, ‘Stalled’ it will be properly killed, and then it’s status will be marked as ‘Killed’ - otherwise, it will be marked directly as ‘Killed’.

**dirac-wms-job-logging-info**

Retrieve history of transitions for a DIRAC job

Usage:
**DIRAC Documentation, Release integration**

**dirac-wms-job-logging-info** [option|cfgfile] ... JobID ...  

**Arguments:**  
JobID: DIRAC Job ID  

**Example:**  

```
$ dirac-wms-job-logging-info 1
```

<table>
<thead>
<tr>
<th>Status</th>
<th>MinorStatus</th>
<th>ApplicationStatus</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received</td>
<td>Job accepted</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Received</td>
<td>False</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Checking</td>
<td>JobSanity</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Checking</td>
<td>JobScheduling</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Waiting</td>
<td>Pilot Agent Submission</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Matched</td>
<td>Assigned</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Matched</td>
<td>Job Received by Agent</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Matched</td>
<td>Submitted To CE</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Running</td>
<td>Job Initialization</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Running</td>
<td>Application</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Completed</td>
<td>Application Finished Successfully</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Completed</td>
<td>Uploading Output Sandbox</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Completed</td>
<td>Output Sandbox Uploaded</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
<tr>
<td>Done</td>
<td>Execution Complete</td>
<td>Unknown</td>
<td>2011-11-02</td>
</tr>
</tbody>
</table>

**dirac-wms-job-parameters**  

Retrieve parameters associated to the given DIRAC job  

**Usage:**  

```
$ dirac-wms-job-parameters [option|cfgfile] ... JobID ... 
```

**Arguments:**  
JobID: DIRAC Job ID  

**Example:**  

```
$ dirac-wms-job-parameters 1
{'CPU(MHz)': '1596.479',
 'CPUNormalizationFactor': '6.8',
 'CPUScalingFactor': '6.8',
 'CacheSize(kB)': '4096KB',
 'GridCEQueue': 'ce.labmc.inf.utfsm.cl:2119/jobmanager-1cgps-prod',
 'HostName': 'wn05.labmc',
 'JobPath': 'JobPath,JobSanity,JobScheduling,TaskQueue',
 'JobSanityCheck': 'Job: 1 JDL: OK, InputData: No input LFNs, Input Sandboxes: 0, OK.',
 'JobWrapperPID': '599',
 'LocalAccount': 'prod006',
 'LocalBatchID': '',
 'LocalJobID': '277821.ce.labmc.inf.utfsm.cl',
 'MatcherServiceTime': '2.27646398544',
 'Memory(kB)': '858540kB',
 'ModelName': 'Intel(R)Xeon(R)CPU 1110@1.60GHz',
 'NormCPUTime(s)': '1.02',
 'OK': 'True',
 'OutputSandboxMissingFiles': 'std.err',
}
dirac-wms-job-peek

Peek StdOut of the given DIRAC job

Usage:
```
dirac-wms-job-peek [option|cfgfile] ... JobID ...
```

Arguments:
- **JobID**: DIRAC Job ID

Example:
```
$ dirac-wms-job-peek 1
```

dirac-wms-job-reschedule

Reschedule the given DIRAC job

Usage:
```
dirac-wms-job-reschedule [option|cfgfile] ... JobID ...
```

Arguments:
- **JobID**: DIRAC Job ID

Example:
```
$ dirac-wms-job-reschedule 1
Rescheduled job 1
```

dirac-wms-job-status

Retrieve status of the given DIRAC job

Usage:
```
dirac-wms-job-status [option|cfgfile] ... JobID ...
```

Arguments:
- **JobID**: DIRAC Job ID

Options:
- **-f**: --file= : Get status for jobs with IDs from the file
- **-g**: --group= : Get status for jobs in the given group
Example:

```
$ dirac-wms-job-status 2
JobID=2 Status=Done; MinorStatus=Execution Complete; Site=EELA.UTFSM.cl;
```

**dirac-wms-job-submit**

Submit jobs to DIRAC WMS

Usage:

```
dirac-wms-job-submit [option|cfgfile] ... JDL ...
```

Arguments:

- **JDL:** Path to JDL file

Example:

```
$ dirac-wms-job-submit Simple.jdl
JobID = 11
```

**dirac-wms-jobs-select-output-search**

Retrieve output sandbox for DIRAC Jobs for the given selection and search for a string in their std.out

Usage:

```
dirac-wms-jobs-select-output-search [option|cfgfile] ... String ...
```

Arguments:

- **String:** string to search for

Options:
- **--Status=** : Primary status
- **--MinorStatus=** : Secondary status
- **--ApplicationStatus=** : Application status
- **--Site=** : Execution site
- **--Owner=** : Owner (DIRAC nickname)
- **--JobGroup=** : Select jobs for specified job group
- **--Date=** : Date in YYYY-MM-DD format, if not specified default is today
- **--File=** : File name, if not specified default is std.out

**dirac-wms-select-jobs**

Select DIRAC jobs matching the given conditions

Usage:
dirac-wms-select-jobs [option|cfgfile] ... JobID ...

Options:

- --Status= : Primary status
- --MinorStatus= : Secondary status
- --ApplicationStatus= : Application status
- --Site= : Execution site
- --Owner= : Owner (DIRAC nickname)
- --JobGroup= : Select jobs for specified job group
- --Date= : Date in YYYY-MM-DD format, if not specified default is today

1.4.3 Others Command Reference

In this subsection the Data Management commands are collected.

**dirac-cert-convert.sh**

Usage:

```
dirac-cert-convert.sh CERT_FILE_NAME.p12
```

**dirac-info**

Report info about local DIRAC installation

Usage:

```
dirac-info [option|cfgfile] ... Site
```

Example:

```
$ dirac-info
  DIRAC version : v5r12
  Setup : Dirac-Production
  VirtualOrganization : vo.formation.idgrilles.fr
```

**dirac-myproxy-upload**

Usage:

```
dirac-myproxy-upload.py <options>|<cfgFile>)*
```

Options:
-f: --file= : File to use as proxy
-D --DN : Use DN as myproxy username
-i --version : Print version

**dirac-platform**

Linux_x86_64_glibc-2.5

Example:

$ dirac-platform
Linux_x86_64_glibc-2.5

**dirac-proxy-get-uploaded-info**

Usage:

dirac-proxy-get-uploaded-info.py (<options>|<cfgFile>)*

Options:

-u: --user= : User to query (by default oneself)

Example:

$ dirac-proxy-get-uploaded-info
Checking for DNs /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
--------------------------------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>UserDN</th>
<th>UserGroup</th>
<th>ExpirationTime</th>
<th>PersistentFlag</th>
</tr>
</thead>
</table>

**dirac-proxy-info**

Usage:

dirac-proxy-info.py (<options>|<cfgFile>)*

Options:

-f: --file= : File to use as user key
-i --version : Print version
-n --novoms : Disable VOMS
-v --checkvalid : Return error if the proxy is invalid
-x --nocs : Disable CS
-e --steps : Show steps info
-j --noclockcheck : Disable checking if time is ok
-m --uploadedinto : Show uploaded proxies info

Example:

$ dirac-proxy-info
subject : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy/CN=proxy
issuer : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy
identity : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar

timeleft : 23:53:55
DIRAC group : dirac_user
path : /tmp/x509up_u40885
username : vhamar
VOMS : True
VOMS fqan : ['/formation']

dirac-proxy-init

Usage:

dirac-proxy-init.py (<options>|<cfgFile>)*

Options:

-v: --valid= : Valid HH:MM for the proxy. By default is 24 hours
-g: --group= : DIRAC Group to embed in the proxy
-b: --strength= : Set the proxy strength in bytes
-l --limited : Generate a limited proxy
-t --strict : Fail on each error. Treat warnings as errors.
-S --summary : Enable summary output when generating proxy
-C: --Cert= : File to use as user certificate
-K: --Key= : File to use as user key
-u: --out= : File to write as proxy
-x --nocs : Disable CS check
-p --pwstdin : Get passwd from stdin
-i --version : Print version
-j --noclockcheck : Disable checking if time is ok
-U --upload : Upload a long lived proxy to the ProxyManager
-P --uploadPilot : Upload a long lived pilot proxy to the ProxyManager
-M --VOMS : Add voms extension

Example:
1.5 Tutorials

This page is the work in progress. See more material here soon!

1.5.1 1. Client Installation

The DIRAC client installation procedure consists of several steps. This example is destined for tutorials. For more information about various options of installing DIRAC Client see the Getting Started guide.

1.1 Install script

Download the `dirac-install` script from [here](http://lhcbproject.web.cern.ch/lhcbproject/dist/Dirac_project/dirac-install):

```bash
wget -np http://lhcbproject.web.cern.ch/lhcbproject/dist/Dirac_project/dirac-install --no-check-certificate
chmod +x dirac-install
```

1.2 Installation

In most cases you are installing the DIRAC client to work as a member of some particular user community or, in other words, Virtual Organization (VO). The managers of your VO usually prepare default settings to be applied for the DIRAC client installation. In this case the installation procedure reduces to the following assuming the name of the Virtual Organization `vo.formation.idgrilles.fr`:

```bash
./dirac-install -V formation
source bashrc
```

The above command will download also `vo.formation.idgrilles.fr_defaults.cfg` file which contains the VO default settings. Check with your VO managers if this mode of installation is available.

1.3 Configuration

Once the client software is installed, it should be configured in order to access the corresponding DIRAC services. The minimal necessary configuration is done by the following command:

```bash
dirac-configure defaults-formation.cfg
```
When you run this command for the first time you might see some errors messages about a failure to access DIRAC services. This is normal because at this point the configuration is not yet done and you do not have a valid proxy. After creating a proxy with `proxy-init` command, just repeat the `dirac-configure` command once again.

1.4 Updating the client installation

The client software update when a new version is available is simply done by running again the `dirac-install` command as in p.1.2. You can run the `dirac-install` giving the exact version of the DIRAC software, for example:

```
dirac-install -r v6r0-pre11
```

1.5.2 2. Managing user credentials

This section assumes that the DIRAC client is already installed and configured.

2.1 Managing Certificates

2.1.1 Donwloading Certificate from browser

- Get the certificate from the browser:
  - Firefox:
    Preferences -> Advanced -> View Certificates -> Select your certificate -> Backup
  - Explorer:
    Tools -> Internet Options ->Content -> Certificates -> Certificates ->Import/Export

As a result you will get the certificate as a file with .p12 extension.

2.1.2 Converting Certificates from P12 to PEM format

- Run `dirac-cert-convert` script to convert your certificate to the appropriate form:

```
dirac-cert-convert.sh <USERCERT>.p12
```

Output of this command must look like:

```
$ dirac-cert-convert.sh usercert.p12
Creating globus directory
Converting p12 key to pem format
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
Converting p12 certificate to pem format
Enter Import Password:
MAC verified OK
Information about your certificate:
subject= /O=GRID-FR/C=FR/O=CNRS/GU=CPPM/CN=Vanessa Hamar
issuer= /C=FR/O=CNRS/CN=GRID2-FR
Done
```
“Enter Import Password:” prompt requires the password given when the certificate was exported from the browser. It will be requested twice. The PEM pass phrase is the password associated with the created private key. This password will be requested each time you will create a proxy. Do not forget it!

- Check that your certificate was correctly converted and placed in the $HOME/.globus directory, in PEM format and with correct permissions:

```bash
$ ls -la ~/.globus
total 16
drwxr-xr-x 2 hamar marseill 2048 Oct 19 13:01 .
drwxr-xr-x 42 hamar marseill 4096 Oct 19 13:00 ..
-rw-r--r-- 1 hamar marseill 6052 Oct 19 13:00 usercert.p12
-rw-r--r-- 1 hamar marseill 1914 Oct 19 13:01 usercert.pem
-r-------- 1 hamar marseill 1917 Oct 19 13:01 userkey.pem
```

### 2.2 Managing Proxies

Before running any command in the grid, it is mandatory to have a valid certificate proxy. The commands to create a valid proxy using DIRAC commands are shown below.

#### 2.2.1 Creating a user proxy

- First, in the machine where the DIRAC client is installed setup the DIRAC environment running the following commands:

```bash
cd $DIRAC_PATH
source bashrc
```

- After the environment is set up, you are able to create your proxy with the following command:

```
dirac-proxy-init --group dirac_user -U
```

For example, with the additional debug option the output must be like the following:

```bash
$ dirac-proxy-init --debug --group dirac_user -u
Generating proxy...  
Enter Certificate password: 
Contacting CS...  
Checking DN /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
Username is atsareg
Creating proxy for atsareg@dirac_user (/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev)
Uploading proxy for dirac_user...
Uploading dirac_user proxy to ProxyManager...
Loading user proxy
Uploading proxy on-the-fly
Cert file /home/andrei/.globus/usercert.pem
Key file /home/andrei/.globus/userkey.pem
Loading cert and key
User credentials loaded
Uploading...
Proxy uploaded
Proxy generated:
subject : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev/CN=proxy
issuer : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
identity : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
timeleft : 23:59:57
DIRAC group : dirac_user
```
As a result of this command, several operations are accomplished:

- a long user proxy (with the length of the validity of the certificate) is uploaded to the DIRAC ProxyManager service, equivalent of the gLite MyProxy service

- a short user proxy is created with the DIRAC extension carrying the DIRAC group name and with the VOMS extension corresponding to the DIRAC group if the gLite UI environment is available.

If the gLite UI environment is not available, the VOMS extensions will not be loaded into the proxy. This is not a serious problem, still most of the operations will be possible.

2.2.2 Getting the proxy information

- Check that your proxy was correctly created and the DIRAC group and the VOMS extension are set correctly, running the command:

  \[
  \text{dirac-proxy-info}
  \]

  For example:

  $ dirac-proxy-info
  
  subject : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy/CN=proxy
  issuer : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy
  identity : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
  timeleft : 23:53:55
  DIRAC group : dirac_user
  path : /tmp/x509up_u40885
  username : vhamar
  VOMS : True
  VOMS fqan : ['/vo.formation.idgrilles.fr']

- At this moment, your proxy must be uploaded to the ProxyManager service. To check that:

  \[
  \text{dirac-proxy-get-uploaded-info}
  \]

  In this case the output shows user DN, group, expiration time and persistency flag:

  $ dirac-proxy-get-uploaded-info
  
  Checking for DNs /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
  
<table>
<thead>
<tr>
<th>UserDN</th>
<th>UserGroup</th>
<th>ExpirationTime</th>
<th>Persistency</th>
</tr>
</thead>
</table>

- The same can be checked in the Web Portal at the following location:

  Systems -> Framework -> Manage Proxy

  Using the portal you have the option to delete your proxies.
1.5.3 3. JDLs and Job Management Basic

JDL stands for Job Description Language and it is the standard way of job description in the gLite environment. DIRAC does not use the JDL objects internally but allows the job description using the JDL syntax. An important difference is that there is no Requirements attribute which is used in the gLite JDL to select specific resources. Instead, certain attributes are interpreted as job requirements, e.g. CPUTime, Site, etc.

3.1 Simple Jobs

The following is the description of the job which just lists the working directory - Simple.jdl:

```plaintext
JobName = "Simple_Job";
Executable = "/bin/ls";
Arguments = "-ltr";
StdOutput = "StdOut";
StdError = "StdErr";
OutputSandbox = {"StdOut","StdErr"};
```

To submit the job:

```
dirac-wms-job-submit Simple.jdl
```

3.2 Jobs with Input Sandbox and Output Sandbox

In most cases the job input data or executable files are available locally and should be transferred to the grid to run the job. In this case the InputSandbox attribute can be used to move the files together with the job.

- Create InputAndOuputSandbox.jdl:

  ```plaintext
  JobName = "InputAndOuputSandbox";
  Executable = "testJob.sh";
  StdOutput = "StdOut";
  StdError = "StdErr";
  InputSandbox = {"testJob.sh"};
  OutputSandbox = {"StdOut","StdErr"};
  ```

- And create a simple shell script.

  ```bash
  #!/bin/bash
  /bin/hostname
  /bin/date
  /bin/ls -la
  ```

- After creation of JDL file the next step is to submit the job, using the command:

  ```
dirac-wms-job-submit InputAndOuputSandbox.jdl
```

3.3 Jobs with Input and Output Data

In case where the data, programs, etc are stored in a Grid Storage Element, it can be specified as part of InputSandbox or InputData. InputSandbox can be declared as a list, separated by commas with each file between "".

Before the grid file can be used, it should be uploaded first to the Grid. This is done using the following command:
**dirac-dms-add-file <LFN> <local_file> SE**

For example:

```
bash-3.2$ dirac-dms-add-file /vo.formation.idgrilles.fr/user/v/vhamar/test.txt test.txt M3PEC-disk -o LogLevel=INFO
2010-10-17 17:15:04 UTC dirac-dms-add-file.py WARN: ReplicaManager.__getClientCertGroup: Proxy information does not contain the VOMs information.
2010-10-17 17:15:05 UTC dirac-dms-add-file.py INFO: ReplicaManager.putAndRegister: Checksum information not provided. Calculating adler32.
2010-10-17 17:15:05 UTC dirac-dms-add-file.py INFO: ReplicaManager.putAndRegister: Checksum calculated to be cc500ba0.
2010-10-17 17:15:06 UTC dirac-dms-add-file.py WARN: StorageElement.isValid: The 'operation' argument is not supplied. It should be supplied in the future.
2010-10-17 17:15:06 UTC dirac-dms-add-file.py INFO: SRM2Storage.__putFile: Using 1 streams
2010-10-17 17:15:06 UTC dirac-dms-add-file.py INFO: SRM2Storage.__putFile: Executing transfer of file:
2010-10-17 17:15:13 UTC dirac-dms-add-file.py ERROR: StorageElement.getPfnForProtocol: Requested protocol not available for SE. DIP for M3PEC-disk
2010-10-17 17:15:14 UTC dirac-dms-add-file.py INFO: ReplicaManger.putAndRegister: Sending accounting took 0.5 seconds
{'Failed': {},
 'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/test.txt': {'put': 7.5088520050048828, 'register': 0.40918898582458496}}}
```

- Use the same testJob.sh shell script as in the previous exercise.
- In the JDL we have to add OutputSE and OutputData:

```plaintext
JobName = "LFNInputSandbox";
Executable = "testJob.sh";
StdOutput = "StdOut";
StdError = "StdErr";
InputSandbox = {"testJob.sh","LFN:/vo.formation.idgrilles.fr/user/v/vhamar/test.txt"};
OutputSandbox = {"StdOut","StdErr"};
OutputSE = "M3PEC-disk";
OutputData = {"StdOut"};
```

- After creation of JDL file the next step is submit a job, using the command:

```
dirac-wms-job-submit <JDL>
```

The same effect can be achieved with the following JDL LFNInputData.jdl:

```plaintext
JobName = "LFNInputData";
Executable = "testJob.sh";
StdOutput = "StdOut";
StdError = "StdErr";
InputSandbox = {"testJob.sh"};
InputData = {"LFN:/vo.formation.idgrilles.fr/user/v/vhamar/test.txt"};
OutputSandbox = {"StdOut","StdErr"};
OutputSE = "M3PEC-disk";
OutputData = {"StdOut"};
```

An important difference of specifying input data as InputSandbox or InputData is that in the first case the data file is always downloaded local to the job running in the Grid. In the InputData case, the file can be either downloaded locally or accessed remotely using some remote access protocol, e.g. rfio or dcap, depending on the policies adopted by your Virtual Organization.

### 3.4 Managing Jobs

#### 3.4.1 Submitting a Job

- After creating the JDL file the next step is to submit a job using the command:
For example:

```bash
bash-3.2$ dirac-wms-job-submit Simple.jdl -o LogLevel=INFO
2010-10-17 15:34:36 UTC dirac-wms-job-submit.py/DiracAPI INFO: <=====DIRAC v5r10-pre2======>
2010-10-17 15:34:36 UTC dirac-wms-job-submit.py/DiracAPI INFO: Will submit job to WMS
JobID = 11
```

In the output of the command you get the DIRAC job ID which is a unique job identifier. You will use it later for other job operations.

### 3.4.2 Getting the job status

- The next step is to monitor the job status using the command:

```bash
bash-3.2$ dirac-wms-job-status 11
JobID=11 Status=Waiting; MinorStatus=Pilot Agent Submission; Site=ANY;
```

### 3.4.3 Retrieving the job output

- And finally, after the job achieves status **Done**, you can retrieve the job Output Sandbox:

```bash
bash-3.2$ dirac-wms-job-get-output [--dir output_directory] <Job_ID>
```

### 1.5.4 4. Data Management Basic

#### 4.1 Getting information

##### 4.1.1 SE availability

- The first thing is to know which Storage Elements are available for the VO, run the command:

```bash
bash-3.2$ dirac-dms-show-se-status
```

For example:

```bash
$ dirac-dms-show-se-status
Storage Element          Read Status  Write Status
DIRAC-USER               Active      Active
IN2P3-disk               Active      Active
IPSL-IPGP-disk           Active      Active
IRES-disk                InActive    InActive
M3PEC-disk               Active      Active
ProductionSandboxSE      Active      Active
```

SE names are defined in the DIRAC Configuration. These are the names that you will use with various data management commands.
4.2 Uploading a file to the Grid

- The next step is to upload a file to a Storage Element and register it into the DIRAC File Catalog. Execute the command:

  ```
  dirac-dms-add-file <LFN> <FILE> <SE>
  ```

  Output must look like this:

  ```
  $ dirac-dms-add-file /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt Test-Lyon.orig M3PEC-disk
  {'Failed': {}, 'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt': {'put': 8.3242118358612061, 'register': 0.51048803329467773}}}
  ```

  Note: The output of this command must be successful before continuing with other exercises.

4.3 Obtaining information about the data

4.3.1 Metadata

- After a file is registered into DIRAC File Catalog the metadata could be consulted any time with:

  ```
  dirac-dms-catalog-metadata <LFN>
  ```

  For example, the metadata for Test-Lyon.txt file is:

  ```
  $ dirac-dms-catalog-metadata /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
  FileName Size GUID Status Checksum
  /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt 15 1D6155B6-0405-BAB0-5552-7913EFD734A7 1 2ec4058b
  ```

4.3.2 File Metadata

- More detailed file metadata can be obtained with the following command:

  ```
  dirac-dms-lfn-metadata <LFN>
  ```

  For example:

  ```
  $ dirac-dms-lfn-metadata /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
  {'Failed': {}, 'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt': {'Checksum': '2ec4058b', 'ChecksumType': 'Adler32', 'CreationDate': datetime.datetime(2010, 10, 17, 20, 31, 31), 'ModificationDate': datetime.datetime(2010, 10, 17, 20, 31, 31), 'FileID': 15L, 'GID': 2, 'GUID': '1D6155B6-0405-BAB0-5552-7913EFD734A7', 'Mode': 509, 'Owner': 'vhamar', 'OwnerGroup': 'dirac_user', 'Size': 15L, 'Status': 1, 'UID': 2}}}
  ```
4.4 Downloading a file

- Retrieve the file previously uploaded to the Grid using the command:

  ```
  dirac-dms-get-file <LFN>
  ```

  Output must be like shown below:

  ```
  $ dirac-dms-get-file /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
  {'Failed': {},
   'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt': '/afs/in2p3.fr/home/h/hamar/Tests/DMS/Test-Lyon.txt'}}
  ```

4.5 Data Replication

4.5.1 Replicating a file

- The command used to create another replica of a given file:

  ```
  dirac-dms-replicate-lfn <LFN> <SE>
  ```

  For example:

  ```
  $ dirac-dms-replicate-lfn /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt DIRAC-USER
  {'Failed': {},
   'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt': {'register': 0.50833415985107422,
                                                                             'replicate': 11.878520965576172}}}
  ```

4.5.2 Replica information

- The following command allows to obtain the replica information for the given file:

  ```
  dirac-dms-lfn-replicas <LFN>
  ```

  An example output is shown below:

  ```
  $ dirac-dms-lfn-replicas /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
  {'Failed': {},
   'Successful': {'/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt': {'M3PEC-disk': 'srm://se0.m3pec.u-bordeaux1.fr/dpm/m3pec.u-bordeaux1.fr/home/vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt'}}}
  ```

4.5.3 Removing a replica

- To remove replicas use the command:

  ```
  dirac-dms-remove-replicas <LFN> <SE>
  ```

  For example:  
  ```
  $ dirac-dms-remove-replicas /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt IBCP-disk
  Successfully removed DIRAC-USER replica of /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
  ```

4.6 Removing Files

- Please remove all the files created during the T.P, using this command:

  ```
  dirac-dms-remove-files <LFN>
  ```
For example:

```
$ dirac-dms-remove-files /vo.formation.idgrilles.fr/user/v/vhamar/Test-Lyon.txt
$
```

### 4.7 Getting the list of user files

- To create a list of all the files stored by the user:

  `dirac-dms-user-lfns`

After running the command a file with all the LFNs will be created, the file name is associated with the user’s VO:

```
$ dirac-dms-user-lfns
/vo.formation.idgrilles.fr/user/v/vhamar: 0 files, 1 sub-directories
/vo.formation.idgrilles.fr/user/v/vhamar/2: 0 files, 3 sub-directories
/vo.formation.idgrilles.fr/user/v/vhamar/2/2389: 1 files, 0 sub-directories
/vo.formation.idgrilles.fr/user/v/vhamar/2/2390: 1 files, 0 sub-directories
/vo.formation.idgrilles.fr/user/v/vhamar/2/2391: 1 files, 0 sub-directories
3 matched files have been put in vo.formation.idgrilles.fr-user-v-vhamar.lfns
```

### 1.5.5 5. File Catalog Interface

#### 5.1 Starting the File Catalog Interface

- DIRAC File Catalog Command Line Interface (CLI) can be used to perform all the data management operations.
  You can start the CLI with the command:

  `dirac-dms-filecatalog-cli`

For example:

```
$ dirac-dms-filecatalog-cli
Starting DIRAC FileCatalog client
File Catalog Client $Revision: 1.17 $Date:
FC:/>help

Documented commands (type help <topic>):
========================================
add chmod find guid ls pwd replicate rmreplica user
cd chown get id meta register rm size
chgrp exit group lcd mkdir replicas rmdir unregister

Undocumented commands:
------------------------
help
FC:/>
```

#### 5.2 Basic File Catalog operations

- Changing directory:
5.3 Managing files and replicas

• Upload a local file to the grid storage and register it in the catalog:

\texttt{add <LFN> <local_file> <SE>}

For example:

\begin{verbatim}
FC:/>cd /vo.formation.idgrilles.fr/user/a/atsareg
FC:/vo.formation.idgrilles.fr/user/a/atsareg> add test.txt test.txt DIRAC-USER
File /vo.formation.idgrilles.fr/user/a/atsareg/test.txt successfully uploaded to the DIRAC-USER SE
FC:/vo.formation.idgrilles.fr/user/a/atsareg> ls -l
-rw-rw-r-x 0 atsareg dirac_user 856 2010-10-24 18:35:18 test.txt
\end{verbatim}

• Download grid file to the local directory:

\texttt{get <LFN> [<local_directory>]} 

For example:

\begin{verbatim}
FC:/vo.formation.idgrilles.fr/user/a/atsareg>get test.txt /home/atsareg/data
File /vo.formation.idgrilles.fr/user/a/atsareg/test.txt successfully downloaded
\end{verbatim}

• Replicate a file registered and stored in a storage element to another storage element:

\texttt{replicate <lfn> <SE>}

For example:

\begin{verbatim}
FC:/vo.formation.idgrilles.fr/user/a/atsareg>replicate test.txt M3PEC-disk
File /vo.formation.idgrilles.fr/user/a/atsareg/test.txt successfully replicated to the M3PEC-disk SE
\end{verbatim}

• List replicas:
replicas <LFN>

For example:

FC:/vo.formation.idgrilles.fr/user/a/atsareg>replicas test.txt
 lfn: /vo.formation.idgrilles.fr/user/a/atsareg/test.txt
M3PEC-disk srm://se0.m3pec.u-bordeaux1.fr:8446/srm/managerv2?SFN=/dpm/m3pec.u-bordeaux1.fr/home/vo.formation.idgrilles.fr/user/a/atsareg/test.txt

• Remove replicas:

rmreplica <LFN> <SE>

For example:

FC:/vo.formation.idgrilles.fr/user/a/atsareg>rmreplica test.txt M3PEC-disk
 lfn: /vo.formation.idgrilles.fr/user/a/atsareg/test.txt
Replica at M3PEC-disk moved to Trash Bin
FC:/vo.formation.idgrilles.fr/user/a/atsareg>replicas test.txt
 lfn: /vo.formation.idgrilles.fr/user/a/atsareg/test.txt

• Remove file:

rm <LFN>

For example:

FC:/vo.formation.idgrilles.fr/user/a/atsareg>rm test.txt
 lfn: /vo.formation.idgrilles.fr/user/a/atsareg/test.txt
File /vo.formation.idgrilles.fr/user/a/atsareg/test.txt removed from the catalog

• Remove directory:

rmdir <path>

For example:

FC:/vo.formation.idgrilles.fr/user/a/atsareg>rmdir newDir
 path: /vo.formation.idgrilles.fr/user/a/atsareg/newDir
Directory /vo.formation.idgrilles.fr/user/a/atsareg/newDir removed from the catalog

5.4 Getting extra information

• Getting file or directory size:

size <LFN>
size <dir_path>

For example:

FC:/vo.formation.idgrilles.fr/user/a/atsareg>size test.txt
 lfn: /vo.formation.idgrilles.fr/user/a/atsareg/test.txt
Size: 856
FC:/vo.formation.idgrilles.fr/user/a/atsareg>size ..
directory: /vo.formation.idgrilles.fr/user/a
Size: 2358927

• Your current identity:
For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>id
user=1(atsareg) group=2(dirac_user)
```

### 7. Advanced Job Management

#### 7.1 Parametric Jobs

A parametric job allows to submit a set of jobs in one submission command by specifying parameters for each job.

To define this parameter the attribute Parameters must be defined in the JDL, the values that it can take are:

- A list (strings or numbers).
- Or, an integer, in this case the attributes ParameterStart and ParameterStep must be defined as integers to create the list of job parameters.

#### 7.1.1 Parametric Job - JDL

A simple example is to define the list of parameters using a list of values, this list can contain integers or strings:

```plaintext
Executable = "testJob.sh";
JobName = "%n_parametric";
Arguments = "\$s";
Parameters = {"first","second","third","fourth","fifth"};
StdOutput = "StdOut_\$s";
StdError = "StdErr_\$s";
InputSandbox = {"testJob.sh"};
OutputSandbox = {"StdOut_\$s","StdErr_\$s"};
```

In this example, 5 jobs will be created corresponding to the Parameters list values. Note that other JDL attributes can contain "%s" placeholder. For each generated job this placeholder will be replaced by one of the values in the Parameters list.

In the next example, the JDL attribute values are used to create a list of 20 integers starting from 1 (ParameterStart) with a step 2 (ParameterStep):

```plaintext
Executable = "testParametricJob.sh";
JobName = "Parametric_$n";
Arguments = "$s";
Parameters = 20;
ParameterStart = 1;
ParameterStep = 2;
StdOutput = "StdOut_$n"
StdError = "StdErr_$n";
InputSandbox = {"testParametericJob.sh"};
OutputSandbox = {"StdOut_$n","StdErr_$n"};
```

Therefore, with this JDL job description will be submitted in at once. As in the previous example, the "%s" placeholder will be replaced by one of the parameter values.

Parametric jobs are submitted as normal jobs, the command output will be a list of the generated job IDs, for example:
DIRAC Documentation, Release integration

$ dirac-wms-job-submit Param.jdl
JobID = [1047, 1048, 1049, 1050, 1051]

These are standard DIRAC jobs. The jobs outputs can be retrieved as usual specifying the job IDs::

$ dirac-wms-job-get-output 1047 1048 1049 1050 1051

7.2 MPI Jobs

Message Passing Interface (MPI) is commonly used to handle the communications between tasks in parallel applications. Two versions and implementations supported in DIRAC are the following::

- MPICH-1 : MPICH1
- MPICH-2 : MPICH2

Users should know that, currently, the MPI jobs can only run on one grid site. So, the maximum number of processors that a user can require for a job depends on the capacity and the policy of the sites.

Another important point, is that some applications need all nodes to work with a shared directory, in some cases, sites provide such a shared disk space but not always.

7.2.1 MPI Jobs - JDL

To define MPI jobs using DIRAC it is necessary:

- Create a wrapper script, this script prepares the environment variables, the arguments are the mpi program without extension c, for example::

```bash
#!/bin/bash
EXECUTABLE=$1
NUMPROC=$2
DOMAIN=`hostname -f|cut -d. -f2-10`
MPICC=`which mpicc`
MPIRUN=`which mpirun`
MPIH=`which mpi.h`
# Optional
echo "========================================="
echo "DATE: " /bin/date'
echo "Domain: " $DOMAIN
echo "Executable: " $EXECUTABLE
echo "Num Proc: " $NUMPROC
echo "MPICC: " $MPICC
echo "MPIRUN: " $MPIRUN
echo "MPIH: " $MPIH
echo "MPI_SHARED_HOME: " `echo $MPI_SHARED_HOME`
echo "========================================="
echo x=`echo $MPI_SHARED_HOME`
echo "Starting MPI script"
mpdtrace
if [ $? -eq 0 ]; then
  mpicc -o $EXECUTABLE.o ./EXECUTABLE.c -lm
  if [[ -z "$x" || "$x" == "no" ]]; then
    DIR=$HOME/TEMP_DIR
  else
    export PATH=$PATH:$DIR
  fi
  for i in `mpdtrace`;
  do
  done
```

1.5. Tutorials 85
Edit the JDL: - Set the JobType attribute to "MPI" - Set Flavor attribute to specify which version of MPI libraries you want to use - MPICH2 or MPICH1 - Set CPUNumber attribute

For example:

```plaintext
JobType = "MPI";
CPUNumber = 2;
Executable = "application.sh";
Arguments = "mpifile 2 ";
StdOutput = "StdOut";
StdError = "StdErr";
InputSandbox = {"application.sh","mpifile.c","inputfile.txt"};
OutputSandbox = {"mpifile.o","StdErr","StdOut"};
Flavor = "MPICH2"
```

MPI Jobs are submitted as normal jobs, for example:

```plaintext
$ dirac-wms-job-submit mpi.jdl
JobID = 1099
```

To retrieve the job outputs use a usual dirac-wms-job-get-output command:

```plaintext
$ dirac-wms-job-get-output 1099
```

### 7.3 DIRAC API

The DIRAC API is encapsulated in several Python classes designed to be used easily by users to access a large fraction of the DIRAC functionality. Using the API classes it is easy to write small scripts or applications to manage user jobs and data.

#### 7.3.1 Submitting jobs using APIs

- First step, create a Python script specifying job requirements.
Test-API.py:

```python
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job

ej = Job()
j.setCPUPtime(500)
j.setExecutable('echo', arguments='hello')
j.setExecutable('ls', arguments='-l')
j.setExecutable('echo', arguments='hello again')
j.setName('API')
dirac = Dirac()
result = dirac.submit(j)
print 'Submission Result: ',result
```

- Send the Job using the script:

```bash
python Test-API.py
```

```bash
$ python testAPI.py
{'OK': True, 'Value': 196}
```

### 7.3.2 Retrieving Job Status

- Create a script Status-API.py:

```python
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job
import sys
dirac = Dirac()
jobid = sys.argv[1]
print dirac.status(jobid)
```

- Execute script:

```bash
python Status-API.py <Job_ID>
```

```bash
$ python Status-API.py 196
{'OK': True, 'Value': {196: {'Status': 'Done', 'MinorStatus': 'Execution Complete', 'Site': 'LCG.IRES.fr'}}}
```

### 7.3.3 Retrieving Job Output

- Example Output-API.py:

```python
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job
import sys
dirac = Dirac()
jobid = sys.argv[1]
print dirac.getOutputSandbox(jobid)
print dirac.getJobOutputData(jobid)
```

- Execute script:
7.3.4 Local submission mode

The Local submission mode is a very useful tool to check the sanity of your job before submission to the Grid. The job executable is run locally in exactly the same way (same input, same output) as it will do on the Grid Worker Node. This allows to debug the job in a friendly local environment.

Let’s perform this exercise in the python shell.

- Load python shell:

```
bash-3.2$ python
Python 2.5.5 (r255:77872, Mar 25 2010, 14:17:52)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-46)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> from DIRAC.Interfaces.API.Dirac import Dirac
>>> from DIRAC.Interfaces.API.Job import Job
>>> j = Job()
>>> j.setExecutable('echo', arguments='hello')
{'OK': True, 'Value': ''}
>>> Dirac().submit(j, mode='local')
2010-10-22 14:41:51 UTC /DiracAPI INFO: <=====DIRAC v5r10-pre2=====>
2010-10-22 14:41:51 UTC /DiracAPI INFO: Executing workflow locally without WMS submission
2010-10-22 14:41:51 UTC /DiracAPI INFO: Executing at /afs/in2p3.fr/home/h/hamar/Tests/APIs/Local
2010-10-22 14:41:51 UTC /DiracAPI INFO: Preparing environment for site DIRAC.Client.fr to execute job
2010-10-22 14:41:51 UTC /DiracAPI INFO: Attempting to submit job to local site: DIRAC.Client.fr
```

```
Executing StepInstance RunScriptStep1 of type ScriptStep1 ['ScriptStep1']
StepInstance creating module instance ScriptStep1 of type Script
2010-10-22 14:41:53 UTC dirac-jobexec.py/Script INFO: Script Module Instance Name: CodeSegment
2010-10-22 14:41:53 UTC dirac-jobexec.py/Script INFO: Command is: /bin/echo hello
2010-10-22 14:41:53 UTC dirac-jobexec.py/Script INFO: /bin/echo hello execution completed with status 0
```

```
2010-10-22 14:41:53 UTC /DiracAPI INFO: Standard output written to std.out
{'OK': True, 'Value': 'Execution completed successfully'}
```

- Exit python shell

- List the directory where you run the python shell, the outputs must be automatically created:

```
bash-3.2$ ls
Local_zbDHRe_JobDir Script1_CodeOutput.log std.err std.out
bash-3.2$ more Script1_CodeOutput.log
<<<<<<<<<<< echo hello Standard Output >>>>>>>>>>
hello
```

7.3.5 Sending Multiple Jobs

- Create a Test-API-Multiple.py script, for example:

```
from DIRAC.Interfaces.API.Dirac import Dirac
from DIRAC.Interfaces.API.Job import Job
```
j = Job()
j.setCPULimit(500)
j.setExecutable('echo', arguments='hello')
for i in range(5):
j.setName('API_%d' % i)
dirac = Dirac()
jobID = dirac.submit(j)
print 'Submission Result: ', jobID

7.3.6 Using APIs to create JDL files.

- Create a Test-API-JDL.py:

```python
from DIRAC.Interfaces.API.Job import Job
j = Job()
j.setName('APItoJDL')
j.setOutputSandbox(['*.log', 'summary.data'])
j.setInputData(['/vo.formation.idgrilles.fr/user/v/vhamar/test.txt', '/vo.formation.idgrilles.fr/user/v/vhamar/test2.txt'])
j.setOutputData(['/vo.formation.idgrilles.fr/user/v/vhamar/output1.data', '/vo.formation.idgrilles.fr/user/v/vhamar/output2.data'], OutputPath='MyFirstAnalysis')
j.setSystemConfig("")
j.setCPULimit(21600)
j.setDestination('LCG.IN2P3.fr')
j.setBannedSites(['LCG.ABCD.fr', 'LCG.EFGH.fr'])
j.setLogLevel('DEBUG')
j.setExecutionEnv({'MYVARIABLE': 'TEST'})
j.setExecutable('echo', arguments='$MYVARIABLE')
print j._toJDL()
```

- Run the API:

```bash
$ python Test-API-JDL.py
```

```
Origin = "DIRAC";
Priority = "1";
Executable = "$DIRACROOT/scripts/dirac-jobexec";
ExecutionEnvironment = "MYVARIABLE=TEST";
StdError = "std.err";
LogLevel = "DEBUG";
BannedSites =
{
  "LCG.ABCD.fr",
  "LCG.EFGH.fr"
};
StdOutput = "std.out";
Site = "LCG.IN2P3.fr";
SystemConfig = "";
OutputPath = "MyFirstAnalysis";
InputSandbox = "jobDescription.xml";
Arguments = "jobDescription.xml -o LogLevel=DEBUG";
JobGroup = "vo.formation.idgrilles.fr";
```
As you can see the parameters added to the job object are represented in the JDL job description. It can now be used together with the `dirac-wms-job-submit` command line tool.

### 1.5.7 8. Data Management Advanced

This section shows how the File Catalog can be used as a Metadata Catalog. The exercises are performed using the File Catalog CLI interface. You can start the CLI with the command:

```
dirac-dms-filecatalog-cli
```

#### 8.1 User metadata

Metadata is the information describing the user data in order to easily select the data sets of interest for user applications. In the DIRAC File Catalog metadata can be associated with any directory. It is important that subdirectories are inheriting the metadata of their parents, this allows to reduce the number of the stored metadata values. Some metadata variables can be declared as indices. Only indexed metadata can be used in data selections.

- Adding metadata to a directory:

  ```
  meta set <directory> <metaname> <metavalue>
  ```

  For example:

  ```
  FC:/vo.formation.idgrilles.fr/user/a/atsareg>meta set . ATMetaStr Test
  FC:/vo.formation.idgrilles.fr/user/a/atsareg>mkdir testDir
  Successfully created directory: /vo.formation.idgrilles.fr/user/a/atsareg/testDir
  FC:/vo.formation.idgrilles.fr/user/a/atsareg>meta set testDir AnotherMeta AnotherTest
  ```

- Getting directory metadata:

  ```
  meta get <directory>
  ```

  For example:

  ```
  FC:/vo.formation.idgrilles.fr/user/a/atsareg>meta get testDir
  AnotherMeta : AnotherTest
  ATMetaStr : Test
  ```

- Creating metadata index:
**meta index <metaname> <metatype>**

For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>meta index NewMetaInt int
```

Added metadata field NewMetaInt of type int

Possible metadata types: int,float,string,date

- Showing existing metadata indices:

```
meta show
```

For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>meta show
  ATMetaStr : VARCHAR(128)
  ATMetaInt : INT
  ATMetaDate : DATETIME
  ATMetaSet : MetaSet
  ATMetaInt1 : INT
  NewMetaInt : INT
  ATMetaFlt : float
```

- Finding files with selection by metadata:

```
find <meta selection>
```

For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg> find ATMetaInt=10,11 ATMetaInt1<15
Query: {'ATMetaInt': {'in': [10, 11]}, 'ATMetaInt1': {'<': 15}}
/vo.formation.idgrilles.fr/user/a/atsareg/newDir/wms_output.py
```

### 8.2 File Provenance Metadata

In the File Catalog you can declare ancestor files for a given file. This is often needed in order to keep track of the derived data provenance path. The ancestor declaration is done as following:

```
ancestorset <descendent> <ancestor>
```

For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg> ancestorset file2 file1
FC:/vo.formation.idgrilles.fr/user/a/atsareg> ancestorset file3 file2
```

Once the chain of ancestors/descendents is created it can be interrogated with the following commands:

```
ancestor <file> <depth>
descendent <file> <depth>
```

For example:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg> ancestor file3 2
/vo.formation.idgrilles.fr/user/a/atsareg/file3
1 /vo.formation.idgrilles.fr/user/a/atsareg/file2
2 /vo.formation.idgrilles.fr/user/a/atsareg/file1

FC:/vo.formation.idgrilles.fr/user/a/atsareg> descendent file1 2
/vo.formation.idgrilles.fr/user/a/atsareg/file1
```
<table>
<thead>
<tr>
<th></th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/vo.formation.idgrilles.fr/user/a/atsareg/file2</td>
</tr>
<tr>
<td>2</td>
<td>/vo.formation.idgrilles.fr/user/a/atsareg/file3</td>
</tr>
</tbody>
</table>
2.1 DIRAC Setup Structure

The basic DIRAC components are *Databases*, *Services* and *Agents*. These components are combined together to form *Systems*.

- **Databases** keep the persistent state of a *System*. They are accessed by Services and Agents as a kind of shared memory.

- **Services** are passive components listening to incoming client requests and reacting accordingly by serving requested information from the *Database* backend or inserting requests on the *Database* backend. *Services* themselves can be clients of other *Services* from the same *DIRAC System* or from other *Systems*.

- **Agents** are the active components which are running continuously invoking periodically their execution methods. Agents are animating the whole system by executing actions, sending requests to the *DIRAC* or third party services.

- **System** is delivering a complex functionality to the rest of *DIRAC*, providing a solution for a given class of tasks. Examples of *Systems* are Workload Management System or Configuration System.

To achieve a functional *DIRAC* installation, cooperation of different *Systems* is required. A set of *Systems* providing a complete functionality to the end user form a *DIRAC Setup*. All *DIRAC* client installations will point to a particular *DIRAC Setup*. *Setups* can span multiple server installations. Each server installation belongs to a *DIRAC Instance* that can be shared by multiple *Setups*.

*Setup* is the highest level of the *DIRAC* component hierarchy. *Setups* are combining together instances of *Systems*. A given user community may have several *Setups*. For example, there can be “Production” *Setup* together with “Test” or “Certification” *Setups* used for development and testing of the new functionality. An instance of a *System* can belong to one or more *Setups*, in other words, different *Setups* can share some *System* instances. Multiple *Setups* for the given community share the same Configuration information which allows them to access the same computing resources.

Each *System* and *Setup* instance has a distinct name. The mapping of *Systems* to *Setups* is described in the Configuration of the *DIRAC* installation in the “/DIRAC/Setups” section.

**ToDo**

- image illustrating the structure
2.2 Getting Started

Before attempting a full installation this chapter attempts to guide a future DIRAC administrator over a Step by Step new DIRAC installation. Following the Steps in this chapter should allow you to better understand how it works the DIRAC installation you are preparing.

2.2.1 Step 1: Minimal Framework Installation

Before doing any DIRAC server installation you should have a look at DIRAC Server Installation, in particular the sections Requirements and Server preparation. After you have created the necessary directory structure and placed the host certificate in the proper location, you are ready for this first Step.

In this Step, the procedure for any server installation is shown. It consists of three different phases:

- Installation of the DIRAC code.
- Creation of the initial DIRAC local configuration file.
- Deployment of the necessary DIRAC components

The first 2 phases are common to all Steps. The code installation phase can be skipped since all components will use the same code. In some cases additional local configuration will be necessary, and thus the second phase will need to be repeated. While the third phase will always be necessary to add new functionality to the installation.

A Minimal DIRAC installation

The minimal set of components that required for a DIRAC server are a Configuration Server and System Administrator services. Additionally one can add the Security Logging and the Bundle Delivery services. The first one receives a summary of all connections received by all DIRAC services in the current installation. The second allows any DIRAC client to download an up-to-date version of CA's public keys and Certification Revocation List, CRL.

The way to achieve this minimal installation is the following:

- Download the dirac-install as described in Install script.
- Create a Step_1.cfg file using the following template and substituting strings within [ ] by appropriate values for your case:

```
# This section determines which DIRAC components will be installed and where
LocalInstallation
{
  # These are options for the installation of the DIRAC software
  # DIRAC release version (this is an example, you should find out the current
  # production release)
  Release = [The version to be installed. Default: HEAD]
  # Python version os the installation
  PythonVersion = 26
  # To install the Server version of DIRAC (the default is client)
  InstallType = server
  # LCG python bindings for SEs and LFC. Specify this option only if your installation
  # uses those services
  LcgVer = 2010-11-20
  # If this flag is set to yes, each DIRAC update will be installed
```
# in a separate directory, not overriding the previous ones
UseVersionsDir = yes
# The directory of the DIRAC software installation
TargetPath = /opt/dirac
# DIRAC extensions to be installed (Web is required if you are installing the Portal on
# this server).
# For each User Community their own extension might be necessary here:
# i.e. LHCB, LHCB-web for LHCB
Extensions = Web

# These are options for the configuration of the installed DIRAC software
# i.e., to produce the initial dirac.cfg for the server
# Give a Name to your User Community, it does not need to be the same name as in EGI,
# it can be used to cover more than one VO in the grid sense
VirtualOrganization = MyVO
# Site name: it should follow the convention [Infrastructure].[name].[country code]
SiteName = [The name for your installation site. I.e. DIRAC.ubuntu.es]
# Setup name
Setup = MyDIRAC-Production
# Default name of system instances
InstanceName = Production
# Flag to use the server certificates
UseServerCertificate = yes
# Do not download CAs, CRLs
SkipCADownload = yes
# Configuration Server URL (This should point to the URL of at least one valid Configuration
# Service in your installation, for the primary server it should not used)
ConfigurationServer = dips://localhost:9135/Configuration/Server
# Flag to set up the Configuration Server as Master (use only in the primary server)
ConfigurationMaster = yes
# Configuration Name
ConfigurationName = MyConfiguration

# These options define the DIRAC components to be installed on "this" DIRAC server.
# The next options should only be set for the primary server,
# they properly initialize the configuration data
# Name of the Admin user (default: None )
AdminUserName = [Your short name for the DIRAC installation. I.e. ricardo]
# DN of the Admin user certificate (default: None )
# In order the find out the DN that needs to be included in the Configuration for a given
# host or user certificate the following command can be used:
# openssl x509 -noout -subject -enddate -in <certfile.pem>
# OpenSSL
AdminUserDN = [The DN of your grid certificate. I.e. /DC=es/DC=irisgrid/O=ecm-ub/CN=Ricardo-Graciani-Diaz]
# Email of the Admin user (default: None )
AdminUserEmail = [Your email. I.e. graciani@ecm.ub.es]
# Name of the Admin group (default: dirac_admin )
# AdminGroupName = dirac_admin
# Name of the installation host (default: the current host )
# Used to build the URLs the services will publish
# This will only allow to make local tests on this installation
Host = localhost
# DN of the host certificate (default: None)
# In order to find out the DN that needs to be included in the Configuration for a given
# host or user certificate the following command can be used:
# openssl x509 -noout -subject -enddate -in <certfile.pem>
# HostDN = [The DN of the host grid certificate. I.e. /DC=ch/DC=cern/OU=computers/CN=volhcb19.cern.ch]

# Components to deploy
# Systems = Configuration, Framework
Services = Configuration/Server
Services += Framework/SecurityLogging
Services += Framework/BundleDelivery
Services += Framework/SystemAdministrator

• Execute the installation of the DIRAC code:

  > ./dirac-install Step_1.cfg

• Produce the initial configuration file:

  > source bashrc
  > dirac-configure Step_1.cfg

• Deploy the requested components:

  > dirac-setup-site

2.2.2 Step 2

2.3 DIRAC Server Installation

The procedure described here outlines the installation of the DIRAC components on a host machine, a DIRAC server. There are two distinct cases of installations:

• *Primary server installation*. This the first installation of a fresh new DIRAC system. No functioning Configuration Service is running yet (*Primary server installation*).

• *Additional server installation*. This is the installation of additional hosts connected to an already existing DIRAC system, with the Master Configuration Service already up and running on another DIRAC server (*Additional server installation*).

The primary server installation should install and start at least the Configuration Service which is the backbone for the entire DIRAC system. The SystemAdministrator Service, once installed, allows remote management of the DIRAC components on the server. In multi-server installations DIRAC components are distributed among a number of servers installed using the procedure for additional host installation.

For all DIRAC installations any number of client installations is possible.
2.3.1 Requirements

Server:

- 9130-9200 ports should be open in the firewall for the incoming TCP/IP connections (this is the default range if predefined ports are used, the port on which services are listening can be configured by the DIRAC administrator):

```
iptables -I INPUT -p tcp --dport 9130:9200 -j ACCEPT
service iptables save
```

- For the server hosting the portal, ports 80 and 443 should be open and redirected to ports 8080 and 8443 respectively, i.e. setting iptables appropriately:

```
iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080
iptables -t nat -I PREROUTING -p tcp --dport 443 -j REDIRECT --to-ports 8443
```

If you have problems with NAT or iptables you can use multipurpose relay `socat`:

```
socat TCP4-LISTEN:80,fork TCP4:localhost:8080 &
socat TCP4-LISTEN:443,fork TCP4:localhost:8443 &
```

- Grid host certificates in pem format;
- At least one of the servers of the installation must have updated CAs and CRLs files; if you want to install the standard Grid CAs you can follow the instructions at https://wiki.egi.eu/wiki/EGI_IGTF_Release. They are usually installed `/etc/grid-security/certificates`. You may also need to install the `fetch-crl` package, and run the `fetch-crl` command once installed.

- If gLite third party services are needed (for example, for the pilot job submission via WMS or for data transfer using FTS) gLite User Interface must be installed and the environment set up by “sourcing” the corresponding script, e.g. `/etc/profile.d/grid-env.sh`.

Client:

- User certificate and private key in .pem format in the `$HOME/.globus` directory with correct permissions.
- User certificate loaded into the Web Browser (currently supported browsers are: Mozilla Firefox, Chrome and Safari)

2.3.2 Server preparation

Any host running DIRAC server components should be prepared before the installation of DIRAC following the steps below. This procedure must be followed for the primary server and for any additional server installations.

- As `root` create a `dirac` user account. This account will be used to run all the DIRAC components:

```
adduser -s /bin/bash -d /home/dirac dirac
```

- As `root`, create the directory where the DIRAC services will be installed:

```
mkdir /opt/dirac
chown -R dirac:dirac /opt/dirac
```

- As `root`, check that the system clock is exact. Some system components are generating user certificate proxies dynamically and their validity can be broken because of the wrong system date and time. Properly configure the NTP daemon if necessary.

- As `dirac` user, create directories for security data and copy host certificate:
In case your host certificate is in the p12 format, you can convert it with:

```bash
openssl pkcs12 -in host.p12 -clcerts -nokeys -out hostcert.pem
openssl pkcs12 -in host.p12 -nocerts -nodes -out hostkey.pem
```

Make sure the permissions are set right correctly, such that the hostkey.pem is only readable by the `dirac` user.

- **As dirac user**, create a directory or a link pointing to the CA certificates directory, for example:

  ```bash
  ln -s /etc/grid-security/certificates /opt/dirac/etc/grid-security/certificates
  ```

  (this is only mandatory in one of the servers. Others can be synchronized from this one using DIRAC tools.)

- **As dirac user** download the `install_site.sh` script:

  ```bash
  mkdir /home/dirac/DIRAC
  cd /home/dirac/DIRAC
  ```

### 2.3.3 Server Certificates

Server certificates are used for validating the identity of the host a given client is connecting to. By default grid host certificate include host/ in the CN (common name) field. This is not a problem for DIRAC components since DISET only keeps the host name after the / if present.

However if the certificate is used for the Web Portal, the client validating the certificate is your browser. All browsers will rise a security alarm if the host name in the url does not match the CN field in the certificate presented by the server. In particular this means that `host/`, or other similar parts should not be present, and that it is preferable to use DNS aliases and request a certificate under this alias in order to be able to migrate the server to a new host without having to change your URLs. DIRAC will accept both real host names and any valid aliases without complaints.

Finally, you will have to instruct you users on the procedure to upload the public key of the CA signing the certificate of the host where the Web Portal is running. This depends from CA to CA, but typically only means clicking on a certain link on the web portal of the CA.

#### Using your own CA

This is mandatory on the server running the web portal.

In case the CA certificate is not coming from traditional sources (installed using a package manager), but installed “by hand”, you need to make sure the hash of that CA certificate is created. Make sure the CA certificate is located under `/etc/grid-security/certificates`, then do the following as root:

```bash
cd /etc/grid-security/certificates
openssl x509 -noout -in cert.pem -hash
ln -s cert.pem hash.0
```

where the output of the `openssl` command gives you the hash of the certificate `cert.pem`, and must be used for the `hash.0` link name. Make sure the `.0` part is present in the name, as this is looked for when starting the web server.
2.3.4 Primary server installation

The installation consists of setting up a set of services, agents and databases for the required DIRAC functionality. The SystemAdministrator interface can be used later to complete the installation by setting up additional components. The following steps should be taken:

- Editing the installation configuration file. This file contains all the necessary information describing the installation. By editing the configuration file one can describe the complete DIRAC server or just a subset for the initial setup. Below is an example of a commented configuration file. This file corresponds to a minimal DIRAC server configuration which allows to start using the system:

```plaintext
# This section determines which DIRAC components will be installed and where
LocalInstallation {  
  # These are options for the installation of the DIRAC software
  # DIRAC release version (this is an example, you should find out the current production release)
  Release = v6r10p4
  # Python version of the installation
  PythonVersion = 26
  # To install the Server version of DIRAC (the default is client)
  InstallType = server
  # LCG python bindings for SEs and LFC. Specify this option only if your installation uses those services
  # LcgVer = 2012-02-20
  # If this flag is set to yes, each DIRAC update will be installed in a separate directory, not overriding the previous ones
  UseVersionsDir = yes
  # The directory of the DIRAC software installation
  TargetPath = /opt/dirac
  # DIRAC extra modules to be installed (Web is required if you are installing the Portal on this server).
  # Only modules not defined as default to install in their projects need to be defined here:
  # i.e. LHCb, LHCbWeb for LHCb
  ExtraModules = Web

  # These are options for the configuration of the installed DIRAC software
  # i.e., to produce the initial dirac.cfg for the server
  # Give a Name to your User Community, it does not need to be the same name as in EGI,
  # it can be used to cover more than one VO in the grid sense
  VirtualOrganization = Name of your VO
  # Site name
  SiteName = DIRAC.HostName.ch
  # Setup name
  Setup = MyDIRAC-Production
  # Default name of system instances
  InstanceName = Production
  # Flag to skip download of CAs, on the first Server of your installation you need to get CAs installed by some external means
  SkipCADownload = yes
  # Flag to use the server certificates
  UseServerCertificate = yes
  # Configuration Server URL (This should point to the URL of at least one valid Configuration Server)
}
```

2.3. DIRAC Server Installation
DIRAC Documentation, Release integration

```bash
# Service in your installation, for the primary server it should not used )
# ConfigurationServer = dips://myprimaryserver.name:9135/Configuration/Server
# Configuration Name
ConfigurationName = MyConfiguration

# These options define the DIRAC components to be installed on "this" DIRAC server.
#
# The next options should only be set for the primary server,
# they properly initialize the configuration data
#
# Name of the Admin user (default: None )
AdminUserName = adminusername
# DN of the Admin user certificate (default: None )
# In order the find out the DN that needs to be included in the Configuration for a given
# host or user certificate the following command can be used::
# openssl x509 -noout -subject -enddate -in <certfile.pem>
# AdminUserDN = /DC=ch/aminDN
# Email of the Admin user (default: None )
AdminUserEmail = adminmail@provider
# Name of the Admin group (default: dirac_admin )
AdminGroupName = dirac_admin
# DN of the host certificate (*) (default: None )
HostDN = /DC=ch/DC=country/OU=computers/CN=computer.dn
# Define the Configuration Server as Master for your installations
ConfigurationMaster = yes

# The following options define components to be installed
#
# Name of the installation host (default: the current host )
# Used to build the URLs the services will publish
# For a test installation you can use 127.0.0.1
# Host = dirac.cern.ch
Host =
# List of Services to be installed
Services = Configuration/Server
Services += Framework/SystemAdministrator
# Flag determining whether the Web Portal will be installed
WebPortal = yes
#
# The following options defined the MySQL DB connectivity
#
# The following option define if you want or not install the mysql that comes with DIRAC on the
# machine
# InstallMySQL = True
Database
{
  # User name used to connect the DB server
  User = Dirac # default value
  # Password for database user acess. Must be set for SystemAdministrator Service to work
  Password = XXXX
  # Password for root DB user. Must be set for SystemAdministrator Service to work
  RootPwd = YYYY
  # location of DB server. Must be set for SystemAdministrator Service to work
  Host = localhost # default
```

100 Chapter 2. Administrator Guide
# There are 2 flags for small and large installations Set either of them to True/yes when appropriate
# MySQLSmallMem: Configure a MySQL with small memory requirements for testing purposes
# innodb_buffer_pool_size=200MB
# MySQLLargeMem: Configure a MySQL with high memory requirements for production purposes
# innodb_buffer_pool_size=10000MB

- Run `install_site.sh` giving the edited configuration file as the argument. The configuration file must have .cfg extension (CFG file):

  ```bash
  ./install_site.sh install.cfg
  ```

- If the installation is successful, in the end of the script execution you will see the report of the status of running DIRAC services, e.g.:

<table>
<thead>
<tr>
<th>Name</th>
<th>Run</th>
<th>Uptime</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration_Server</td>
<td>Run</td>
<td>41</td>
<td>30268</td>
</tr>
<tr>
<td>Framework_SystemAdministrator</td>
<td>Run</td>
<td>21</td>
<td>30339</td>
</tr>
<tr>
<td>Web_httpd</td>
<td>Run</td>
<td>5</td>
<td>30828</td>
</tr>
<tr>
<td>Web_paster</td>
<td>Run</td>
<td>5</td>
<td>30829</td>
</tr>
</tbody>
</table>

Now the basic services - Configuration and SystemAdministrator - are installed. The rest of the installation can proceed using the DIRAC Administrator interface, either command line (System Administrator Console) or using Web Portal (eventually, not available yet).

It is also possible to include any number of additional systems, services, agents and databases to be installed by “install_site.sh”.

**Important Notice:** after executing `install_site.sh` (or `dirac-setup-site`) a runsvdir process is kept running. This is a watchdog process that takes care to keep DIRAC component running on your server. If you want to remove your installation (for instance if you are testing your install .cfg) you should first remove links from startup directory, kill the runsvdir, the runsv processes:

```bash
#!/bin/bash
source /opt/dirac/bashrc
RUNSVCTRL=`which runsvctrl`
chpst -u dirac $RUNSVCTRL d /opt/dirac/startup/*
killall runsv svlogd
killall runsvdir
# If you did also installed a MySQL server uncomment the next line
dirac-stop-mysql
```

## 2.3.5 Additional server installation

To add a new server to an already existing DIRAC Installation the procedure is similar to the one above. You should perform all the preliminary steps to prepare the host for the installation. One additional operation is the registration of the new host in the already functional Configuration Service.

- Then you edit the installation configuration file:

  ```bash
  # This section determines which DIRAC components will be installed and where
  # LocalInstallation
  ```

2.3. **DIRAC Server Installation**
# DIRAC release version (this is an example, you should find out the current production release)
Release = v6r3p7
# To install the Server version of DIRAC (the default is client)
InstallType = server
# LCG python bindings for SEs and LFC. Specify this option only if your installation uses those services
# LcgVer = 2012-02-20
# If this flag is set to yes, each DIRAC update will be installed in a separate directory, not overriding the previous ones
UseVersionsDir = yes
# The directory of the DIRAC software installation
TargetPath = /opt/dirac
# DIRAC extra packages to be installed (Web is required if you are installing the Portal on this server).
# For each User Community their extra package might be necessary here:
#  i.e. LHCb, LHCbWeb for LHCb
ExtraModules =

# These are options for the configuration of the previously installed DIRAC software
# i.e., to produce the initial dirac.cfg for the server
# Give a Name to your User Community, it does not need to be the same name as in EGI,
# it can be used to cover more than one VO in the grid sense
VirtualOrganization = Name of your VO
# Site name
SiteName = DIRAC.HostName2.ch
# Setup name
Setup = MyDIRAC-Production
# Default name of system instances
InstanceName = Production
# Flag to use the server certificates
UseServerCertificate = yes
# Configuration Server URL (This should point to the URL of at least one valid Configuration Service in your installation, for the primary server it should not used)
ConfigurationServer = dips://myprimaryserver.name:9135/Configuration/Server
ConfigurationServer += dips://localhost:9135/Configuration/Server
# Configuration Name
ConfigurationName = MyConfiguration

# These options define the DIRAC components being installed on "this" DIRAC server.
# The simplest option is to install a slave of the Configuration Server and a SystemAdministrator for remote management.
# The following options defined components to be installed
# Name of the installation host (default: the current host )
# Used to build the URLs the services will publish
# Host = dirac.cern.ch
Host =
# List of Services to be installed
Services = Configuration/Server
Services += Framework/SystemAdministrator

• Now run install_site.sh giving the edited CFG file as the argument::
If the installation is successful, the SystemAdministrator service will be up and running on the server. You can now set up the required components as described in Setting up DIRAC services and agents using the System Administrator Console

2.3.6 Post-Installation step

In order to make the DIRAC components running we use the runit mechanism (http://smarden.org/runit/). For each component that must run permanently (services and agents) there is a directory created under \opt\dirac\startup that is monitored by a runsvdir daemon. The installation procedures above will properly start this daemon. In order to ensure starting the DIRAC components at boot you need to add a hook in your boot sequence. A possible solution is to add an entry in the /etc/inittab:

```
SV:123456:respawn:/opt/Dirac/sbin/runsvdir-start
```

or if using upstart (in RHEL6 for example), add a file /etc/init/dirac.conf containing:

```
start on runlevel [123456]
stop on runlevel [0]

respawn
exec /opt/Dirac/sbin/runsvdir-start
```

On specific machines, or if network is needed, it’s necessary to make sure the runsvdir_start script is executed after a certain service is started. For example, on Amazon EC2, I recommend changing the first line by:

```
start on started elastic-network-interfaces
```

Together with a script like (it assumes that in your server DIRAC is using dirac local user to run):

```
#!/bin/bash
source /opt/dirac/bashrc
RUNSVCTRL=`which runsvctrl`
chpst -u dirac $RUNSVCTRL d /opt/dirac/startup/*
killsall runsv svlogd
killsall runsvdir
/opt/dirac/pro/mysql/share/mysql/mysql.server stop --user=dirac
sleep 10
/opt/dirac/pro/mysql/share/mysql/mysql.server start --user=dirac
sleep 20
RUNSVDIR=`which runsvdir`
exec chpst -u dirac $RUNSVDIR -P /opt/dirac/startup 'log: DIRAC runsv'
```

The same script can be used to restart all DIRAC components running on the machine.

2.3.7 Setting up DIRAC services and agents using the System Administrator Console

To use the System Administrator Console, you will need first to install the DIRAC Client software on some machine. To install the DIRAC Client, follow the procedure described in the User Guide.

- Start admin command line interface using administrator DIRAC group:

```
dirac-proxy-init -g dirac_admin
dirac-admin-sysadmin-cli --host <HOST_NAME>
```
where the HOST_NAME is the name of the DIRAC service host

• At any time you can use the help command to get further details:

```
dirac.pic.es >help
```

Documented commands (type help <topic>):
========================================
add execfile install restart show stop
exec exit quit set start update

Undocumented commands:
======================
help

• Add instances of DIRAC systems which service or agents will be running on the server, for example:

```
add instance WorkloadManagement Production
```

• Install MySQL database. You have to enter two passwords one is the root password for MySQL itself (if not already done in the server installation) and another one is the password for user who will own the DIRAC databases, in our case the user name is Dirac:

```
install mysql
MySQL root password:
MySQL Dirac password:
```

• Install databases, for example:

```
install db ComponentMonitoringDB
```

• Install services and agents, for example:

```
install service WorkloadManagement JobMonitoring
```

```
install agent Configuration CE2CSAgent
```

Note that all the necessary commands above can be collected in a text file and the whole installation can be accomplished with a single command:

```
execfile <command_file>
```

### 2.3.8 Component Configuration and Monitoring

At this point all the services should be running with their default configuration parameters. To change the components configuration parameters

• Login into web portal and choose dirac_admin group, you can change configuration file following these links:

```
Systems -> Configuration -> Manage Configuration
```

• Use the command line interface to the Configuration Service:

```
$ *dirac-configuration-cli*
```

• In the server all the logs of the services and agents are stored and rotated in files that can be checked using the following command:
tail -f /opt/dirac/startup/<System>_<Service or Agent>/log/current

# 2.4 Install WebAppDIRAC

## 2.4.1 Web configuration file

We use `web.cfg` configuration file. The structure of the `web.cfg` file is the following:

```plaintext
WebApp
{
    Balancer = None # nginx in case you have installed nginx
    #NumProcesses = 1
    #SSLProtocol = "" [PROTOCOL_SSLv2, PROTOCOL_SSLv23, PROTOCOL_SSLv3, PROTOCOL_TLSv1] in case you do not want to use the default protocol
    Theme = tabs # [desktop]

    Schema
    {
        Tools{
            Proxy Upload = DIRAC.ProxyUpload
            Job Launchpad = DIRAC.JobLaunchpad
            Notepad = DIRAC.Notepad
        }
        OldPortal{
            Request Manager = link|https://lhcb-web-dirac.cern.ch/DIRAC/LHCb-Production/lhcb_user/ProductionRequest/display
        }
        Applications
        {
            Public State Manager = DIRAC.PublicStateManager
            Job Monitor = DIRAC.JobMonitor
            Pilot Monitor = DIRAC.PilotMonitor
            Accounting = DIRAC.AccountingPlot
            Configuration Manager = DIRAC.ConfigurationManager
            Registry Manager = DIRAC.RegistryManager
            File Catalog = DIRAC.FileCatalog
            System Administration = DIRAC.SystemAdministration
            Activity Monitor = DIRAC.ActivityMonitor
            Transformation Monitor = DIRAC.TransformationMonitor
            Request Monitor = DIRAC.RequestMonitor
            Pilot Summary = DIRAC.PilotSummary
            Resource Summary = DIRAC.ResourceSummary
            Site Summary = DIRAC.SiteSummary
            Proxy Manager = DIRAC.ProxyManager
            #ExampleApp = DIRAC.ExampleApp
        }
        DIRAC = link|http://diracgrid.org
    }
}
```

Define external links:

```plaintext
Web
{
    Lemon Host Monitor
    {
        volhcbo1 = link|https://lemonweb.cern.ch/lemon-web/info.php?entity=1vbox01&detailed=yes
    }
}
```
2.4.2 Install and configure NGINX

The official site of NGINX is the following: http://nginx.org/ The required NGINX version has to be greater than 1.4.

- Install Nginx using package manager:

  ```bash
  yum install nginx
  ```

- Manual install
  1. `wget http://nginx.org/download/nginx-1.6.0.tar.gz`
  2. `cd nginx-1.6.0`
  3. `./configure`
  4. `make`
  5. `sudo make install` (without sudo you have to specify the installation directory)

- Configure NGINX

  In the installed directory of NGINX you have to edit the nginx.conf file. In our installation it is under `/usr/local/nginx/conf` directory. You have to delete part of the nginx.conf file starting from `#gzip on;` line

  ```
  #keepalive_timeout 0;
  keepalive_timeout 65;
  #gzip on;
  server {
    ....
  }
  ```

  to the end of file. Note: DO NOT delete `}` You have to add the following line:

  ```
  include site.conf;
  ```

The content of the nginx.conf (/usr/local/nginx/conf/nginx.conf):

```
#user nobody;
worker_processes 2;

#error_log  logs/error.log;
#error_log  logs/error.log notice;
#error_log  logs/error.log  info;
#pid logs/nginx.pid;

events {
  worker_connections  1024;
}

http {
  include  mime.types;
  default_type application/octet-stream;

  #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
  #    '$status $body_bytes_sent "$http_referer"'
  #    '"$http_user_agent" "$http_x_forwarded_for"';
```

```
#access_log logs/access.log main;

sendfile on;
#tcp_nopush on;

#keepalive_timeout 0;
keepalive_timeout 65;

#gzip on;
include site.conf;

You have to copy and paste under /usr/local/nginx/conf directory and please modify the content according to your installation:

upstream tornadoserver {
    #One for every tornado instance you're running that you want to balance
    server 127.0.0.1:8000;
}

server {
    listen 80;

    #Your server name if you have weird network config. Otherwise leave commented
    server_name volhcb25.cern.ch;

    root /opt/dirac/WebPrototype/webRoot;

    location ~ ^/[a-zA-Z-]+/(s:.*/g:.*/)?static/(.+.(jpg|jpeg|gif|png|bmp|ico|pdf))$ {
        alias /opt/dirac/WebPrototype/;
        #Add one more for every static path. For instance for LHCbWebDIRAC:
        try_files LHCbWebDIRAC/WebApp/static/$2 WebAppDIRAC/WebApp/static/$2 /;
        #try_files WebAppDIRAC/WebApp/static/$2 /;
        expires 10d;
        gzip_static on;
        gzip_disable "MSIE [1-6].";
        add_header Cache-Control public;
        break;
    }

    location ~ ^/[a-zA-Z-]+/(s:.*/g:.*/)?static/(.+)$ {
        alias /opt/dirac/WebPrototype/;
        #Add one more for every static path. For instance for LHCbWebDIRAC:
        try_files LHCbWebDIRAC/WebApp/static/$2 WebAppDIRAC/WebApp/static/$2 /;
        expires 1d;
        gzip_static on;
        gzip_disable "MSIE [1-6].";
        add_header Cache-Control public;
        break;
    }

    location ~ /DIRAC/ {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
    }

2.4. Install WebAppDIRAC
proxy_set_header X-Scheme $scheme;
proxy_pass http://tornadoserver;
proxy_read_timeout 3600;
proxy_send_timeout 3600;

gzip on;
gzip_proxied any;
gzip_comp_level 9;
gzip_types text/plain text/css application/javascript application/xml application/json;

# WebSocket support (nginx 1.4)
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
break;
}

location / {
  rewrite ^ http://$server_name/DIRAC/ permanent;
}

server {
  listen 443 default ssl; ## listen for ipv4
  server_name volhcb25.cern.ch;

  #Certs that will be shown to the user connecting to the web.
  #Preferably NOT grid certs. Use something that the user cert will not complain about
  ssl_certificate /opt/dirac/etc/grid-security/hostcert.pem;
  ssl_certificate_key /opt/dirac/etc/grid-security/hostkey.pem;
  ssl_client_certificate /opt/dirac/pro/etc/grid-security/allCAs.pem;
  ssl_verify_client on;
  ssl_verify_depth 10;
  ssl_session_cache shared:SSL:10m;
  root /opt/dirac/WebPrototype;

  location ~ ^/[a-zA-Z-]+/(s:.*/g:.*/)?static/.+\.(jpg|jpeg|gif|png|bmp|ico|pdf) {
    alias /opt/dirac/WebPrototype/;
    #Add one more for every static path. For instance for LHCbWebDIRAC:
    try_files LHCbWebDIRAC/WebApp/static/$2 WebAppDIRAC/WebApp/static/$2 /;
    #try_files WebAppDIRAC/WebApp/static/$2 /;
    expires 10d;
    gzip_static on;
    gzip_disable "MSIE [1-6]\.";
    add_header Cache-Control public;
    break;
  }

  location ~ ^/[a-zA-Z-]+/(s:.*/g:.*/)?static/.+ {
    alias /opt/dirac/WebPrototype/;
    #Add one more for every static path. For instance for LHCbWebDIRAC:
    try_files LHCbWebDIRAC/WebApp/static/$2 WebAppDIRAC/WebApp/static/$2 /;
    #try_files WebAppDIRAC/WebApp/static/$2 /;
expires 1d;
gzip_static on;
gzip_disable "MSIE [1-6]\.";
add_header Cache-Control public;
break;
}

location ~ /DIRAC/ {
    proxy_pass_header Server;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Scheme $scheme;
    proxy_pass http://tornadoserver;
    proxy_read_timeout 3600;
    proxy_send_timeout 3600;
    proxy_set_header X-Ssl_client_verify $ssl_client_verify;
    proxy_set_header X-Ssl_client_s_dn $ssl_client_s_dn;
    proxy_set_header X-Ssl_client_i_dn $ssl_client_i_dn;

    gzip on;
    gzip_proxied any;
    gzip_comp_level 9;
    gzip_types text/plain text/css application/javascript application/xml application/json;

    # WebSocket support (nginx 1.4)
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    break;
}
}

location / {
    rewrite ^ https://$server_name/DIRAC/ permanent;
}
}

You have to use the genCAsFile.sh to generate the following file: ssl_client_certificate
/opt/dirac/pro/etc/grid-security/allCAs.pem; The content of the genCAsFile.sh file is the following:

#!/bin/bash

gsCerts=/etc/grid-security/certificates
allF="/opt/dirac/etc/grid-security/allCAs.pem"
copiedCAs=0
invalidCAs=0
echo "Copying CA certificates into $allF"
for cert in $gsCerts/*.0
    do
        ossle="openssl x509 -noout -in $cert"
        if $ossle -checkend 3600; then
            openssl x509 -in $cert >> $allF.gen
            copiedCAs=`expr "$copiedCAs" + "1"`
        else
            echo " - CA $cert is expired"
        fi
    done
}

2.4. Install WebAppDIRAC
You have to add to the web.cfg the following lines:

```
DevelopMode = False
Balancer = nginx
NumProcesses = 1
```

The last step is to create /etc/init.d/nginx and add to this file the following lines:

```
#!/bin/sh
#
# nginx - this script starts and stops the nginx daemon
#
# chkconfig: - 85 15
# description: Nginx is an HTTP(S) server, HTTP(S) reverse \\ proxy and IMAP/POP3 proxy server
# processname: nginx
# config: /etc/nginx/nginx.conf
# config: /etc/sysconfig/nginx
# pidfile: /var/run/nginx.pid
#
# Source function library.
. /etc/rc.d/init.d/functions
#
# Source networking configuration.
. /etc/sysconfig/network
#
# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0

nginx="/usr/local/nginx/sbin/nginx"
prog=$(basename $nginx)

NGINX_CONF_FILE="/etc/nginx/nginx.conf"
NGINX_CONF_FILE="/usr/local/nginx/conf/nginx.conf"
[ -f /etc/sysconfig/nginx ] && . /etc/sysconfig/nginx

lockfile=/var/lock/subsys/nginx

make_dirs() {
  # make required directories
  #user=`$nginx -V 2>&1 | grep "configure arguments:" | sed 's/[^*]*--user=\([^ ]*\)\([^/]*\)/\1/g'`
  #if [ "-z "$user" ]; then
  #  useradd -M -s /bin/nologin $user
  #fi
  #options=`$nginx -V 2>&1 | grep 'configure arguments:'`
  #for opt in $options; do
  #  value=`echo $opt | grep '.*-temp-path'`;
  #  if [ "$value" ]; then
  #    mkdir -p $value && chown -R $user $value
  #fi
}
start() {
  [ -x $nginx ] || exit 5
  [ -f $NGINX_CONF_FILE ] || exit 6
  make_dirs
  echo -n "$Starting $prog: 
  daemon $nginx -c $NGINX_CONF_FILE
  retval=$?
  echo
  [ $retval -eq 0 ] && touch $lockfile
  return $retval
}

stop() {
  echo -n "$Stopping $prog: 
  killproc $prog -QUIT
  retval=$?
  echo
  [ $retval -eq 0 ] && rm -f $lockfile
  return $retval
}

restart() {
  configtest || return $?
  stop
  sleep 1
  start
}

reload() {
  configtest || return $?
  echo -n "$Reloading $prog: 
  killproc $nginx -HUP
  RETVAL=$?
  echo
}

force_reload() {
  restart
}

configtest() {
  $nginx -t -c $NGINX_CONF_FILE
}

rh_status() {
  status $prog
}

rh_status_q() {
  rh_status >/dev/null 2>&1
}
• Start, Stop and restart nginx:

```
/etc/init.d/nginx start|stop|restart
```

### 2.4.3 Nginx and CRLs

You can configure Nginx to check the certificate revoked list. You have to generate `allRevokedCerts.pem` file. You can use the following simple `bash` script to generate the file:

```bash
#!/bin/bash

gsCerts=/etc/grid-security/certificates

allF="/opt/dirac/etc/grid-security/allRevokedCerts.pem"
copiedCAs=0
invalidCAs=0
echo "Copying revoked certificates into $allF"
for cert in $gsCerts/*.r0
do
    openssl crl -in $cert >> $allF.gen
copiedCAs=`expr "$copiedCAs" + "1"`
done
echo " + Copied $copiedCAs revoked certificates into $allF"

mv $allF.gen $allF
```

Note: you can use a chron job to generate the `allRevokedCerts.pem` file.

You have to add the `site.conf` the following line:
2.4.4 Install WebAppDIRAC

- Install:
  - python dirac-install -t server $installCfg
  - source $installDir/bashrc
  - dirac-configure $installCfg $DEBUG
  - dirac-setup-site $DEBUG

$installCfg:

LocalInstallation
{
  # These are options for the installation of the DIRAC software
  # DIRAC release version (this is an example, you should find out the current
  # production release)
  Release = v8r0p24
  # Python version of the installation
  PythonVersion = 26
  # To install the Server version of DIRAC (the default is client)
  InstallType = server
  # LCG python bindings for SEs and LFC. Specify this option only if your installation
  # uses those services
  LcgVer = 2013-09-24
  # If this flag is set to yes, each DIRAC update will be installed
  # in a separate directory, not overriding the previous ones
  UseVersionsDir = yes
  # The directory of the DIRAC software installation
  TargetPath = /Users/zoltanmathe/newweb
  # DIRAC extra modules to be installed (Web is required if you are installing the Portal on
  # this server).
  # Only modules not defined as default to install in their projects need to be defined here:
  # i.e. LHCb, LHCbWeb for LHCb
  ExtraModules = WebAppDIRAC,LHCb,LHCbWeb
  Project = LHCb
  WebApp = yes
}

- Update using: dirac-admin-sysadmin-cli
  - dirac-admin-sysadmin-cli -H hostname
  - update version of DIRAC, for example v8r1

2.5 System Administrator Console

The System Administrator Console (SAC) is the interface which allows a system administrator to connect to any
DIRAC server which is running a SystemAdministrator service. This interface allows to perform all the system
maintenance tasks remotely.
2.5.1 Starting SAC

The SAC is invoked using dirac-admin-sysadmin-cli command for a given DIRAC server, for example:

```
dirac-admin-sysadmin-cli --host volhcb01.cern.ch
```

This starts a special shell with a number of commands defined. There is a help available to see the list of commands and get info about particular commands:

```
volhcb01.cern.ch>help

Documented commands (type help <topic>):
========================================
add  execfile  install  restart  show  stop
exec  exit  quit  set  start  update

volhcb01.cern.ch>help set

   Set the host to be managed

   usage:
   
   set host <hostname>
```

2.5.2 Getting information

The following command shows information about the host setup and currently used DIRAC software and extensions:

```
volhcb03.cern.ch> show info

Setup: LHCb-Certification
DIRAC version: v5r12-pre9
LHCb version v5r11p10
LHCWeb version v1r1
```

One can look up details of the software installed with the following command:

```
volhcb01.cern.ch> show software

{'Agents': {'Configuration': ['CE2CSAgent', 'UsersAndGroups'], 'DataManagement': ['TransferAgent', 'LFCvsSEAgent', 'ReplicationScheduler', 'FTSRegisterAgent', ...

```

It will show all the components for which the software is available on the host, so these components can be installed and configured for execution. The information is grouped by component type (Agents or Services) and by system. See below for how to setup the DIRAC components for running.

The status of the installed components can be obtained like:

```
volhcb01.cern.ch> show status

<table>
<thead>
<tr>
<th>System</th>
<th>Name</th>
<th>Type</th>
<th>Setup</th>
<th>Installed</th>
<th>Runit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceStatus</td>
<td>ResourceStatus</td>
<td>service</td>
<td>SetUp</td>
<td>Installed Down 2532910 0</td>
<td></td>
</tr>
<tr>
<td>WorkloadManagement</td>
<td>SandboxStore</td>
<td>service</td>
<td>SetUp</td>
<td>Installed Run</td>
<td></td>
</tr>
</tbody>
</table>
The output of the command shows for each component its system, name and type as well as the status information:

- Setup status shows if the component is set up for running on the host. It can take two values:SetUp/NotSetUp.
- Installed status shows if the component is installed on the host. This means that it is configured to run with Runit system.

Show setup command allows administrators to know which components, Services and Agents are setup up in the host:

```
mardirac1.in2p3.fr > show setup
{'Agents': ['CE2CSAgent'],
 'Framework': ['TopErrorMessagesReporter', 'SystemLoggingDBCLeaner', 'CAUpdateAgent'],
 'WorkloadManagement': ['JobHistoryAgent', 'InputDataAgent', 'StalledJobAgent', 'TaskQueueDirector', 'MightyOptimizer', 'PilotStatusAgent', 'JobCleaningAgent', 'StatesAccountingAgent'],
 'Services': {'Accounting': ['ReportGenerator', 'DataStore'], 'Configuration': ['Server'], 'Framework': ['Monitoring', 'BundleDelivery', 'SecurityLogging', 'Notification', 'UserProfileManager', 'SystemAdministrator', 'ProxyManager', 'SystemLogging'], 'RequestManagement': ['RequestManager'], 'WorkloadManagement': ['JobMonitoring', 'WMSAdministrator', 'SandboxStore', 'Matcher', 'JobStateUpdate', 'JobManager']}
```

SAC also allow which databases are installed:

```
mardirac1.in2p3.fr > show database
MySQL root password:
```

```
DataLoggingDB : Not installed
SandboxMetadataDB : Installed
JobDB : Installed
MPIJobDB : Not installed
FileCatalogDB : Installed
TransformationDB : Not installed
JobLoggingDB : Installed
UserProfileDB : Installed
```

Show the status of the MySQL server:
mardiracl.in2p3.fr >show mysql

FlushTables : 1
OpenTables : 47
NumberOfSlowQueries : 0
NumberOfQuestions : 24133
UpTime : 15763
NumberOfThreads : 13
NumberOfOpens : 203
QueriesPerSecond : 1.530

Is also possible to check logs for services and agents using SAC:

mardiracl.in2p3.fr>show log WorkloadManagement JobMonitoring
2011-03-16 14:28:15 UTC WorkloadManagement/JobMonitoring INFO: Sending records to security log service...
2011-03-16 14:28:15 UTC WorkloadManagement/JobMonitoring INFO: Data sent to security log service
2011-03-16 14:29:15 UTC WorkloadManagement/JobMonitoring INFO: Sending records to security log service...
2011-03-16 14:29:15 UTC WorkloadManagement/JobMonitoring INFO: Data sent to security log service

As of v6r13 it is now possible to check the history of installed components in DIRAC with show installations:

[sergiovm.cern.ch]> show installations
Num | Host | Name | Module | System | Date |
--- | --- | --- | --- | --- | --- |
1 | sergiovm.cern.ch | InstalledComponentsDB | InstalledComponentsDB | Framework | 01-06-2015 16:12 |
2 | sergiovm.cern.ch | ComponentMonitoring | ComponentMonitoring | Framework | 01-06-2015 16:12 |
3 | sergiovm.cern.ch | Server | Server | Configuration | 01-06-2015 16:12 |
4 | sergiovm.cern.ch | SystemAdministrator | SystemAdministrator | Framework | 01-06-2015 16:12 |

Accepted parameters by show installations:

- **list**: Changes the display mode of the results
- **current**: Show only the components that are still installed
- **-n <name>**: Show only installations of the component with the given name
- **-h <host>**: Show only installations in the given host
- **-s <system>**: Show only installations of components from the given system
- **-m <module>**: Show only installations of the given module
- **-t <type>**: Show only installations of the given type
- **-itb <date>**: Show installations made before the given date (‘dd-mm-yyyy’)
- **-ita <date>**: Show installations made after the given date (‘dd-mm-yyyy’)
- **-utb <date>**: Show installations of components uninstalled before the given date (‘dd-mm-yyyy’)
- **-uta <date>**: Show installations of components uninstalled after the given date (‘dd-mm-yyyy’)

### 2.5.3 Managing DIRAC services and agents

Using SAC the installation of DIRAC components (DBs, Services, Agents) and MySQL Server can be done.
Usage:

install mysql
install db <database>
install service <system> <service>
install agent <system> <agent>

To install MySQL server:

mardirac1.in2p3.fr >install mysql
Installing MySQL database, this can take a while ...
MySQL Dirac password:
MySQL: Already installed

Installation of Databases for services can be added:

mardirac1.in2p3.fr >install db MPIJobDB
Adding to CS WorkloadManagement/MPIJobDB
Database MPIJobDB from EELADIRAC/WorkloadManagementSystem installed successfully

Addition of new services:

mardirac1.in2p3.fr >install service WorkloadManagement MPIService
service WorkloadManagement_MPIService is installed, runit status: Run

Addition of new agents:

mardirac1.in2p3.fr >install agent Configuration CE2CSAgent
agent Configuration_CE2CSAgent is installed, runit status: Run

The SAC can also be used to start services or agents or database server.
Usage:

start <system|*> <service|agent|*>
start mysql

For example, start a service:

mardirac1.in2p3.fr >start WorkloadManagement MPIService
WorkloadManagement_MPIService started successfully, runit status:
WorkloadManagement_MPIService : Run

Restart services or agents or database server:

restart <system|*> <service|agent|*>
restart mysql

Restarting all the services and agents:

mardirac1.in2p3.fr >restart *
All systems are restarted, connection to SystemAdministrator is lost

Restarting a specific service or agent:

mardirac1.in2p3.fr >restart WorkloadManagement MPIService
WorkloadManagement_MPIService started successfully, runit status:
WorkloadManagement_MPIService : Run
Stop services or agents or database server:

```
stop <system|*> <service|agent|*>
stop mysql
```

Stop all the services and agents:

```
mardirac1.in2p3.fr >stop *
```

Stop a specific service or agent:

```
mardirac1.in2p3.fr >stop WorkloadManagement MPIService
```

WorkloadManagement_MPIService stopped successfully, runit status:

```
WorkloadManagement_MPIService : Down
```

### 2.5.4 Updating the DIRAC installation

The SAC allows to update the software on the target host to a given version.

Usage:

```
update <version>
```

For example:

```
$ dirac-admin-sysadmin-cli --host mardirac1.in2p3.fr
DIRAC Root Path = /home/vanessa/DIRAC-v5r12
mardirac1.in2p3.fr >update v5r12p7
Software update can take a while, please wait ...
Software successfully updated.
You should restart the services to use the new software version.
mardirac1.in2p3.fr >restart *
All systems are restarted, connection to SystemAdministrator is lost
mardirac1.in2p3.fr >quit
```

If the administrator needs to continue working with SAC, it must be started again.

### 2.6 DIRAC Configuration

The Configuration Service is providing the necessary information for the operations of a whole DIRAC Installation (which might include several *Setups*). In this section, the structure of the DIRAC Configuration and its contents are described. The procedure to add new configuration data and to update the existing settings is explained.

#### 2.6.1 DIRAC Configuration

The DIRAC Configuration information has a hierarchical structure and can come from different sources. This section describes the main sections of the DIRAC configuration and the way how this information is delivered to the consumers.
Configuration structure

The DIRAC Configuration is organized in a tree structure. It is divided in sections, which can also be seen as directories. Each section can contain other sections and options. The options are the leafs in the configuration tree, which contain the actual configuration data.

At the top level of the Configuration tree there are the following sections:

**DIRAC**  This section contains the most general information about the DIRAC installation.

**Systems**  This section provides configuration data for all the DIRAC Systems, their instances and components - services, agents and databases.

**Registry**  The Registry contains information about DIRAC users, groups and communities (VOs).

**Resources**  The Resources section provides description of all the DIRAC computing resources. This includes computing and storage elements as well as descriptions of several DIRAC and third party services.

**Operations**  This section collects various operational parameters needed to run the system.

The top level sections are described in details in dedicated chapters of the guide.

Configuration sources

The DIRAC Configuration can be defined in several places with strict rules how the settings are resolved by the clients. The possible configuration data sources are listed below in the order of preference of the option resolution:

**Command line options**  For all the DIRAC commands there is option ‘-o’ defined which takes one configuration option setting. For example:

```
dirac-wms-job-submit job.jdl -o /DIRAC/Setup=Dirac-Production
```

**Command line argument specifying a CFG file**  If a filename with the .cfg extension is passed as an argument to any DIRAC command it will be interpreted as a configuration file. For example:

```
dirac-wms-job-submit job.jdl my.cfg
```

**$HOME/.dirac.cfg**  This is the file in the user’s home directory with the CFG format

**$DIRACROOT/etc/dirac.cfg**  This is the configuration file in the root directory of the DIRAC installation

**Configuration Service**  Configuration data available from the global DIRAC Configuration Service

The client needing a configuration option is first looking for it in the command line arguments. If the option is not found, the search continues in the user configuration file, then in the DIRAC installation configuration file and finally in the Configuration Service. These gives a flexible mechanism of overriding global options by specific local settings.

2.6.2 Configuration System

The configuration file from DIRAC server is located under $DIRAC_ROOT_PATH/etc/<Conf Name>.cfg, this file is divided in sections and subsections.

A similar tree with the description of all the attributes is tried to be represented in this help tree.

**DIRAC Section**

In this section global attributes are configured.
Two subsections are part of DIRAC section:

- Configuration: In this subsection, access to Configuration servers is kept.
- Setups: Define the instance to be used for each the systems of each Setup.

**DIRAC / Configuration - Subsection**

This subsection is used to configure the Configuration Servers attributes. It should not edited by hand since it is updated by the Master Configuration Server to reflect the current situation of the system.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableAutoMerge</td>
<td>How many time the secondary servers are going to refresh configuration from master. Expressed as Integer and seconds as unit.</td>
<td>RefreshTime = 600</td>
</tr>
<tr>
<td>Name</td>
<td>Name of Configuration file</td>
<td>Name = Dirac-Prod</td>
</tr>
<tr>
<td>PropagationTime</td>
<td>Define the instance to be used for this setup</td>
<td>Accounting = Production</td>
</tr>
<tr>
<td>RequestManagement</td>
<td>Define the instance to be used for this setup</td>
<td>Configuration = Production</td>
</tr>
<tr>
<td>WorkloadManagement</td>
<td>Define the instance to be used for this setup</td>
<td>DataManagement = Production</td>
</tr>
<tr>
<td>StorageManagement</td>
<td>Define the instance to be used for this setup</td>
<td>Framework = Production</td>
</tr>
<tr>
<td>Accounting</td>
<td>Define the instance to be used for this setup</td>
<td>RequestManagement = Production</td>
</tr>
<tr>
<td>Configuration</td>
<td>Define the instance to be used for this setup</td>
<td>StorageManagement = Production</td>
</tr>
<tr>
<td>DataManagement</td>
<td>Define the instance to be used for this setup</td>
<td>WorkloadManagement = Production</td>
</tr>
<tr>
<td>Framework</td>
<td>Define the instance to be used for this setup</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Define the instance to be used for this setup</td>
<td></td>
</tr>
</tbody>
</table>

**DIRAC / Setups - Subsection**

In this subsection all the installed Setups are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>Define the instance to be used for this setup</td>
<td>Accounting = Production</td>
</tr>
<tr>
<td>Configuration</td>
<td>Define the instance to be used for this setup</td>
<td>Configuration = Production</td>
</tr>
<tr>
<td>DataManagement</td>
<td>Define the instance to be used for this setup</td>
<td>DataManagement = Production</td>
</tr>
<tr>
<td>Framework</td>
<td>Define the instance to be used for this setup</td>
<td>Framework = Production</td>
</tr>
<tr>
<td>RequestManagement</td>
<td>Define the instance to be used for this setup</td>
<td>RequestManagement = Production</td>
</tr>
<tr>
<td>StorageManagement</td>
<td>Define the instance to be used for this setup</td>
<td>StorageManagement = Production</td>
</tr>
<tr>
<td>WorkloadManagement</td>
<td>Define the instance to be used for this setup</td>
<td>WorkloadManagement = Production</td>
</tr>
</tbody>
</table>
For each Setup known to the installation, there must be a subsection with the appropriated name. Each option represents a DIRAC System available in the Setup and the Value is the instance of System that is used in that setup. For instance, since the Configuration is unique for the whole installation, all setups should have the same instance for the Configuration systems.

**DIRAC / Security - Subsection**

In this subsection security server configuration attributes are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CertFile</td>
<td>Directory where host certificate is located in the server.</td>
<td>CertFile = /opt/dirac/etc/grid-security/hostcert.pem</td>
</tr>
<tr>
<td>KeyFile</td>
<td>Directory where host key is located in the server.</td>
<td>KeyFile = /opt/dirac/etc/grid-security/hostcert.pem</td>
</tr>
<tr>
<td>SkipCAChecks</td>
<td>Boolean value this attribute allows to express if the CA certificates are or not be checked.</td>
<td>SkipCAChecks = No</td>
</tr>
<tr>
<td>UseServerCertificate</td>
<td>Use server certificate, expressed as boolean.</td>
<td>UseServerCertificate = yes</td>
</tr>
</tbody>
</table>

This section should only appear in the local dirac.cfg file of each installation, never in the central configuration.

**Operations - Section**

This section allows to configure options concerning to:

- Scheduling
- Pilots
- InputDataPolicy
- Job description
- Service Shifters
- Virtual Organization special parameters
- Transformations

In the short term, most of this schema will be moved into [vo]/[setup] dependent sections in order to allow better support for multi-VO installations.

**Operations / Email - Subsection**

In this subsection all the installed systems are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-Name</td>
<td>This attribute define the e-mail of the person in charge of the system</td>
<td>Production = <a href="mailto:hamar@cppm.in2p3.fr">hamar@cppm.in2p3.fr</a> Logging = <a href="mailto:hamar@cppm.in2p3.fr">hamar@cppm.in2p3.fr</a></td>
</tr>
</tbody>
</table>

**Operations / InputDataPolicy - Subsection**

In this subsection the Data Policy mechanism for files used in the JobWrapper are defined.
Operations / JobDescription - Subsection

JobDescription subsection describes allowed options in submitted payload (needs further documentation of supported fields).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowedJobTypes</td>
<td>List of users jobs accepted by the server</td>
<td>AllowedJobTypes = MPI AllowedJobTypes += User AllowedJobTypes += Test</td>
</tr>
</tbody>
</table>

Job Scheduling

The /Operations/<vo>/<setup>/JobScheduling section contains all parameters that define DIRAC’s behaviour when deciding what job has to be executed. Here’s a list of parameters that can be defined:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskQueueCPUTimeIntervals</td>
<td>Timeable cpu time values that the task queues can have.</td>
<td>360, 1800, 3600, 21600, 43200, 86400, 172800, 259200, 345600, 518400, 691200, 864000, 1080000</td>
</tr>
<tr>
<td>EnableSharesCorrection</td>
<td>Enable automatic correction of the priorities assigned to each task queue based on previous history</td>
<td>False</td>
</tr>
<tr>
<td>CheckJobLimits</td>
<td>Limit the amount of jobs running at sites based on their attributes</td>
<td>False</td>
</tr>
<tr>
<td>CheckMatchingDelay</td>
<td>Delay running a job at a site if another job has started recently and the conditions are met</td>
<td>False</td>
</tr>
</tbody>
</table>

Before enabling the correction of priorities, take a look at Job Priority Handling. Priorities and how to correct them is explained there. The configuration of the corrections would be defined under JobScheduling/ShareCorrections.

Limiting the number of jobs running Once JobScheduling/EnableJobLimits is enabled, DIRAC will check how many and what type of jobs are running at the configured sites. If there are more than a configured threshold, no more jobs of that type will run at that site. To define the limits create a JobScheduling/RunningLimit/<Site name> section for each site a limit has to be applied. Limits are defined by creating a section with the job attribute (like JobType name), and setting the limits inside. For instance, to define that there can’t be more than 150 jobs running with JobType=MonteCarlo at site DIRAC.Somewhere.co set JobScheduling/RunningLimit/DIRAC.Somewhere.co/JobType/MonteCarlo=150

Setting the matching delay DIRAC allows to throttle the amount of jobs that start at a given site. This throttling is defined under JobScheduling/MatchingDelay. It is configured similarly as the Limiting the number of jobs running. But instead of defining the maximum amount of jobs that can run at a site, the minimum seconds between starting jobs is defined. For instance JobScheduling/MatchingDelay/DIRAC.Somewhere.co/JobType/MonteCarlo=10 won’t allow jobs with JobType=MonteCarlo to start at site DIRAC.Somewhere.co with less than 10 seconds between them.

Example An example with all the options under JobScheduling follows. Remember that JobScheduling is defined under /Operations/<vo>/<setup>/JobScheduling for multi-VO installations, and /Operations/<setup>/JobScheduling for single-VO ones:
JobScheduling
{
  taskQueueCPUtimeIntervals = 360, 1800, 3600, 21600, 43200, 86400, 172800, 259200, 345600
  EnableSharesCorrection = True
  ShareCorrections
  {
    ShareCorrectorsToStart = WMSHistory
    WMSHistory
    {
      GroupsInstance
      {
        MaxGlobalCorrectionFactor = 3
        WeekSlice
        {
          TimeSpan = 604800
          Weight = 80
          MaxCorrection = 2
        }
        HourSlice
        {
          TimeSpan = 3600
          Weight = 20
          MaxCorrection = 5
        }
      }
      UserGroupInstance
      {
        Group = dirac_user
        MaxGlobalCorrectionFactor = 3
        WeekSlice
        {
          TimeSpan = 604800
          Weight = 80
          MaxCorrection = 2
        }
        HourSlice
        {
          TimeSpan = 3600
          Weight = 20
          MaxCorrection = 5
        }
      }
    }
  }
  CheckJobLimits = True
  RunningLimit
  {
    DIRAC.Somewhere.co
    {
      JobType
      {
        MonteCarlo = 150
        Test = 10
      }
    }
  }
  CheckMatchingDelay = True
  MatchingDelay
}
Pilot version

The /Operations/<vo>/<setup>/Pilot section define What version of DIRAC will be used to submit pilot jobs to the resources.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>What project version will be used</td>
<td>Version with which the component that submits pilot jobs is installed</td>
</tr>
<tr>
<td>Project</td>
<td>What installation project will be used when</td>
<td>DIRAC</td>
</tr>
<tr>
<td></td>
<td>submitting pilot jobs to the resources</td>
<td></td>
</tr>
<tr>
<td>Check-Version</td>
<td>Check if the version used by pilot jobs is the</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>one that they were submitted with</td>
<td></td>
</tr>
</tbody>
</table>

Operations / Shifter - Subsection

In this subsection managers are described for some systems. User credentials for the agents that will be used during execution of some tasks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Service_Manager&gt;</td>
<td>Name of service managers</td>
<td>Admin ProductionManager SAMManager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DataManager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User = vhamar</td>
</tr>
<tr>
<td>&lt;Service_Manager&gt;/User</td>
<td>DIRAC user name</td>
<td>User = vhamar</td>
</tr>
<tr>
<td>&lt;Service_Manager&gt;/Group</td>
<td>DIRAC user group</td>
<td>Group = dirac_admin</td>
</tr>
</tbody>
</table>

Agents requiring to act with a credential have always the option shifterProxy with a certain default: DataManager, ...

At each installation this default identity can be changed for each of them provided the corresponding section is created here.

The default identities currently used by DIRAC Agents are:

- **SAMManager**: Configuration/CE2CSAgent
- **DataManager**: DataManagement/FTSCleaningAgent, DataManagement/FTSMonitorAgent, DataManagement/FTSSubmitAgent, DataManagement/LFCvsSEAgent, DataManagement/RegistrationAgent, DataManagement/RemovalAgent, DataManagement/ReplicationScheduler, DataManagement/SEvSEFCAgent, DataManagement/TransferAgent, StorageManagement/MigrationMonitoringAgent, StorageManagement/MigrationRequestAgent, StorageManagement/SENamespaceCatalogCheckAgent, StorageManagement/SEPinRequestAgent, StorageManagement/SERequestFinalizationAgent, StorageManagement/SERequestPreparationAgent, StorageManagement/SEStageMonitorAgent, StorageManagement/SEStageRequestAgent, Transformation/MCExtensionAgent, Transformation/TransformationCleaningAgent, Transformation/ValidateOutputDataAgent
• **ProductionManager**: Transformation/RequestTaskAgent, Transformation/TransformationAgent, Transformation/WorkflowTaskAgent, WorkloadManagement/InputDataAgent

In general, to force any Agent to execute using a “manager” credential, instead of the certificate of the server it is only necessary to add a valid *shifterProxy* option in its configuration.

### Operations / VOs - Subsections

<VO_NAME> subsections allows to define pilot jobs versions for each setup defined for each VO supported by the server.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;VO_NAME&gt;</code></td>
<td>Subsection: Virtual organization name</td>
<td>vo.formation.idgrilles.fr</td>
</tr>
<tr>
<td><code>&lt;VO_NAME&gt;/&lt;SETUP_NAME&gt;/</code></td>
<td>Subsection: VO Setup name</td>
<td>Dirac-Production</td>
</tr>
<tr>
<td><code>&lt;VO_NAME&gt;/&lt;SETUP_NAME&gt;/Version/</code></td>
<td>Subsection: Version (Name fixed)</td>
<td>Version</td>
</tr>
<tr>
<td><code>&lt;VO_NAME&gt;/&lt;SETUP_NAME&gt;/Version/PilotVersion</code></td>
<td>DIRAC version to be installed for the pilots in the WNs</td>
<td>PilotVersion = v6r0-pre7</td>
</tr>
</tbody>
</table>

This section will progressively incorporate most of the other sections under /Operations in such a way that different values can be defined for each [VO] (in multi-VO installations) and [Setup]. A helper class is provided to access to these new structure.

```python
```

### Operations / Transformations - Subsection

### Operations / Transformations / Options

### Operations / TransformationPlugins / Options

### Registry - Section

This section allows to register users, hosts and groups in DIRAC way. Also some attributes applicable for all the configuration are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>DefaultGroup</em></td>
<td>Default user group to be used</td>
<td><em>DefaultGroup</em> = user</td>
</tr>
<tr>
<td><em>DefaultProxyTime</em></td>
<td>Default proxy time expressed in seconds</td>
<td><em>DefaultProxyTime</em> = 4000</td>
</tr>
</tbody>
</table>

### Registry / Groups - Subsections

This subsection is used to describe DIRAC groups registered in the server.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;GROUP_NAME&gt;</code></td>
<td>Subsection, represents the name of the group</td>
<td>dirac_user</td>
</tr>
<tr>
<td><code>&lt;GROUP_NAME&gt;/Users</code></td>
<td>DIRAC users logins than belongs to the group</td>
<td>Users = atsareg Users += msapunov</td>
</tr>
<tr>
<td><code>&lt;GROUP_NAME&gt;/Properties</code></td>
<td>Properties of the group, this will change the permissions of the group.</td>
<td>Properties = NormalUser</td>
</tr>
<tr>
<td><code>&lt;GROUP_NAME&gt;/VOMSRole</code></td>
<td>Role of the users in the VO</td>
<td>VOMSRole = /biomed</td>
</tr>
<tr>
<td><code>&lt;GROUP_NAME&gt;/VOMSVO</code></td>
<td>Virtual organization associated with the group</td>
<td>VOMSVVO = biomed</td>
</tr>
<tr>
<td>JobShare</td>
<td>Just for normal users</td>
<td>JobShare = 200</td>
</tr>
<tr>
<td>AutoUploadProxy</td>
<td>Controls automatic Proxy upload by dirac-proxy-init</td>
<td>AutoUploadProxy = True</td>
</tr>
<tr>
<td>AutoUploadPilotProxy</td>
<td>Controls automatic Proxy upload by dirac-proxy-init for Pilot groups</td>
<td>AutoUploadPilotProxy = True</td>
</tr>
<tr>
<td>AutoAddVOMS</td>
<td>Controls automatic addition of VOMS extension by dirac-proxy-init</td>
<td>AutoAddVOMS = True</td>
</tr>
</tbody>
</table>

- Default properties by group:

  ** dirac_admin:
  - Properties = AlarmsManagement
  - Properties += ServiceAdministrator
  - Properties += CSAdministrator
  - Properties += JobAdministrator
  - Properties += FullDelegation
  - Properties += ProxyManagement
  - Properties += Operator

  ** dirac_pilot
  - Properties = GenericPilot
  - Properties += LimitedDelegation
  - Properties += Pilot

  ** dirac_user
  - Properties = NormalUser

---

**Registry / Hosts - Subsections**

In this section each trusted hosts (DIRAC secondary servers) are described using simple attributes. An subsection called as DIRAC host name must be created and inside of this the following attributes must be included:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;DIRAC_HOST&gt;</code></td>
<td>Subsection DIRAC host name</td>
<td>host-dirac.in2p3.fr</td>
</tr>
<tr>
<td><code>&lt;DIRAC_HOST&gt;</code></td>
<td>Host distinguish name obtained from host certificate</td>
<td>DN = /O=GRID-FR/C=FR/O=CNRS/OU=CC-IN2P3/CN=dirac.in2p3.fr</td>
</tr>
</tbody>
</table>
Registry / Users - Subsections

In this section each user is described using simple attributes. An subsection with the DIRAC user name must be created. Some of the attributes than can be included are mandatory and others are considered as helpers:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DIRAC_USER_NAME&gt;/DN</td>
<td>Distinguish name obtained from user certificate</td>
<td>DN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=AndreiTsaregorodtsev</td>
</tr>
<tr>
<td>&lt;DIRAC_USER_NAME&gt;/CN</td>
<td>Canonical name of certification authority who sign the certificate.</td>
<td>CN = /C=FR/O=CNRS/CN=GRID2-FR</td>
</tr>
<tr>
<td>&lt;DIRAC_USER_NAME&gt;/Email</td>
<td>User e-mail (Mandatory)</td>
<td>Email = <a href="mailto:atsareg@in2p3.fr">atsareg@in2p3.fr</a></td>
</tr>
<tr>
<td>&lt;DIRAC_USER_NAME&gt;/mobile</td>
<td>Cellular phone number</td>
<td>mobile = +03062155555555</td>
</tr>
<tr>
<td>&lt;DIRAC_USER_NAME&gt;/Quota</td>
<td>Quota assigned to the user. Expressed in MBs.</td>
<td>Quota = 300</td>
</tr>
</tbody>
</table>

Registry / VO - Subsections

In this section each Virtual Organization (VO) is described in a dedicated subsection. The VO is a term coming from grid infrastructures where VO parameters are handled by the VOMS services. In DIRAC VO is not necessarily corresponding to some VOMS described VO. However, the VO options can include specific VOMS information. It is not mandatory for the DIRAC VO to have the same name as the corresponding VOMS VO. However, having these names the same can avoid confusions at the expense of having names longer than necessary.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;VO_NAME&gt;/VOAdmin</td>
<td>VO administrator user name</td>
<td>VOAdmin = joel</td>
</tr>
<tr>
<td>&lt;VO_NAME&gt;/VOMSName</td>
<td>VOMS VO name</td>
<td>VOMSName = lhcb</td>
</tr>
<tr>
<td>&lt;VO_NAME&gt;/SubmitPools</td>
<td>Default Submit Pools for the users belonging to the VO</td>
<td>SubmitPools = lhcbPool</td>
</tr>
</tbody>
</table>

VOMSServers subsection This subsection of the VO/<VO_NAME> section contains parameters of all the VOMS servers that can be used with the given <VO_NAME>. It has a subsection per each VOMS server ( <VOMS_SERVER> ), the name of the section is the host name of the VOMS server. These parameters are used in order to create appropriate vomses and vomsdir directories when installing DIRAC clients.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;VOMS_SERVER&gt;/Port</td>
<td>Port of the VOMS server port</td>
<td>Port = 15003</td>
</tr>
<tr>
<td>&lt;VOMS_SERVER&gt;/CA</td>
<td>CA that issued the VOMS server certificate</td>
<td>CA = /C=FR/O=CNRS/CN=GRID2-FR</td>
</tr>
</tbody>
</table>

VOMSServices subsection This subsection contains URLs to obtain specific VOMS informations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOMSAttributes</td>
<td>URL to get VOMS attributes</td>
<td>VOMSAttributes = <a href="https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAttributes">https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAttributes</a></td>
</tr>
<tr>
<td>VOMSAdmin</td>
<td>URL to get VOMS administrator info</td>
<td>VOMSAdmin = <a href="https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAdmin">https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAdmin</a></td>
</tr>
<tr>
<td>VOMSCompatibility</td>
<td>URL to get VOMS compatibility info</td>
<td>VOMSCompatibility = <a href="https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCompatibility">https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCompatibility</a></td>
</tr>
<tr>
<td>VOMSCertificates</td>
<td>URL to get VOMS certificate info</td>
<td>VOMSCertificates = <a href="https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCertificates">https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCertificates</a></td>
</tr>
</tbody>
</table>
Resources - Section

In this section all the physical resources than can be used by DIRAC users are described.

Resources / FileCatalogs - Subsections

This subsection include the definition of the File Catalogs to be used in the installation. In case there is more than one File Catalog defined in this section, the first one in the section will be used as default by the ReplicaManager client.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileCatalog</td>
<td>Subsection used to configure DIRAC File catalog</td>
<td>FileCatalog</td>
</tr>
<tr>
<td>FileCatalog/AccessType</td>
<td>Access type allowed to the particular catalog</td>
<td>AccessType = Read-Write</td>
</tr>
<tr>
<td>FileCatalog/Status</td>
<td>To define the catalog as active or inactive</td>
<td>Status = Active</td>
</tr>
<tr>
<td>FileCatalog/MetaCatalog</td>
<td>If the Catalog is a MetaDataCatalog</td>
<td>MetaCatalog = True</td>
</tr>
</tbody>
</table>

Resources / Sites - Subsections

In this section each DIRAC site available for the users is described. The convention to name the sites consist of 3 strings: - Grid site name, expressed in uppercase, for example: LCG, EELA - Institution acronym in uppercase, for example: CPPM - Country: country where the site is located, expressed in lowercase, for example fr

The three strings are concatenated with "." to produce the name of the sites.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;</td>
<td>Subsection named with the site name</td>
<td>LCG.CPPM.fr</td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/Name</td>
<td>Site name gave by the site administrator</td>
<td>NAME = in2p3</td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/CE</td>
<td>List of CEs using CE FQN</td>
<td>CE = ce01.in2p3.fr CE += ce02.in2p3.fr</td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/CEs</td>
<td>Subsection used to describe each CE available</td>
<td>CEs</td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/Coordinates</td>
<td>Site geographical coordinates</td>
<td>Coordinates = -8.637979:41.152461</td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/Mail</td>
<td>Mail address site responsable</td>
<td>Mail = <a href="mailto:atsareg@in2p3.fr">atsareg@in2p3.fr</a></td>
</tr>
<tr>
<td>&lt;DIRAC_SITE_NAME&gt;/SE</td>
<td>Closest SE respect to the CE</td>
<td>SE = se01.in2p3.fr</td>
</tr>
</tbody>
</table>

CEs sub-subsection  This sub-subsection specify the attributes of each particular CE of the site. Must be noticed than in each DIRAC site can be more than one CE.
## Resources / StorageElements - Subsections

All the storages elements available for the users are described in this subsection. This information will be moved bellow the Sites section.
## Resources / StorageElementGroups - Subsections

All the storages elements groups available for the users are described in this subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE-USER</td>
<td>Default SEs to be used when uploading output data from Payloads</td>
<td>CERN-USER</td>
</tr>
</tbody>
</table>

## Systems configuration

Each DIRAC system has its corresponding section in the Configuration namespace.

## Accounting System configuration

In this subsection are described the databases, services and URLs related with Accounting framework for each setup.
Databases used by Accounting System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;DATABASE_NAME&gt;</code></td>
<td>Subsection. Database name</td>
<td>AccountingDB</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/DBName</code></td>
<td>Database name</td>
<td>DBName = AccountingDB</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/Host</code></td>
<td>Database host server where the DB is located</td>
<td>Host = db01.in2p3.fr</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/MaxQueueSize</code></td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated with Accounting System are: - AccountingDB

All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log verbosity</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port used by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to comunicate with the service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization = all</td>
</tr>
</tbody>
</table>

Accounting system related services are:

DataStore service is in charge of receiving Accounting data.

No special options must be configured to use this service.

ReportGenerator service is in charge of producing accounting reports (plots or CSV files).

No special options must be configured.

Accounting Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;SERVICE_NAME&gt;</code></td>
<td>URL associated with the service, value URL using dips protocol</td>
<td>DataStore = dips://dirac.eela.if.ufrj.br:9133/Accounting/DataStore</td>
</tr>
</tbody>
</table>

Services associated with Accounting System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStore</td>
<td>9133</td>
</tr>
<tr>
<td>ReportGenerator</td>
<td>9134</td>
</tr>
</tbody>
</table>
Configuration System configuration

In this subsection are described the databases, services and URLs related with Accounting framework for each setup.

Systems / Configuration / <INSTANCE> / Service - Sub-subsection  In this subsection all the services of Configuration system are described.

Systems / Configuration / <INSTANCE> / Service / Server - Sub-subsection  In this subsection the Server service is configured. The attributes are showed in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandlerPath</td>
<td>Relative path directory where the service is located</td>
<td>HandlerPath = DIRAC/ConfigurationSystem/Service/ConfigurationHandler.py</td>
</tr>
<tr>
<td>Port</td>
<td>Port where the service is responding</td>
<td>Port = 9135</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Subsection to configure authorization over the service Default authorization</td>
<td>Authorization = all</td>
</tr>
<tr>
<td>Authorization/commitNewData</td>
<td>Define who can commit new configuration</td>
<td>commitNewData = CSAdministrator</td>
</tr>
<tr>
<td>Authorization/getVersionContents</td>
<td>Define who can get version contents</td>
<td>getVersionContents = CSAdministrator</td>
</tr>
<tr>
<td>Authorization/rollbackToVersion</td>
<td>Define who can roll back the configuration to a previous version</td>
<td>rollBackToVersion = ServiceAdministrator rollBackToVersion += CSAdministrator</td>
</tr>
</tbody>
</table>

Systems / Configuration / <INSTANCE> / Agents - Sub-subsection

In this subsection each agent is described.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Subsection named as the agent is called.</td>
<td>CE2CSAgent</td>
</tr>
</tbody>
</table>

Common options for all the agents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Log Level associated to the agent</td>
<td>LogLevel = DEBUG</td>
</tr>
<tr>
<td>LogBackends</td>
<td></td>
<td>LogBackends = stdout, server</td>
</tr>
<tr>
<td>MaxCycles</td>
<td>Maximum number of cycles made for Agent</td>
<td>MaxCycles = 500</td>
</tr>
<tr>
<td>MonitoringEnabled</td>
<td>Indicates if the monitoring of agent is enabled. Boolean values</td>
<td>MonitoringEnabled = True</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Each many time a new cycle must start expresed in seconds</td>
<td>PollingTime = 2600</td>
</tr>
<tr>
<td>Status</td>
<td>Agent Status, possible values Active or Inactive</td>
<td>Status = Active</td>
</tr>
</tbody>
</table>

Agents associated with Configuration System:

Systems / Configuration / <INSTANCE> / Agents /CE2CSAgent - Sub-subsection  CE2CSAgent is the agent in charge of update sites parameters configuration for a specific VO. - Queries BDII for unknown CE. - Queries BDII for CE information and put it to CS.

The attributes of this agent are shown in the table below:
**AlternativeBDIIs**

List of alternatives BDIs

**Example**

AlternativeBDIIs = bdii01.in2p3.fr

**BannedCEs**

Banned Computing Elements

**Example**

BannedCEs = []

**MailTo**

E-mail of the person in charge of update the Sites configuration

**Example**

MailTo = hamar@cppm.in2p3.fr

**MailFrom**

E-mail address used to send the information to be updated

**Example**

MailFrom = dirac@mardirac.in2p3.fr

**VirtualOrganization**

Name of the VO

**Example**

VirtualOrganization = vo.formation.idgrilles.fr

**Systems / Configuration / <INSTANCE> / Agents / UsersAndGroups - Sub-subsection**

Queries VOMRS servers and updates the users and groups as defined in the Registry/VOMS/Mapping section.

The attributes of this agent are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailTo</td>
<td>E-mail of the person in charge of update the Sites configuration</td>
<td>MailTo = <a href="mailto:hamar@cppm.in2p3.fr">hamar@cppm.in2p3.fr</a></td>
</tr>
<tr>
<td>MailFrom</td>
<td>E-mail address used to send the information to be updated</td>
<td>MailFrom = <a href="mailto:dirac@mardirac.in2p3.fr">dirac@mardirac.in2p3.fr</a></td>
</tr>
<tr>
<td>LFCCheckEnabled</td>
<td>Boolean, check users are properly defined in LFC</td>
<td>LFCCheckEnabled = True</td>
</tr>
</tbody>
</table>

**Systems / Configuration / <INSTANCE> / URLs - Sub-subsection**

Configuration Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, the value is a URL using dips protocol</td>
<td>dips://guaivira.lsd.ufcg.edu.br:9135/Configuration/Server</td>
</tr>
</tbody>
</table>

Services associated with Configuration System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>9135</td>
</tr>
</tbody>
</table>

**DataManagement System configuration**

In this subsection are described the databases, services and URLs related with the DataManagement system for each setup.

**Systems / DataManagement / <INSTANCE> / Databases - Sub-subsection**

Databases used by DataManagement System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DATABASE_NAME&gt;</td>
<td>Subsection. Database name</td>
<td>FileCatalogDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/DBName</td>
<td>Database name</td>
<td>DBName = FileCatalogDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/Host</td>
<td>Database host server where the DB is located</td>
<td>Host = db01.in2p3.fr</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/MaxQueueSize</td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated with DataManagement System are: - FileCatalogDB - DataIntegrityDB - DataLoggingDB
All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port used by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to communicate with the service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Define to who is required the authorization</td>
<td>Default = all</td>
</tr>
</tbody>
</table>

DataManagement services are:

FileCatalogHandler is a simple Replica and Metadata Catalog service. Special options are required to configure this service, showed in the next table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultUmask</td>
<td>Default UMASK</td>
<td>DefaultUmask = 509</td>
</tr>
<tr>
<td>DirectoryManager</td>
<td>Directory manager</td>
<td>DirectoryManager = DirectoryLevelTree</td>
</tr>
<tr>
<td>FileManager</td>
<td>File Manager</td>
<td>FileManager = FileManager</td>
</tr>
<tr>
<td>GlobalReadAcess</td>
<td>Boolean Global Read Access</td>
<td>GlobalReadAccess = True</td>
</tr>
<tr>
<td>LFNPFNConvention</td>
<td>Boolean indicating to use LFN PFN convention</td>
<td>LFNPFNConvention = True</td>
</tr>
<tr>
<td>SecurityManager</td>
<td>Security manager to be used</td>
<td>SecurityManager = NoSecurityManager</td>
</tr>
<tr>
<td>SEManager</td>
<td>Storage Element manager</td>
<td>SEManager = SEManagerDB</td>
</tr>
<tr>
<td>ResolvePFN</td>
<td>Boolean indicating if resolve PFN must be done</td>
<td>ResolvePFN = True</td>
</tr>
<tr>
<td>VisibleStatus</td>
<td>Visible Status</td>
<td>VisibleStatus = AprioriGood</td>
</tr>
<tr>
<td>UniqueGUID</td>
<td>Use a unique GUID</td>
<td>UniqueGUID = False</td>
</tr>
<tr>
<td>UserGroupManager</td>
<td>User group manager</td>
<td>UserGroupManager = UserAndGroupManagerDB</td>
</tr>
</tbody>
</table>

StorageElementHandler is the implementation of a simple StorageElement service in the DISET framework

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasePath</td>
<td>Directory path used as base for DIRAC SE</td>
<td>BasePath = /opt/dirac/data</td>
</tr>
</tbody>
</table>

This is a service which represents a DISET proxy to the Storage Element component. This is used to get and put files from a remote storage.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasePath</td>
<td>Temporary directory use for transfers</td>
<td>BasePath = storageElement</td>
</tr>
</tbody>
</table>
This is a service which exposes TransferDB methods via a DISET interface.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example and default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestsPerCycle</td>
<td>Number of requests executed in one agent’s cycle</td>
<td>RequestsPerCycle = 10</td>
</tr>
<tr>
<td>MinProcess</td>
<td>Minimal number of sub-processes running.</td>
<td>MinProcess = 1</td>
</tr>
<tr>
<td>MaxProcess</td>
<td>Maximal number of sub-processes running.</td>
<td>MaxProcess = 4</td>
</tr>
<tr>
<td>RequestType</td>
<td>Request type.</td>
<td>RequestType = transfer</td>
</tr>
<tr>
<td>shifterProxy</td>
<td>Proxy to use.</td>
<td>shifterProxy = DataManager</td>
</tr>
<tr>
<td>TaskMode</td>
<td>Flag to enable/disable tasks execution.</td>
<td>TaskMode = True</td>
</tr>
<tr>
<td>FTSMode</td>
<td>Flag to enable/disable FTS scheduling.</td>
<td>FTSMode = False</td>
</tr>
<tr>
<td>ThroughputTimescale</td>
<td>Monitoring time period of the FTS processing used for scheduling (in seconds).</td>
<td>ThroughputTimescale = 3600</td>
</tr>
<tr>
<td>HopSigma</td>
<td>Acceptable time shift to start of FTS transfer.</td>
<td>HopSigma = 0.0</td>
</tr>
<tr>
<td>SchedulingType</td>
<td>Choose transfer speed between number of files per hour or amount of transferred data per hour.</td>
<td>SchedulingType = Files</td>
</tr>
<tr>
<td>ActiveStrategies</td>
<td>List of active startegies to use.</td>
<td>ActiveStrategies = MinimiseTotalWait</td>
</tr>
<tr>
<td>AcceptableFailureRate</td>
<td>Percentage limit of success rate in monitored FTS transfers to accept/reject FTS channel.</td>
<td>AcceptableFailureRate = 75</td>
</tr>
</tbody>
</table>

By default TransferAgent is running in both modes (task execution and FTS scheduling), but for online processing FTSMode should be disabled.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, value URL using dips protocol</td>
<td>dips://dirac.eela.if.ufrj.br:9197/DataManagement/FileCatalog</td>
</tr>
</tbody>
</table>

Services associated with DataManagement System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileCatalog</td>
<td>9197</td>
</tr>
<tr>
<td>StorageElement</td>
<td>9148</td>
</tr>
<tr>
<td>StorageElementProxy</td>
<td>9139</td>
</tr>
<tr>
<td>TransferDBMonitoring</td>
<td>9191</td>
</tr>
</tbody>
</table>
WorkloadManagement System configuration

In this subsection are described the databases, services and URLs related with WorkloadManagement System for each setup.

Systems / WorkloadManagement / <INSTANCE> / Databases - Sub-subsection  Databases used by WorkloadManagement System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DATABASE_NAME&gt;</td>
<td>Subsection. Database name</td>
<td>JobDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/DBName</td>
<td>Database name</td>
<td>DBName = JobDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/Host</td>
<td>Database host server where the DB is located</td>
<td>Host = db01.in2p3.fr</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/MaxQueueSize</td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated to WorkloadManagement System are: - JobDB - JobLoggingDB - MPIJobDB - PilotAgentDB - SandboxMetadataDB - TaskQueueDB

Systems / WorkloadManagement / <INSTANCE> / Services - Sub-subsection  All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log verbosity</td>
<td>LogLevel = INFO</td>
</tr>
</tbody>
</table>
| LogBackends         | Log backends                                    | LogBackends = stdout  
|                     |                                                  | LogBackends += server |
| MaskRequestParameters| Request to mask the values, possible values: yes or no | MaskRequestParameters = yes |
| MaxThreads          | Maximum number of threads used in parallel for the server | MaxThreads = 50 |
| Port                | Port useb by DIRAC service                      | Port = 9140 |
| Protocol            | Protocol used to comunicate with the service     | Protocol = dips   |
| Authorization       | Subsection used to define which kind of Authorization is required to talk with the service | Authorization |
| Authorization/Default| Define to who is required the authorization      | Default = all    |

WorkloadManagement services are:

Systems / WorkloadManagement / <INSTANCE> / Service / JobManager - Sub-subsection  JobManagerHandler is the implementation of the JobManager service in the DISET framework

No special options required to configure this service.

Systems / WorkloadManagement / <INSTANCE> / Service / JobMonitoring - Sub-subsection  JobMonitoring-Handler is the implementation of the JobMonitoring service in the DISET framework

No special options required to configure this service.

Systems / WorkloadManagement / <INSTANCE> / Service / JobStateUpdate - Sub-subsection  JobStateUpdate-Handler is the implementation of the Job State updating service in the DISET framework

Special option for the service configuration are showed in the next table:
**Systems / WorkloadManagement / <INSTANCE> / Service / Matcher - Sub-subsection**  
Matcher class. It matches Agent Site capabilities to job requirements. It also provides an XMLRPC interface to the Matcher

A special authorization needs to be added:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>setActiveTaskQueues</td>
<td>Define DIRAC group allowed to get the active task queues in the system</td>
<td>setActiveTaskQueues = dirac_admin</td>
</tr>
</tbody>
</table>

**Systems / WorkloadManagement / <INSTANCE> / Service / SandboxStore - Sub-section**  
SandboxHandler is the implementation of the Sandbox service in the DISET framework

Some extra options are required to configure this service:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backend</td>
<td>Backend = local</td>
<td></td>
</tr>
<tr>
<td>BasePath</td>
<td>Base path where the files are stored task queues in the system</td>
<td>BasePath = /opt/dirac/storage/sandboxes</td>
</tr>
<tr>
<td>DelayedExternalDeletion</td>
<td>Boolean used to define if the external deletion must be done</td>
<td>DelayedExternalDeletion = True</td>
</tr>
<tr>
<td>MaxSandboxSize</td>
<td>Maximum size of sanbox files expressed in MB</td>
<td>MaxSandboxSize = 10</td>
</tr>
<tr>
<td>SandboxPrefix</td>
<td>Path prefix where sandbox are stored</td>
<td>SandboxPrefix = Sandbox</td>
</tr>
</tbody>
</table>

**Systems / WorkloadManagement / <INSTANCE> / Service / WMSAdministrator - Sub-subsection**  
This is a DIRAC WMS administrator interface.

No extra options are required to configure this service.

**Systems / WorkloadManagement / <INSTANCE> / Agents - Sub-section**

In this subsection each agent is described.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Subsection named as the agent is called.</td>
<td>InputDataAgent</td>
</tr>
</tbody>
</table>

Common options for all the agents are described in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Log Level associated to the agent</td>
<td>LogLevel = DEBUG</td>
</tr>
<tr>
<td>LogBackends</td>
<td></td>
<td>LogBackends = stdout, server</td>
</tr>
<tr>
<td>MaxCycles</td>
<td>Maximum number of cycles made for Agent</td>
<td>MaxCycles = 500</td>
</tr>
<tr>
<td>MonitoringEnabled</td>
<td>Indicates if the monitoring of agent is enabled. Boolean values</td>
<td>MonitoringEnabled = True</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Each many time a new cycle must start expressed in seconds</td>
<td>PollingTime = 2600</td>
</tr>
<tr>
<td>Status</td>
<td>Agent Status, possible values Active or Inactive</td>
<td>Status = Active</td>
</tr>
</tbody>
</table>

Agents associated with Configuration System:
The Job Cleaning Agent controls removing jobs from the WMS in the end of their life cycle. The attributes are showed in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job-ByJob</td>
<td>Boolean than express if job by job must be processed</td>
<td>JobByJob = True</td>
</tr>
<tr>
<td>MaxJobsAtOnce</td>
<td>Maximum number of jobs to be processed at the same time</td>
<td>MaxJobsAtOnce = 200</td>
</tr>
<tr>
<td>ProductionTypes</td>
<td>Production types</td>
<td>ProductionTypes = DataReconstruction ProductionTypes += DataStripping ProductionTypes += MCSimulation ProductionTypes += Merge ProductionTypes += production</td>
</tr>
<tr>
<td>ThrottlingPeriod</td>
<td></td>
<td>ThrottlingPeriod = 0</td>
</tr>
</tbody>
</table>

And also the common options for all the agents.

JobHistoryAgent sends periodically numbers of jobs in various states for various sites to the Monitoring system to create historical plots.

Special attributes for this agent are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdatedPeriod</td>
<td>Each many time must update the information Expressed in seconds</td>
<td>UpdatedPeriod = 300</td>
</tr>
</tbody>
</table>

The Pilot Status Agent updates the status of the pilot jobs if the PilotAgents database.

Special attributes for this agent are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridEnv</td>
<td>Path where is located the file to load Grid Environment Variables</td>
<td>GridEnv = /usr/profile.d/grid-env</td>
</tr>
<tr>
<td>PilotAccountingEnabled</td>
<td>Boolean type attribute than allows to specify if accounting is enabled</td>
<td>PilotAccountingEnabled = Yes</td>
</tr>
<tr>
<td>PilotStalledDays</td>
<td>Number of days without response of a pilot before be declared as Stalled</td>
<td>PilotStalledDays = 3</td>
</tr>
</tbody>
</table>

Site director is in charge of submit pilot jobs to special Computing Elements.

Special attributes for this agent are (updated for v6r15):
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CETypes</td>
<td>List of CEs types allowed to submit pilot jobs</td>
<td>CETypes = CREAM</td>
</tr>
<tr>
<td>CEs</td>
<td>List of CEs to submit pilot jobs</td>
<td>CEs = ce01.in2p3.fr</td>
</tr>
<tr>
<td>Site</td>
<td>Sites name list where the pilots will be submitted</td>
<td>Site =</td>
</tr>
<tr>
<td>PilotDN</td>
<td>Pilot DN used to submit the pilot jobs</td>
<td>PilotDN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar</td>
</tr>
<tr>
<td>PilotGroup</td>
<td>DIRAC group used to submit the pilot jobs</td>
<td>PilotGroup = dirac_pilot</td>
</tr>
<tr>
<td>GetPilotOutput</td>
<td>Boolean value used to indicate the pilot output will be or not retrieved</td>
<td>GetPilotOutput = True</td>
</tr>
<tr>
<td>GridEnv</td>
<td>Path where is located the file to load Grid Environment Variables</td>
<td>GridEnv = /usr/profile.d/grid-env</td>
</tr>
<tr>
<td>MaxQueueLength</td>
<td>Maximum cputime used for a queue, will set maxCPU time to this value</td>
<td>MaxQueueLength = 86400*3</td>
</tr>
<tr>
<td>SendPilotAccounting</td>
<td>Boolean value than indicates if the pilot job will send information for accounting</td>
<td>SendPilotAccounting = yes</td>
</tr>
<tr>
<td>UpdatePilotStatus</td>
<td>Attribute used to define if the status of the pilot will be updated</td>
<td>UpdatePilotStatus = True</td>
</tr>
<tr>
<td>VO or Community</td>
<td>Optional, will be obtained by other means if not set</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>Which group is allowed to use these pilots</td>
<td></td>
</tr>
<tr>
<td>PilotLogLevel</td>
<td>LogLevel of the pilot</td>
<td>PilotLogLevel = DEBUG</td>
</tr>
<tr>
<td>MaxJobsInFillMode</td>
<td>How many jobs the pilot can run</td>
<td>MaxJobsInFillMode=5</td>
</tr>
<tr>
<td>MaxPilotsToSubmit</td>
<td>How many pilots to submit per cycle</td>
<td></td>
</tr>
<tr>
<td>PilotWaitingFlag</td>
<td>Boolean to limit the number of waiting pilots</td>
<td>PilotWaitingFlag = False</td>
</tr>
<tr>
<td>MaxPilotWaitingTime</td>
<td>How old pilots can be to count them as a waiting pilot???</td>
<td></td>
</tr>
<tr>
<td>FailedQueueCycleFactor</td>
<td>How many cylices to skip if queue was not working</td>
<td></td>
</tr>
<tr>
<td>AddPilotsToEmptySites</td>
<td>To submit pilots to empty sites in any case (False by default)</td>
<td>AddPilotsToEmptySites = True</td>
</tr>
<tr>
<td>PilotStatusUpdateCycleFactor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExtraPilotModules</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Systems / WorkloadManagement / <INSTANCE> / Agents / StalledJobAgent - Sub-subsection**  
The StalledJobAgent hunts for stalled jobs in the Job database. Jobs in “running” state not receiving a heart beat signal for more than stalledTime seconds will be assigned the “Stalled” state.

The FailedTimeHours and StalledTimeHours are actually given in number of cycles. One Cycle is 30 minutes and can be changed in the Systems/WorkloadManagement/<Instance>/JobWrapper section with the CheckingTime and MinCheckingTime options.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailedTime-Hours</td>
<td>How much time in hours pass before a stalled job is declared as failed Note: Not actually in hours</td>
<td>FailedTimeHours = 6</td>
</tr>
<tr>
<td>StalledTime-Hours</td>
<td>How much time in hours pass before running job is declared as stalled Note: Not actually in hours</td>
<td>StalledTimeHours = 2</td>
</tr>
<tr>
<td>Matched-Time</td>
<td>Age in seconds until matched jobs are rescheduled</td>
<td>MatchedTime = 7200</td>
</tr>
<tr>
<td>RescheduledTime</td>
<td>Age in seconds until rescheduled jobs are rescheduled</td>
<td>RescheduledTime = 600</td>
</tr>
<tr>
<td>Completed-Time</td>
<td>Age in seconds until completed jobs are declared failed, unless their minor status is “Pending Requests”</td>
<td>CompletedTime = 86400</td>
</tr>
</tbody>
</table>

**Systems / WorkloadManagement / <INSTANCE> / Agents / StatesAccountingAgent - Sub-subsection**

StateAccountingAgent sends periodically numbers of jobs in various states for various sites to the Monitoring system to create historical plots.

This agent doesn’t have special options to configure.

**Systems / WorkloadManagement / <INSTANCE> / Agents / TaskQueueDirector - Sub-subsection**

The TaskQueue Director Agent controls the submission of pilots via the PilotDirectors. These are Backend-specific PilotDirector derived classes. This is a simple wrapper that performs the instantiation and monitoring of the PilotDirector instances and add workload to them via ThreadPool mechanism.

### From the base Agent class it uses the following configuration Parameters

- WorkDir:
- PollingTime:
- MaxCycles:

The following parameters are searched for in WorkloadManagement/TaskQueueDirector:

- ThreadStartDelay:
- SubmitPools: All the Submit pools that are to be initialized
- DefaultSubmitPools: If no specific pool is requested, use these

### It will use those Directors to submit pilots for each of the Supported SubmitPools

- SubmitPools (see above)

### SubmitPools may refer to:

- a full GRID infrastructure (like EGEE, OSG, NDG,...) access remotely through RBs or WMSs servers distributing the load over all available resources (CEs) using ad hoc middleware (gLite, LCG, ...).
- individual GRID Computing Elements again access remotely through their corresponding GRID interface using ad hoc middleware.
- classic batch systems (like LSF, BQS, PBS, Torque, Condor, ...) access locally trough their corresponding head nodes using their own specific tools
- standalone computers access by direct execution (fork or exec)

In first two cases, the middleware takes care of properly handling the secure transfer of the payload to the executing node. In the last two DIRAC will take care of all relevant security aspects.

For every SubmitPool category (GRID or DIRAC) and there must be a corresponding Section with the necessary parameters:
• Pool: if a dedicated Threadpool is desired for this SubmitPool

GRID:
• GridMiddleware: <GridMiddleware>PilotDirector module from the PilotAgent directory will be used, currently LCG, gLite types are supported

For every supported GridMiddleware there must be a corresponding Section with the necessary parameters:

• gLite:
• LCG:
• DIRAC:

For every supported “Local backend” there must be a corresponding Section with the necessary parameters:

• PBS:
• Torque:
• LSF:
• BQS:
• Condor:

(This are the parameters referring to the corresponding SubmitPool and PilotDirector classes, not the ones referring to the CE object that does the actual submission to the backend)

The following parameters are taken from the TaskQueueDirector section if not present in the corresponding SubmitPool section:

• GenericPilotDN:
• GenericPilotGroup:

The pilot submission logic is as follows:

• Determine prioritySum: sum of the Priorities for all TaskQueues in the system.
• Determine pilotsPerPriority: result of dividing the number of pilots to submit per iteration by the prioritySum.
• select TaskQueues from the WMS system appropriated for PilotSubmission by the supported SubmitPools
• For each TaskQueue determine a target number of pilots to submit:
  – Multiply the priority by pilotsPerPriority.
  – Apply a correction factor for proportional to maxCPU divided by CPU of the TaskQueue ( double number of pilots will be submitted for a TaskQueue with half CPU required ). To apply this correction the minimum CPU considered is lowestCPUBoost.
  – Apply poisson statistics to determine the target number of pilots to submit (even a TQ with a very small priorities will get a chance of getting pilots submitted).
  – Determine a maximum number of “Waiting” pilots in the system: ( 1 + extraPilotFraction ) * [No. of Jobs in TaskQueue] + extraPilots
  – Attempt to submit as many pilots a the minimum between both number.
  – Pilot submission request is inserted into a ThreadPool.
• Report the sum of the Target number of pilots to be submitted.
• Wait until the ThreadPool is empty.
• Report the actual number of pilots submitted.

2.6. DIRAC Configuration
In summary:

All TaskQueues are considered on every iteration, pilots are submitted statistically proportional to the priority and the Number of waiting tasks of the TaskQueue, boosted for the TaskQueues with lower CPU requirements and limited by the difference between the number of waiting jobs and the number of already waiting pilots.

This module is prepared to work:

- locally to the WMS DIRAC server and connect directly to the necessary DBs.
- remotely to the WMS DIRAC server and connect via appropriated DISET methods.

Obsolete Job JDL Option:

GridExecutable SoftwareTag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowedSubmitPools</td>
<td>Pools where is possible to submit pilot jobs</td>
<td>AllowedSubmitPools = gLite AllowedSubmitPools += DIRAC</td>
</tr>
<tr>
<td>DefaultSubmitPools</td>
<td>Default submit pilot pools</td>
<td>DefaultSubmitPools = DIRAC</td>
</tr>
<tr>
<td>extraPilots</td>
<td>Number of extra pilot jobs to be submitted</td>
<td>extraPilots = 4</td>
</tr>
<tr>
<td>extraPilotFraction</td>
<td>Percentage of private pilots fraction to be submitted</td>
<td>extraPilotFraction = 0.2</td>
</tr>
<tr>
<td>ExtraPilotOptions</td>
<td>Extra configuration options to be added during pilot jobs are executed</td>
<td>ExtraPilotOptions = -g 2010-11-20</td>
</tr>
<tr>
<td>GridMiddleware</td>
<td>Pool Grid middleware</td>
<td>GridMiddleware = gLite</td>
</tr>
<tr>
<td>ListMatchDelay</td>
<td></td>
<td>ListMatchDelay =</td>
</tr>
<tr>
<td>lowestCPUBoost</td>
<td></td>
<td>lowestCPUBoost = 7200</td>
</tr>
<tr>
<td>maxPilotWaitingHours</td>
<td>Maximum number hours of pilots in waiting status</td>
<td>maxPilotWaitingHours = 6</td>
</tr>
<tr>
<td>maxThreadsInPool</td>
<td>Maximum number of threads by pool</td>
<td>maxThreadsInPool = 2</td>
</tr>
<tr>
<td>minThreadsInPool</td>
<td>Minimum number of threads by pool</td>
<td>minThreadsInPool = 0</td>
</tr>
<tr>
<td>pilotsPerIteration</td>
<td>Number of pilots by iteration</td>
<td>pilotsPerIteration = 40</td>
</tr>
<tr>
<td>PilotScript</td>
<td>Path in DIRAC server where the pilot script is located</td>
<td>PilotScript = /opt/dirac/pro/DIRAC/WorkloadManagementSystem/PilotAgent/dirac-pilot.py</td>
</tr>
<tr>
<td>SubmitPools</td>
<td>Pools where is possible to submit pilot jobs</td>
<td>SubmitPools = gLite</td>
</tr>
<tr>
<td>ThreadStartDelay</td>
<td>ThreadStartDelay</td>
<td>ThreadStartDelay = 0</td>
</tr>
<tr>
<td>totalThreadsInPool</td>
<td>Total number of threads for each pool</td>
<td>totalThreadsInPool = 40</td>
</tr>
</tbody>
</table>

Submission pools:
Options available to configure gLite pool submission are showed in a table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failing</td>
<td>Distinguish name to be used to submit the pilot jobs</td>
<td>GenericPilotDN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar</td>
</tr>
<tr>
<td>GenericPilotDN</td>
<td>DIRAC group used to submit the pilot jobs</td>
<td>GenericPilotGroup = dirac_pilot</td>
</tr>
<tr>
<td>GridMiddleware</td>
<td>Pool submission of Grid middleware</td>
<td>GridMiddleware = gLite</td>
</tr>
<tr>
<td>LoggingServers</td>
<td>Loggin servers available for the pool</td>
<td>LoggingServers = lb01.in2p3.fr</td>
</tr>
<tr>
<td>MaxJobsinFillMode</td>
<td>Maximum number of jobs to run by a pilot job</td>
<td>MaxJobsinFillMode = 5</td>
</tr>
<tr>
<td>PrivatePilotFraction</td>
<td>Portion of private pilots to be submitted expressed in a value between 0 and 1</td>
<td>PrivatePilotFraction = 0.5</td>
</tr>
<tr>
<td>Rank</td>
<td>Rank in gLite format</td>
<td>Rank = ( other.GlueCEStateWaitingJobs == 0 ? ( other.GlueCEStateFreeCPUs * 10 / other.GlueCEInfoTotalCPUs + other.GlueCEInfoTotalCPUs / 500 ) : -other.GlueCEStateWaitingJobs * 4 / (other.GlueCEStateRunningJobs + 1 ) - 1 )</td>
</tr>
</tbody>
</table>

Options available to configure DIRAC pool submission are showed in a table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridMiddleware</td>
<td>Pool submission grid middleware</td>
<td>GridMiddleware = DIRAC</td>
</tr>
</tbody>
</table>

In this subsection each executor is described.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executor</td>
<td>Subsection named as the Executor is called.</td>
<td>InputData</td>
</tr>
</tbody>
</table>

Common options for all the executors are described in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Log Level associated to the executor</td>
<td>LogLevel = DEBUG</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log Level associated to the executor</td>
<td>LogBackends = stdout, server</td>
</tr>
<tr>
<td>Status</td>
<td>Executor Status, possible values Active or Inactive</td>
<td>Status = Active</td>
</tr>
</tbody>
</table>

Executors associated with Configuration System:
The Input Data Executor queries the file catalog for specified job input data and adds the relevant information to the job optimizer parameters to be used during the scheduling decision.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailedJobStatus</td>
<td>MinorStatus if Executor fails the job</td>
<td>FailedJobStatus = “Input Data Not Available”</td>
</tr>
<tr>
<td>CheckFileMetadata</td>
<td>Boolean, check file metadata; will ignore Failover SE files</td>
<td>CheckFileMetadata = True</td>
</tr>
</tbody>
</table>

The Job Path Agent determines the chain of Optimizing Agents that must work on the job prior to the scheduling decision. Initially this takes jobs in the received state and starts the jobs on the optimizer chain. The next development will be to explicitly specify the path through the optimizers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasePath</td>
<td>Path for jobs through the executors</td>
<td>BasePath = JobPath, JobSanity</td>
</tr>
<tr>
<td>VOPlugin</td>
<td>Name of a VO Plugin??</td>
<td>VOPlugin = ””</td>
</tr>
<tr>
<td>InputData</td>
<td>Name of the InputData instance</td>
<td>InputData = InputData</td>
</tr>
<tr>
<td>EndPath</td>
<td>Last executor for a job</td>
<td>EndPath = JobScheduling</td>
</tr>
</tbody>
</table>

The JobSanity executor screens jobs for the following problems

- Problematic JDL
- Jobs with too much input data e.g. > 100 files
- Jobs with input data incorrectly specified e.g. castor:/
- Input sandbox not correctly uploaded.
- Output data already exists (not implemented)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputDataCheck</td>
<td>Boolean, check if input data is properly formatted, default=True</td>
<td>InputDataCheck = True</td>
</tr>
<tr>
<td>MaxInputDataPerJob</td>
<td>Integer, Maximum number of input lfns</td>
<td>MaxInputDataPerJob=100</td>
</tr>
<tr>
<td>InputSandboxCheck</td>
<td>Check for input sandbox files</td>
<td>InputSandboxCheck = True</td>
</tr>
<tr>
<td>PlatformCheck</td>
<td>Check if platform is supported Not Implemented</td>
<td>PlatformCheck = False</td>
</tr>
<tr>
<td>OutputDataCheck</td>
<td>Check if output data exists Not Implemented</td>
<td>OutputDataCheck = True</td>
</tr>
</tbody>
</table>

The Job Scheduling Executor takes the information gained from all previous optimizers and makes a scheduling decision for the jobs. Subsequent to this jobs are added into a Task Queue and pilot agents can be submitted. All issues preventing the successful resolution of a site candidate are discovered here where all information is available. This Executor will fail affected jobs meaningfully.
### Systems / WorkloadManagement / <INSTANCE> / JobWrapper - Sub-subsection

The Job Wrapper Class is instantiated with arguments tailored for running a particular job. The JobWrapper starts a thread for execution of the job and a Watchdog Agent that can monitor progress.

The options used to configure JobWrapper are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferLimit</td>
<td>Size limit of the buffer used for transmission between the WN and DIRAC server</td>
<td>BufferLimit = 10485760</td>
</tr>
<tr>
<td>CleanUpFlag</td>
<td>Boolean</td>
<td>CleanUpFlag = True</td>
</tr>
<tr>
<td>DefaultCatalog</td>
<td>Default catalog where must be registered the output files if this is not defined by the user FileCatalog define DIRAC file catalog</td>
<td>DefaultCatalog = FileCatalog</td>
</tr>
<tr>
<td>DefaultCPUTime</td>
<td>Default CPUTime expressed in seconds</td>
<td>DefaultCPUTime = 600</td>
</tr>
<tr>
<td>DefaultErrorFile</td>
<td>Name of default error file</td>
<td>DefaultErrorFile = std.err</td>
</tr>
<tr>
<td>DefaultOutputFile</td>
<td>Name of default output file</td>
<td>DefaultOutputFile = std.out</td>
</tr>
<tr>
<td>DefaultOutputSE</td>
<td>Default output storage element</td>
<td>DefaultOutputSE = IN2P3-disk</td>
</tr>
<tr>
<td>MaxJobPeekLines</td>
<td>Maximum number of output job lines showed</td>
<td>MaxJobPeekLines = 20</td>
</tr>
<tr>
<td>OutputSandboxLimit</td>
<td>Limit of sandbox output expressed in MB</td>
<td>OutputSandboxLimit = 10</td>
</tr>
</tbody>
</table>

### Systems / WorkloadManagement / <INSTANCE> / URLs - Sub-subsection

WorkloadManagement Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, value URL using dips protocol</td>
<td>JobManager = dips://dirac.eela.if.ufrj.br:9132/WorkloadManagement/JobManager</td>
</tr>
</tbody>
</table>

Services associated with WorkloadManagement System:
### Service and Port

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobManager</td>
<td>9132</td>
</tr>
<tr>
<td>JobMonitoring</td>
<td>9130</td>
</tr>
<tr>
<td>JobStateUpdate</td>
<td>9136</td>
</tr>
<tr>
<td>Matcher</td>
<td>9170</td>
</tr>
<tr>
<td>MPIService</td>
<td>9171</td>
</tr>
<tr>
<td>SandboxStore</td>
<td>9196</td>
</tr>
<tr>
<td>WMSAdministrator</td>
<td>9145</td>
</tr>
</tbody>
</table>

### RequestManagement System configuration

In this subsection are described the databases, services and URLs related with RequestManagement System for each setup.

#### Systems / RequestManagement / <INSTANCE> / Databases - Sub-subsection

Databases used by RequestManagement System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;DATABASE_NAME&gt;</code></td>
<td>Subsection. Database name</td>
<td>RequestDB DBName</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/DBName</code></td>
<td>Database host server where the DB is located</td>
<td>= RequestDB Host</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/Host</code></td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>= db01.in2p3.fr</td>
</tr>
<tr>
<td><code>&lt;DATABASE_NAME&gt;/MaxQueueSize</code></td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>= MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated to RequestManagement System are: - RequestDB

#### Systems / RequestManagement / <INSTANCE> / Service - Sub-subsection

All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log verbosity</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port used by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to comunicate with the service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Define to who is required the authorization</td>
<td>Default = all</td>
</tr>
</tbody>
</table>

DataStore services are:

#### Systems / WorkloadManagement / <INSTANCE> / Service / RequestManager - Sub-subsection

RequestManager is the implementation of the RequestDB service in the DISET framework.

Special options to configure this service are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Define the path where the request files are stored</td>
<td>Path = /opt/dirac/requestDB</td>
</tr>
</tbody>
</table>
Systems / RequestManagement / <INSTANCE> / URLs - Sub-subsection  RequestManagement Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, value</td>
<td>RequestManager = dips://dirac.eela.if.ufrj.br:9143/RequestManagement/RequestManager</td>
</tr>
<tr>
<td></td>
<td>URL using dips protocol</td>
<td></td>
</tr>
</tbody>
</table>

Services associated with RequestManagement System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestManager</td>
<td>9143</td>
</tr>
</tbody>
</table>

Framework System configuration

In this subsection are described the databases, services and URLs related with Framework System for each setup.

Systems / Framework / <INSTANCE> / Databases - Sub-subsection  Databases used by DataManagement System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DATABASE_NAME&gt;</td>
<td>Subsection. Database name</td>
<td>ProxyDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/DBName</td>
<td>Database name</td>
<td>DBName</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/Host</td>
<td>Database host server where the DB is located</td>
<td>Host</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/MaxQueueSize</td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated to Framework System are: - ComponentMonitoringDB - NotificationDB - ProxyDB - SystemLoggingDB - UserProfileDB

Systems / Framework / <INSTANCE> / Service - Sub-subsection  All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of logs</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port useb by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to communicate with service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Define to who is required the authorization</td>
<td>Default = all</td>
</tr>
</tbody>
</table>

Services associated with Framework system are:

Systems / Framework / <INSTANCE> / Service / BundleDelivery - Sub-subsection  Bundle delivery services is used to transfer Directories to clients by making tarballs.
**NAME** | **Description** | **Example**
---|---|---
CAs | Boolean, bundle CAs | CAs = True
CRLs | Boolean, bundle CRLs | CRLs = True
DirsToBundle | Section with Additional directories to serve | DirsToBundle/NameA = /opt/dirac/NameA

**Systems / Framework / <INSTANCE> / Service / Monitoring - Sub-subsection**

Monitoring service is in charge of recollect the information necessary to create the plots.

Extra options required to configure the monitoring system are:

**Name** | **Description** | **Example**
---|---|---
DataLocation | Path where data for monitoring is stored | DataLocation = data/Monitoring

**Systems / Framework / <INSTANCE> / Service / Notification - Sub-subsection**

The Notification service provides a toolkit to contact people via email (eventually SMS etc.) to trigger some actions.

The original motivation for this is due to some sites restricting the sending of email but it is useful for e.g. crash reports to get to their destination.

Another use-case is for users to request an email notification for the completion of their jobs. When output data files are uploaded to the Grid, an email could be sent by default with the metadata of the file.

It can also be used to set alarms to be promptly forwarded to those subscribing to them.

Extra options required to configure the Notification system are:

**Name** | **Description** | **Example**
---|---|---
SMSSwitch | SMS switch used to send messages | SMSSwitch = sms.switch.ch

**Systems / Framework / <INSTANCE> / Service / Plotting - Sub-subsection**

Plotting Service generates graphs according to the client specifications and data.

Extra options required to configure plotting system are:

**Name** | **Description** | **Example**
---|---|---
PlotsLocation | Path where data for monitoring is stored | PlotsLocation = data/plots

**Systems / Framework / <INSTANCE> / Service / ProxyManager - Sub-subsection**

ProxyManager is the implementation of the ProxyManagement service in the DISET framework. Using MyProxy server is not fully supported at the moment.

**Name** | **Description** | **Example**
---|---|---
UseMyProxy | Use myproxy server | UseMyProxy = False

**Systems / Framework / <INSTANCE> / Service / SecurityLogging - Sub-subsection**

SecurityLogging service is used by all server to log all connections.

**Name** | **Description** | **Example**
---|---|---
DataLocation | Directory where log info is kept | DataLocation = data/securityLog

**Systems / Framework / <INSTANCE> / Service / SystemAdministrator - Sub-subsection**

SystemAdministrator service is a tool to control and monitor the DIRAC services and agents.

Extra options are not required to be configured to use this service.
Systems / Framework / <INSTANCE> / Service / SystemLogging - Sub-subsection SystemLoggingHandler is the implementation of the Logging service in the DISET framework. Extra options are not required to be configured to use this service.

Systems / Framework / <INSTANCE> / Service / SystemLoggingReport - Sub-subsection SystemLoggingReportHandler allows a remote system to access the content of the SystemLoggingDB. No extra options are required to be configured.

Systems / Framework / <INSTANCE> / Service / UserProfileManager - Sub-subsection ProfileManager manages web user profiles in the DISET framework. No extra options needs to be configured to use this service.

Systems / Framework / <INSTANCE> / Agents - Sub-subsection In this subsection each agent is described.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Subsection named as the agent is called.</td>
<td>CAUpdateAgent</td>
</tr>
</tbody>
</table>

Common options for all the agents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Log Level associated to the agent</td>
<td>LogLevel = DEBUG</td>
</tr>
<tr>
<td>LogBackends</td>
<td></td>
<td>LogBackends = stdout, server</td>
</tr>
<tr>
<td>MaxCycles</td>
<td>Maximum number of cycles made for Agent</td>
<td>MaxCycles = 500</td>
</tr>
<tr>
<td>MonitoringEnabled</td>
<td>Indicates if the monitoring of agent is enabled. Boolean values</td>
<td>MonitoringEnabled = True</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Each many time a new cycle must start expresed in seconds</td>
<td>PollingTime = 2600</td>
</tr>
<tr>
<td>Status</td>
<td>Agent Status, possible values Active or Inactive</td>
<td>Status = Active</td>
</tr>
</tbody>
</table>

Agents associated with Configuration System:

Systems / Framework / <INSTANCE> / Agents / CAUpdateAgent - Sub-subsection CA Update agent uses the Framework/BundleDelivery service to get up-to-date CAs and CRLs for all agent and servers using the same dirac installation. This agent has no options.

Systems / Framework / <INSTANCE> / Agents / MyProxyRenewalAgent - Sub-subsection Proxy Renewal agent is the key element of the Proxy Repository which maintains the user proxies alive. This Agent allows to run DIRAC with short proxies in the DIRAC proxy manager. It relies on the users uploading proxies for each relevant group to a MyProxy server. It needs to be revised to work with multiple groups. This agent is currently not functional.

The attributes of this agent are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinValidity</td>
<td>Proxy Minimal validity time expressed in seconds</td>
<td>MinValidity = 10000</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Polling time in seconds</td>
<td>PollingTime = 1800</td>
</tr>
<tr>
<td>ValidityPeriod</td>
<td>The period for which the proxy will be extended. The value is in hours</td>
<td>ValidityPeriod = 15</td>
</tr>
</tbody>
</table>
System LoggingDBCleaner erases records whose messageTime column contains a time older than ‘RemoveDate’ days, where ‘RemoveDate’ is an entry in the Configuration Service section of the agent.

The attributes of this agent are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveDate</td>
<td>Each many days the database must be clean Expressed in days</td>
<td>RemoveDate = 30</td>
</tr>
</tbody>
</table>

TopErrorMessagesReporter produces a list with the most common errors injected in the SystemLoggingDB and sends a notification to a mailing list and specific users.

The attributes of this agent are showed in the table below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailList</td>
<td>List of DIRAC users than the reporter going to receive Top Error Messages</td>
<td>MailList = <a href="mailto:mseco@in2p3.fr">mseco@in2p3.fr</a></td>
</tr>
<tr>
<td>NumberOfErrors</td>
<td>Number of top errors to be reported</td>
<td>NumberOfErrors = 10</td>
</tr>
<tr>
<td>QueryPeriod</td>
<td>Each how many time the agent is going to make the query, expressed in days</td>
<td>QueryPeriod = 7</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Login of DIRAC user in charge of review the error message monitor</td>
<td>Reviewer = mseco</td>
</tr>
<tr>
<td>Threshold</td>
<td>Threshold = 10</td>
<td></td>
</tr>
</tbody>
</table>

Framework Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, the value is a URL using dips protocol</td>
<td>Plotting = dips://dirac.eela.if.ufrj.br:9157/Framework/Plotting</td>
</tr>
</tbody>
</table>

Services associated with Framework System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundleDelivery</td>
<td>9158</td>
</tr>
<tr>
<td>Monitoring</td>
<td>9142</td>
</tr>
<tr>
<td>Notification</td>
<td>9154</td>
</tr>
<tr>
<td>Plotting</td>
<td>9157</td>
</tr>
<tr>
<td>ProxyManagement</td>
<td>9152</td>
</tr>
<tr>
<td>SecurityLogging</td>
<td>9153</td>
</tr>
<tr>
<td>SystemAdministrator</td>
<td>9162</td>
</tr>
<tr>
<td>SystemLogging</td>
<td>9141</td>
</tr>
<tr>
<td>SystemLoggingReport</td>
<td>9144</td>
</tr>
<tr>
<td>UserProfileManager</td>
<td>9155</td>
</tr>
</tbody>
</table>

In this subsection are described the databases, services and URLs related with RequestManagement System for each setup.

Databases used by RequestManagement System. Note that each database is a separate subsection.
The databases associated to StorageManagement System are:

- **StorageManagementDB**

### Systems / RequestManagement / <INSTANCE> / Service - Sub-subsection

All the services have common options to be configured for each one. Those options are presented in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log verbosity</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port used by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to communicate with the service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Define to who is required the authorization</td>
<td>Default = all</td>
</tr>
</tbody>
</table>

DataStore services are:

### Systems / StorageManagement / <INSTANCE> / Service / StorageManager - Sub-subsection

### Systems / RequestManagement / <INSTANCE> / URLs - Sub-subsection

RequestManagement Services URLs.

DataStore services are:

### Transformation System configuration

In this subsection are described the databases, services, agents, and URLs related to Transformation System for each setup.

### Systems / Transformation / <INSTANCE> / Agents - Sub-subsection

Agents associated with DataManagement System:
Systems / Transformation / <INSTANCE> / Agents / InputDataAgent - Sub-subsection  The InputDataAgent updates the transformation files of active transformations given an InputDataQuery fetched from the Transformation Service.

Possibility to speedup the query time by only fetching files that were added since the last iteration. Use the CS option RefreshOnly (False by default) and set the DateKey (empty by default) to the meta data key set in the DIRAC FileCatalog.

This Agent also reads some options from Operations/Transformations:

- DataProcessing
- DataManipulation
- ExtendableTransfTypes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullUpdatePeriod</td>
<td>Time after a full update will be done</td>
<td>86400</td>
</tr>
<tr>
<td>RefreshOnly</td>
<td>Only refresh new files, needs the DateKey</td>
<td>False</td>
</tr>
<tr>
<td>DateKey</td>
<td>Meta data key for file creation date</td>
<td></td>
</tr>
<tr>
<td>TransformationTypes</td>
<td>TransformationTypes to handle in this agent instance</td>
<td></td>
</tr>
</tbody>
</table>

Systems / Transformation / <INSTANCE> / Agents / MCExtensionAgent - Sub-subsection  This agent extends the number of tasks given the Transformation definition.

It also uses the Operations / Transformations / Options:

- Transformations/ExtendableTransfTypes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransformationTypes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TasksPerIteration</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>MaxFailureRate</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>MaxWatingJobs</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>EnableFlag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Systems / Transformation / <INSTANCE> / Agents / RequestTaskAgent - Sub-subsection  The Request Task Agent takes request tasks created in the transformation database and submits to the request management system.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>shifterProxy</td>
<td></td>
<td>DataManager</td>
</tr>
<tr>
<td>TransType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PluginLocation</td>
<td>DIRAC.TransformationSystem.Client.TaskManagerPlugin</td>
<td></td>
</tr>
<tr>
<td>maxNumberOfThreads</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>TasksPerLoop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TaskUpdateStatus</td>
<td>Checking, Deleted, Killed, Staging, Stalled, Matched, Scheduled, Rescheduled, Completed, Submitted, Assigned, Received, Waiting, Running</td>
<td></td>
</tr>
<tr>
<td>SubmitTasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SubmitStatus</td>
<td>Active, Completing</td>
<td></td>
</tr>
<tr>
<td>MonitorTasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MonitorFiles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CheckReserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CheckReservedStatus</td>
<td>Active, Completing, Stopped</td>
<td></td>
</tr>
<tr>
<td>UpdateTaskStatus</td>
<td>Active, Completing, Stopped</td>
<td></td>
</tr>
<tr>
<td>UpdateFileStatus</td>
<td>Active, Completing, Stopped</td>
<td></td>
</tr>
</tbody>
</table>

**Systems / Transformation / <INSTANCE> / Agents / TransformationAgent - Sub-subsection** The TransformationAgent processes transformations found in the transformation database.

This Agent also reads some options from *Operations / Transformations / Options*:

- DataProcessing
- DataManipulation

And from *Operations / TransformationPlugins / Options*, depending on the Plugin used for the Transformation:

- SortedBy
- MaxFiles
- NoUnusedDelay

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PluginLocation</td>
<td></td>
<td>DIRAC.TransformationSystem.Agent.TransformationPlugin</td>
</tr>
<tr>
<td>transformationStatus</td>
<td></td>
<td>Active, Completing, Flush</td>
</tr>
<tr>
<td>MaxFiles</td>
<td></td>
<td>5000</td>
</tr>
<tr>
<td>TransformationTypes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReplicaCacheValidity</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>maxThreadsInPool</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NoUnusedDelay</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Transformation</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>TransfIDMeta</td>
<td>MetaData key to use to identify output data</td>
<td>TransfIDMeta=TransformationID</td>
</tr>
<tr>
<td>DirectoryLocations</td>
<td>Location of the OutputData</td>
<td>TransformationDB, MetadataCatalog</td>
</tr>
<tr>
<td>ActiveSEs</td>
<td>From which SEs files will be removed</td>
<td>[]</td>
</tr>
<tr>
<td>EnableFlag</td>
<td></td>
<td>True/False</td>
</tr>
<tr>
<td>TransformationLogSE</td>
<td>StorageElement holding log files</td>
<td>LogSE</td>
</tr>
<tr>
<td>ArchiveAfter</td>
<td>How many days before archiving transformations</td>
<td>ArchiveAfter=7</td>
</tr>
<tr>
<td>shifterProxy</td>
<td>shifter to use to operations</td>
<td>shifterProxy=DataManager</td>
</tr>
</tbody>
</table>

**Systems / Transformation / <INSTANCE> / Agents / ValidateOutputDataAgent - Sub-subsection**  
The ValidateOutputDataAgent runs few integrity checks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransformationTypes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DirectoryLocations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActiveSEs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TransfIDMeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableFlag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Systems / Transformation / <INSTANCE> / Agents / WorkflowTaskAgent - Sub-section**  
The WorkflowTaskAgent takes workflow tasks created in the TransformationDB and submits them to the WMS. Since version v6r13 there are some new capabilities in the form of TaskManager plugins.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TaskUpdateStatus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shifterProxy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CheckReserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MonitorFiles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SubmitTasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TasksPerLoop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MonitorTasks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Systems / Transformation / <INSTANCE> / Databases - Sub-section**  
Databases used by RequestManagement System. Note that each database is a separate subsection.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DATABASE_NAME&gt;</td>
<td>Subsection. Database name Database name</td>
<td>TransformationDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/DBName</td>
<td>Database host server where the DB is located</td>
<td>DBName = TransformationDB</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/Host</td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>Host = db01.in2p3.fr</td>
</tr>
<tr>
<td>&lt;DATABASE_NAME&gt;/MaxQueueSize</td>
<td>Maximum number of simultaneous queries to the DB per instance of the client</td>
<td>MaxQueueSize = 10</td>
</tr>
</tbody>
</table>

The databases associated to Transformation System are: - TransformationDB

**Systems / Transformation / <INSTANCE> / Services - Sub-section**  
All the services have common options to be configured for each one. Those options are presented in the following table:
### DIRAC Documentation, Release integration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Level of log verbosity</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Log backends</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LogBackends += server</td>
</tr>
<tr>
<td>MaskRequestParameters</td>
<td>Request to mask the values, possible values: yes or no</td>
<td>MaskRequestParameters = yes</td>
</tr>
<tr>
<td>MaxThreads</td>
<td>Maximum number of threads used in parallel for the server</td>
<td>MaxThreads = 50</td>
</tr>
<tr>
<td>Port</td>
<td>Port used by DIRAC service</td>
<td>Port = 9140</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol used to communicate with the service</td>
<td>Protocol = dips</td>
</tr>
<tr>
<td>Authorization</td>
<td>Subsection used to define which kind of Authorization is required to talk with the service</td>
<td>Authorization</td>
</tr>
<tr>
<td>Authorization/Default</td>
<td>Define to who is required the authorization</td>
<td>Default = all</td>
</tr>
</tbody>
</table>

Transformation services are:

**Systems / Transformation / <INSTANCE> / Services / TransformationManager - Sub-subsection**

**Systems / Transformation / <INSTANCE> / URLs - Sub-subsection**  
Transformation Services URLs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICE_NAME&gt;</td>
<td>URL associated with the service, value URL using dips protocol</td>
<td>TransformationManager = dips://.eela.if.ufrj.br:9131/Transformation/TransformationManager</td>
</tr>
</tbody>
</table>

Services associated with RequestManagement System:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransformationManager</td>
<td>9131</td>
</tr>
</tbody>
</table>

**Web Portal configuration**

**Other sections**

**System Authorization**

For each system authorization rules must be configured, a short introduction about the different options available are showed in the next table:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmsManagement</td>
<td>Allow to set notifications and manage alarms</td>
<td></td>
</tr>
<tr>
<td>BookkeepingManagement</td>
<td>Allow Bookkeeping Management</td>
<td></td>
</tr>
<tr>
<td>CSAdministrator</td>
<td>CS Administrator - possibility to edit the Configuration Service</td>
<td></td>
</tr>
<tr>
<td>FileCatalogManagement</td>
<td>Allow FC Management</td>
<td></td>
</tr>
<tr>
<td>FullDelegation</td>
<td>Allow getting full delegated proxies</td>
<td></td>
</tr>
<tr>
<td>GenericPilot</td>
<td>Generic pilot</td>
<td></td>
</tr>
<tr>
<td>JobAdministrator</td>
<td>Job Administrator</td>
<td></td>
</tr>
<tr>
<td>JobSharing</td>
<td>Job sharing among members of a group</td>
<td></td>
</tr>
<tr>
<td>LimitedDelegation</td>
<td>Allow getting only limited proxies (ie. pilots)</td>
<td></td>
</tr>
<tr>
<td>NormalUser</td>
<td>Normal user operations</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>Pilot</td>
<td>Private pilot</td>
<td></td>
</tr>
<tr>
<td>PrivateLimitedDelegation</td>
<td>Allow getting only limited proxies for one self</td>
<td></td>
</tr>
<tr>
<td>ProductionManagement</td>
<td>Allow managing production</td>
<td></td>
</tr>
<tr>
<td>ProxyManagement</td>
<td>Allow managing proxies</td>
<td></td>
</tr>
<tr>
<td>PPGAuthority</td>
<td>Allow production request approval on behalf of PPG</td>
<td></td>
</tr>
<tr>
<td>ServiceAdministrator</td>
<td>DIRAC Service Administrator</td>
<td></td>
</tr>
<tr>
<td>SiteManager</td>
<td>Site Manager</td>
<td></td>
</tr>
<tr>
<td>TrustedHost</td>
<td>Host defined in the system to be trusted</td>
<td></td>
</tr>
</tbody>
</table>

**Correspondence between port number and DIRAC Services**

DIRAC services and ports are expressed in the next two tables:

- Ordered by Systems / Services
- Ordered by Port
### Ordered by System / Services

<table>
<thead>
<tr>
<th>Port</th>
<th>System</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>9133</td>
<td>Accounting</td>
<td>DataStore</td>
</tr>
<tr>
<td>9134</td>
<td>Accounting</td>
<td>ReportGenerator</td>
</tr>
<tr>
<td>9135</td>
<td>Configuration</td>
<td>Server</td>
</tr>
<tr>
<td>9197</td>
<td>DataManagement</td>
<td>FileCatalog</td>
</tr>
<tr>
<td>9148</td>
<td>DataManagement</td>
<td>StorageElement</td>
</tr>
<tr>
<td>9149</td>
<td>DataManagement</td>
<td>StorageElementProxy</td>
</tr>
<tr>
<td>9158</td>
<td>Framework</td>
<td>BundleDelivery</td>
</tr>
<tr>
<td>9142</td>
<td>Framework</td>
<td>Monitoring</td>
</tr>
<tr>
<td>9154</td>
<td>Framework</td>
<td>Notification</td>
</tr>
<tr>
<td>9157</td>
<td>Framework</td>
<td>Plotting</td>
</tr>
<tr>
<td>9152</td>
<td>Framework</td>
<td>ProxyManager</td>
</tr>
<tr>
<td>9153</td>
<td>Framework</td>
<td>SecurityLogging</td>
</tr>
<tr>
<td>9162</td>
<td>Framework</td>
<td>SystemAdministrator</td>
</tr>
<tr>
<td>9141</td>
<td>Framework</td>
<td>SystemLogging</td>
</tr>
<tr>
<td>9144</td>
<td>Framework</td>
<td>SystemLoggingReport</td>
</tr>
<tr>
<td>9155</td>
<td>Framework</td>
<td>UserProfileManager</td>
</tr>
<tr>
<td>9143</td>
<td>RequestManagement</td>
<td>RequestManager</td>
</tr>
<tr>
<td>9132</td>
<td>WorkloadManagement</td>
<td>JobManager</td>
</tr>
<tr>
<td>9130</td>
<td>WorkloadManagement</td>
<td>JobMonitoring</td>
</tr>
<tr>
<td>9136</td>
<td>WorkloadManagement</td>
<td>JobStateUpdate</td>
</tr>
<tr>
<td>9170</td>
<td>WorkloadManagement</td>
<td>Matcher</td>
</tr>
<tr>
<td>9196</td>
<td>WorkloadManagement</td>
<td>SandboxStore</td>
</tr>
<tr>
<td>9145</td>
<td>WorkloadManagement</td>
<td>WMSAdministrator</td>
</tr>
</tbody>
</table>

### Ordered by port number

<table>
<thead>
<tr>
<th>Port</th>
<th>System</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>9130</td>
<td>WorkloadManagement</td>
<td>JobMonitoring</td>
</tr>
<tr>
<td>9132</td>
<td>WorkloadManagement</td>
<td>JobManager</td>
</tr>
<tr>
<td>9133</td>
<td>Accounting</td>
<td>DataStore</td>
</tr>
<tr>
<td>9134</td>
<td>Accounting</td>
<td>ReportGenerator</td>
</tr>
<tr>
<td>9135</td>
<td>Configuration</td>
<td>Server</td>
</tr>
<tr>
<td>9136</td>
<td>WorkloadManagement</td>
<td>JobStateUpdate</td>
</tr>
<tr>
<td>9141</td>
<td>Framework</td>
<td>SystemLogging</td>
</tr>
<tr>
<td>9142</td>
<td>Framework</td>
<td>Monitoring</td>
</tr>
<tr>
<td>9143</td>
<td>RequestManagement</td>
<td>RequestManager</td>
</tr>
<tr>
<td>9144</td>
<td>Framework</td>
<td>SystemLoggingReport</td>
</tr>
<tr>
<td>9145</td>
<td>WorkloadManagement</td>
<td>WMSAdministrator</td>
</tr>
<tr>
<td>9148</td>
<td>DataManagement</td>
<td>StorageElement</td>
</tr>
<tr>
<td>9149</td>
<td>DataManagement</td>
<td>StorageElementProxy</td>
</tr>
<tr>
<td>9152</td>
<td>Framework</td>
<td>ProxyManager</td>
</tr>
<tr>
<td>9153</td>
<td>Framework</td>
<td>SecurityLogging</td>
</tr>
<tr>
<td>9154</td>
<td>Framework</td>
<td>Notification</td>
</tr>
<tr>
<td>9155</td>
<td>Framework</td>
<td>UserProfileManager</td>
</tr>
<tr>
<td>9157</td>
<td>Framework</td>
<td>Plotting</td>
</tr>
<tr>
<td>9158</td>
<td>Framework</td>
<td>BundleDelivery</td>
</tr>
<tr>
<td>9162</td>
<td>Framework</td>
<td>SystemAdministrator</td>
</tr>
<tr>
<td>9170</td>
<td>WorkloadManagement</td>
<td>Matcher</td>
</tr>
<tr>
<td>9196</td>
<td>WorkloadManagement</td>
<td>SandboxStore</td>
</tr>
<tr>
<td>9197</td>
<td>DataManagement</td>
<td>FileCatalog</td>
</tr>
</tbody>
</table>

Note: This configuration file can be edited by hand, but we strongly recommend you to configure using DIRAC Web Portal.
### 2.6.3 DIRAC Section

The *DIRAC* section contains general parameters needed in most of installation types. In the table below options directly placed into the section are described.

- **VirtualOrganization** The name of the Virtual Organization of the installation User Community. The option is defined in a single VO installation.
  
  ValueType: string

- **Setup** The name of the DIRAC installation Setup. This option is defined in the client installations to define which subset of DIRAC Systems the client will work with. See *DIRAC Configuration* for the description of the DIRAC configuration nomenclature.
  
  ValueType: string

- **Extensions** The list of extensions to the Core DIRAC software used by the given installation.
  
  ValueType: list

#### Configuration subsection

The *Configuration* subsection defines several options to discover and use the configuration data.

- **Configuration/Servers** This option defines a list of configuration servers, both master and slaves, from which clients can obtain the configuration data.
  
  ValueType: list

- **Configuration/MasterServer** the URL of the Master Configuration Server. This server is used for updating the Configuration Service.
  
  ValueType: string

- **Configuration/EnableAutoMerge** Enables automatic merging of the modifications done in parallel by several clients.
  
  ValueType: boolean

#### Security subsection

The *Security* subsection defines several options related to the DIRAC/DISET security framework.

- **Security/UseServerCertificates** Flag to use server certificates and not user proxies. This is typically true for the server installations.
  
  ValueType: boolean

- **Security/SkipCAChecks** Flag to skip the server identity by the client. The flag is usually defined in the client installations.
  
  ValueType: boolean

#### Setups subsection

The subsection defines the names of different DIRAC *Setups* as subsection names. In each subsection of the *Setup* section the names of corresponding System instances are defined. In the example below “Production” instances of *Systems Configuration and Framework* are defined as part of the “Dirac-Prduction” *Setup*:
2.6.4 Managing DIRAC Configuration

2.7 DIRAC Systems

In this chapter the description of DIRAC Systems is presented. For each System, the functionality of all the constituent components are described together with their configuration parameters.

2.7.1 Accounting System

Table of contents

- Accounting System
  - Multi-DB accounting

Multi-DB accounting

Since v6r12 each accounting type can be stored in a different DB. By default all accounting types data will be stored in the database defined under /Systems/Accounting/_Instance_/Databases/AccountingDB. To store a type data in a different database (say WMSHistory) define the data base location under the databases directory. Then define /Systems/Accounting/_Instance_/Databases/MultiDB and set an option with the type name and value pointing to the database to use. For instance:

Systems
{ Accounting
  Development
  AccountingDB
  { Host = localhost
    User = dirac
    Password = dirac
    DBName = accounting
  }
  Acc2
  { Host = somewhere.internet.net
    User = dirac
  }
With the previous configuration all accounting data will be stored and retrieved from the usual database except for the _WMSHistory_ type that will be stored and retrieved from the _Acc2_ database.

### 2.7.2 Configuration System

#### 2.7.3 Data Management System

**Requests’ Processing Agents**

- **author** Krzysztof Daniel Ciba  
  <Krzysztof.Ciba@NOSPAMgmail.com>
- **date** Tue, 13th Mar 2012
- **version** first and final

**Requests types and operations**

There are three agents in the DIRAC DataManagementSystem dealing with requests, one for each request’s type: **TransferAgent**, processing *transfer* requests, **RemovalAgent**, processing *removal* requests and **RegistrationAgent** for handling *register* requests. The current implementation is able to handle the following operations:

- **transfer**
  - *putAndRegister*
  - *replicateAndRegister*
- **removal**
  - *removeFile*
  - *replicaRemoval*
  - *reTransfer*
- **register**
  - *registerFile*
The `putAndRegister` and `reTransfer` operations are only used in LHCb online environment, while all the others are of general use and can be executed for any DIRAC user.

### Design

#### Base classes and inheritance
The common functionality in all agents has been grouped and implemented inside a base class `RequestAgentBase`. This class is responsible for creating and managing of the `ProcessPool` and tasks, requests’ retrieval and creation, execution of tasks and their callbacks. The inheritance diagram is presented below.

As you can see the `TransferAgent` is somehow special, as depending of configuration, it is able to execute requests in two modes:

- task execution using sub-processes,
- files scheduling for FTS.

Same story applies to tasks, there are three specialized classes to handling particular requests types:

- `TransferTask` for execution of transfer requests
- `RegistrationTask` for execution of register requests
- `RemovalTask` for execution of removal requests

and all of them are inherited from base class `RequestTask`, allowing common functionality, like operation dispatcher, common tools, logging, monitoring, request execution and updating etc. The inheritance diagram for tasks classes is shown below:
As all RequestTask-inherited objects are executed in separate sub-processes and hence in pure and empty python environment, the most important and common global DIRAC objects are created and globalized in RequestTask constructor. This includes gLogger, gConfig, gProxyManager, S_OK and S_ERROR. The constructor also imports a set of common modules: os, sys, re, time and everything from types module.

This globalization is made by attaching them to the _builtins_ module symbols, using :makeGlobal: function:

```python
def makeGlobal( self, objName, objDef ):
    """ export :objDef: to global name space using :objName: name

    :param self: self reference
    :param str objName: symbol name
    :param mixed objDef: symbol definition
    :throws: NameError if symbol of that name is already in
    """
    if objName not in __builtins__:
        if type( __builtins__ ) == type( {} ):
            __builtins__[objName] = objDef
        else:
            setattr( __builtins__, objName, objDef )
    return True
```

where objName is a symbol’s name used together with reference to its definition objDef. This technique will allow to use them in all functions of inherited classes like a global objects defined on a module level.

All other DIRAC tools and clients (i.e. RequestManager) are private in RequestTask class and will be imported and instantiated on demand during first call. They are accessible using facades - a proxied public methods, i.e. if you are going to use ReplicaManager just call:

```
self.replicaManager().someMethod()
```

The main reason for using facades is keeping the memory consumption low as modules are imported and objects are created only on demand. All currently proxied tools are:

- DataLoggingClient – self.dataLoggingClient()
- ReplicaManager – self.replicaManager()
- RequestClient – self.requestClient()
- StorageFactory – self.storageFactory()

The logger message handlers for all levels are also proxied, so you can directly use them in your code, i.e.:
self.info("An info message")
self.debug("This will be shown only in debug")

Concerning `MonitoringClient` (or better known its global instance `gMonitor`), if someone wants to send some metric over there, she has to put in agent’s code registration of activity and then in a particular task use `RequestTask.addMark` to save monitoring data. All monitored activities are held in `RequestTask.__monitor` dict which at the end of processing is returned from `RequestTask.__call__`. The values are then pushed to the default callback function defined in `RequestAgentBase`, from where all of them are exported to `gMonitor` instance created in agent’s module.

**Request execution** For handling sub-request one has to register their actions handlers using `RequestTask.addOperationAction` method. This method checks if handler is defined as a method of inherited class and then puts its definition into internal operation dispatcher dictionary with a key of sub-request’s operation name.

Each operation handler should follow the signature:

```python
def operationName( self, index, requestObj, subRequestAttrs, subRequestFiles )
```

where index is a sub-request counter, `requestObj` is a `RequestContainer` instance, `subRequestAttrs` is a dict with sub-request attributes and `subRequestFiles` is a dict with files attached to the sub-request. Those handlers should always return `S_OK` with value of (modified or not)`requestObj` or `S_ERROR` with some error message otherwise.

The processing of request os done automatically in `RequestTask.__call__`, one doesn’t have to worry about changing credentials, looping over sub-requests or request finalizing – only sub-request processing matters in all child classes.

Let’s follow the path of request processing using ‘removal’ request example (there is no difference for ‘registration’ and ‘transfer’ in task execution mode):

Execution of request starts with retrieval of ‘Waiting’ request out of `RequestClient` using `RequestAgentBase.getRequest`.

The request is then serialized to XML string and put into a kwargs dictionary of `ProcessPool.createAndQueueTask`. Copy of that information is also stored inside agent’s `requestHolder` dictionary, used to clean up assigned requests during agent’s finalization.
Once there is a free slot in ProcessPool task queue, ProcessTask is en-queued and lands in WorkingProcess instance, which is managing task execution in a separate sub-process. Inside its run method, an instance of ProcessTask is created together with its callable object – our RemovalTask and immediately executed.

Inside RequestTask.__call__ function request is checked against its owner: if OwnerDN and OwnerGroup attributes are set, proxy for this user is created ans swapped with the default (DataManager) proxy. After that processing enters the loop over sub-requests. If particular sub-request has ‘Waiting’ state and its operation matches registered one, the sub-request files and attributes are passed as parameters to the operation handler. After the handler’s execution, the request is again serialized to the XML and updated using RequestClient. At this stage finalization of request is also triggered, but only if JobID attribute is different from 0.

The results of RequestTask.__call__ are stored inside of ProcessTask instance, which again is en-queued into results queue of ProcessPool. Now processing is shifted to the ProcessPool instance again. The results are picked up and sent to the callback function, which is used to copy over the monitored records to the gMonitor and to clean up reference of original request from requestHolder.

The motivation for adding requestHolder dictionary is to have in place a fail-over mechanism for tasks that have never returned any values from their sub-process execution (i.e. sub-process is stuck waiting for a signal, there is a never released lock in the underlying library, the sub-process has been killed by ProcessPool or evil and mad person from outside world). In that case, no matter what was the original occasion, the assigned requests are put back to the RequestClient at the end of last cycle of the agent, when RequestAgentBase.finalize method is called.
TransferAgent and FTS scheduling  The TransferAgent has two modes of operation:

- standalone, when all requests are handled using ProcessPool and TransferTask,
- scheduling for FTS with fail-back TransferTask functionality.

By default FTS scheduling is disabled and all requests are processed using tasks. The fail-back mechanism is triggered in case that:

- the FTS channels between SourceSE and TargetSE is not defined,
- there is a trouble to define correct replication tree,
- request’s owner is different from DataManager,
- sub-request operation isn’t a ‘replicateAndRegister’.
Execution of the FTS mode is following (see above diagram):

1. The request is checked against its owner, if OwnerDN and OwnerGroup attributes are set, request is passed to the task execution mode.

2. The value Waiting sub-requests operation attributes are compared with the only one handled ‘replicateAndRegister’, if any of sub-request operation is different from ‘replicateAndRegister’, the request is passed to the task execution mode.

3. Request is sent to \texttt{TransferAgent.schedule} function, where subsequent methods are called:

   - \texttt{TransferTask.checkReadyReplicas}: all files and their registered replicas are compared, if a particular file has been already replicated, its status is set to ‘Done’

   - \texttt{TransferAgent.registerFiles}: fail-over registration mechanism for files that have been already replicated (in \texttt{FTSSubmitAgent}) but not registered (error in registration in \texttt{FTSMonitorAgent})

   - \texttt{TransferAgent.scheduleFiles}: for all ‘Waiting’ files the replication tree is constructed using \texttt{StrategyHandler}, when it is ready, \texttt{TransferDB.Channels}, \texttt{TransfeDB.FileToCat} and \texttt{TransferDB.ReplicationTree} records are pushed into \texttt{TransferDB} and file status is set to ‘Scheduled’, if for any reason \texttt{ReplicationTree} cannot be created, the request is put into task execution mode.

4. When all statuses of files are set to ‘Done’ in previous methods, sub-request status is set to ‘Done’ and the same check is repeated for all sub-requests and request itself. At this stage request is also finalized, if JobID attribute is different from 0.

The request is going to be executed many times, until all replicas are created, but please notice, that FTS scheduling is done only once, during first execution, when there are still ‘Waiting’ files in sub-request. All following operations are only repeating \texttt{TransferTask.checkReadyReplicas} and \texttt{TransferAgent.registerFiles} calls to update files statuses.

**Configuration and installation**
### Options common to all agents

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Logging level</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Logging handlers</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Time period in seconds for agent’s polling</td>
<td>PollingTime = 60</td>
</tr>
<tr>
<td>ControlDirectory</td>
<td>Control directory location</td>
<td>ControlDirectory = Control/DataManagement/0</td>
</tr>
<tr>
<td>RequestsPerCycle</td>
<td>Number of requests to process in one agent cycle</td>
<td>RequestsPerCycle = 10</td>
</tr>
<tr>
<td>MinProcess</td>
<td>Minimal number of sub-processes running</td>
<td>MinProcess = 1</td>
</tr>
<tr>
<td>MaxProcess</td>
<td>Maximal number of sub-processes running</td>
<td>MaxProcess = 4</td>
</tr>
<tr>
<td>ProcessPoolQueueSize</td>
<td>Capacity of task queue in ProcessPool</td>
<td>ProcessPoolQueueSize = 10</td>
</tr>
<tr>
<td>shifterProxy</td>
<td>Default proxy used to process request</td>
<td>shifterProxy = DataManager</td>
</tr>
<tr>
<td>RequestType</td>
<td>Request type:</td>
<td>RequestType = &lt;requestType&gt;</td>
</tr>
</tbody>
</table>

- **Request type:**
  - register for RegistrationAgent
  - removal for RemovalAgent
  - transfer for TransferAgent

#### <TaskName> subsection (<TaskName> = RegistrationTask, RemovalTask, TransferTask)

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Logging level</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Logging handlers</td>
<td>LogBackends = stdout</td>
</tr>
<tr>
<td>TaskMode</td>
<td>Flag to disable/enable tasks for processing</td>
<td>TaskMode = True</td>
</tr>
<tr>
<td>FTSMode</td>
<td>Flag to disable/enable FTS scheduling</td>
<td>FTSMode = True</td>
</tr>
<tr>
<td>ThroughputTimescale</td>
<td>Time period used to monitor FTS transfer history</td>
<td>ThroughputTimescale = 3600</td>
</tr>
</tbody>
</table>

#### StrategyHandler subsection

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HopSigma</td>
<td>Acceptable time shift to start of FTS transfer</td>
<td>HopSigma = 0.0</td>
</tr>
<tr>
<td>SchedulingType</td>
<td>Transfer speed calculation:</td>
<td>SchedulingType =</td>
</tr>
<tr>
<td></td>
<td>- number of files per hour (Files)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- amount of data per hour (Throughput)</td>
<td></td>
</tr>
<tr>
<td>ActiveStrategies</td>
<td>List of active strategies to use: DynamicThroughput, MinimiseTotalWait, Simple, Swarm</td>
<td>ActiveStrategies =</td>
</tr>
<tr>
<td>AcceptableFailureRate</td>
<td>Percentage limit of success rate in monitored FTS transfers to accept/reject FTS channel from scheduling.</td>
<td>AcceptableFailureRate =</td>
</tr>
</tbody>
</table>
Default options  Default configuration for all agents:

TransferAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/TransferAgent
    RequestsPerCycle = 10
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = transfer
    shifterProxy = DataManager
    TaskMode = True
    FTSMode = True
    ThroughputTimescale = 3600
    StrategyHandler {
        LogLevel = INFO
        LogBackends = stdout
        HopSigma = 0.0
        SchedulingType = File
        ActiveStrategies = MinimiseTotalWait
        AcceptableFailureRate = 75
    }
    TransferTask {
        LogLevel = INFO
        LogBackends = stdout
    }
}

RegistrationAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/RegistrationAgent
    RequestsPerCycle = 10
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = register
    shifterProxy = DataManager
    RegistrationTask {
        LogLevel = INFO
        LogBackends = stdout
    }
}

RemovalAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/RemovalAgent
    RequestsPerCycle = 50
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = removal
    shifterProxy = DataManager
    RemovalTask {

```
Installation procedure

1. On a fresh DIRAC installation procedure is very simple:
   - **RegistrationAgent** and **RemovalAgent**
     Follow the normal installation procedure, but make sure the new configuration sections is in place and updated.
   - **TransferAgent** in mixed mode (**FTSMode** and **TaskMode**)
     This is the default configuration that can be used i.e. in LHCb DIRAC prod system.
     Make sure FTS agents and databases are installed and properly configured (**TransferDB**, **FTSMonitorAgent**, **FTSSubmitAgent** and **FTSCleanupAgent**). Install **TransferAgent**.
   - **TransferAgent** in **TaskMode** only
     This mode should be used in LHCb online version of DIRAC or for VOs without FTS service available.
     Install **TransferAgent**, disable **FTSMode** in its configuration section.

2. Upgrading existing DIRAC installation:
   - stop **TransferAgent**, **ReplicationScheduler**, **RegistrationAgent** and **RemovalAgent** agents
   - update configuration sections by replacing **TransferAgent**, **RemovalAgent** and **RegistrationAgent** and deleting **ReplicationScheduler** fragments
   - start new agents using `runsvctrl u` command

**FTS support in DIRAC**

**author** Krzysztof Daniel Ciba <Krzysztof.Ciba@NOSPAMgmail.com>
**date** Tue, 10th Jun 2013
**version** first and final
The DIRAC Data Management System (DMS), together with the DIRAC Storage Management System (SMS) provides the necessary functionality to execute and control all activities related with your data. The DMS provides from the basic functionality to upload a local file in a StorageElement (SE) and register the corresponding replica in the FileCatalog (FC) to massive data replications using FTS or retrievals of data archived on Tape for it later processing.

To achieve this functionality the DMS and SMS require a proper description of the involved external servers (SE, FTS, etc.) as well as a number of Agents and associated Servers that animate them. In the following sections the different aspects of each functional component are explained in some detail.

**StorageElements**

DIRAC provides an abstraction of a SE interface that allows to access different kind of them with a single interface. The access to each kind of SE (SRMv2, DIRAC SE, ...) is achieved by using specific plugin modules that provide a common interface. The information necessary to define the proper plugin module and to properly configure this plugin to access a certain SE has to be introduced in the DIRAC Configuration. An example of such configuration is:

```plaintext
CERN-USER
{
  ReadAccess = Active
  WriteAccess = Active
  AccessProtocol.1
  {
    ProtocolName = SRM2
    Access = remote
    Protocol = srm
    Host = srm-lhcb.cern.ch
    Port = 8443
    WSUrl = /srm/managerv2?SFN=
  }
}
```
FTS transfers in DIRAC

DIRAC DMS can be configured to make use of FTS servers in order to schedule and monitor efficient transfer of large amounts of data between SEs. As of today, FTS servers are only able to handle transfers between SRM SEs. You will need to define at least two different SRM StorageElements in your Configuration and one FTS endpoint. In the current implementation of the DIRAC FTS interface FTS transfers are always assigned to the FTS server associated to the Site local to the destination SE. However you can associate the same FTS server to more than one site.

In order to configure and test support for FTS transfers in your DIRAC installation you should follow these steps:

- Make sure that there are FTS servers configured for the use of your VirtualOrganization. You can find this out, for instance, by using the “lcg-infosites” command of a gLite User Interface:

  ```
  $ lcg-infosite --vo <name of your VO> fts
  $ # for instance:
  $ lcg-infosites --vo lhcb fts
  https://fts.pic.es:8443/glite-data-transfer-fts/services/FileTransfer
  https://fts-fzk.gridka.de:8443/glite-data-transfer-fts/services/FileTransfer
  https://fts.cr.cnaf.infn.it:8443/glite-data-transfer-fts/services/FileTransfer
  https://lcgfts.gridpp.rl.ac.uk:8443/glite-data-transfer-fts/services/FileTransfer
  https://fts-t2-service.cern.ch:8443/glite-data-transfer-fts/services/FileTransfer
  https://fts22-t0-export.cern.ch:8443/glite-data-transfer-fts/services/FileTransfer
  ```

- Determine which channels are supported on a particular FTS server. You can know that using the command `glite-transfer-channel-list`. You need to use the `-s` option and pass one of the above URLs replacing “FileTransfer” by “ChannelManagement”. Channels are list with the format Site1-Site2, STAR is a keyword that applies to any site.

  ```
  $ glite-transfer-channel-list -s https://fts.pic.es:8443/glite-data-transfer-fts/services/ChannelManagement
  STAR-NCG
  STAR-LIPCOIMBRA
  BSC-PIC
  LAPALMA-PIC
  PIC-NCG
  STAR-PIC
  ...
  $ glite-transfer-channel-list -s https://fts.pic.es:8443/glite-data-transfer-fts/services/ChannelManagement
  Channel: STAR-PIC
  Between: * and PIC
  State: Active
  Contact: fts-support@pic.es
  Bandwidth: 0
  Nominal throughput: 0
  Number of files: 50, streams: 5
  Number of VO shares: 5
  VO 'atlas' share is: 50
  VO 'cms' share is: 50
  VO 'dteam' share is: 50
  VO 'lhcb' share is: 50
  VO 'ops' share is: 50
  ```
• Include the URL of the FTS server in the DIRAC Configuration:

```
```

• Now you need to make sure that the DIRAC components that take care of FTS transfers are in place. You need to configure and startup a number of components. This can be done with the “dirac-setup-server” command and a the following FTS.cfg describing what you need:

```
LocalInstallation
{
    Systems = DataManagement, RequestManagement
    DataBases = RequestDB
    Services = DataManagement/TransferDBMonitoring
    Agents = DataManagement/FTSSubmitAgent, DataManagement/FTSMonitorAgent
}
```

• Then one needs to configure the DIRAC Channels that will be handled by the FTS Agents. The methods to create and manipulate the DIRAC Channels for FTS are not exposed on a Service interface. This has to be done with a simple python script from the server:

```python
from DIRAC.Core.Base import Script
Script.parseCommandLine()
from DIRAC.DataManagementSystem.DB.TransferDB import TransferDB

sourceSite = 'ShortSite-Name1'  # LCG.CERN.ch -> CERN
destinationSite = 'ShortSite-Name2'

transferDB = TransferDB()
res = transferDB.createChannel( sourceSite, destinationSite )
if not res['OK']:
    print res['Message']
    exit(-1)

channelID = res['Value']
print 'Created FTS Channel %s' % channelID
```

• At this point some transfer can be attempted between the configured SEs. For that purpose you can use the command line script:

```
$ dirac-dms-fts-submit -h
Submit an FTS request, monitor the execution until it completes
Usage:
dirac-dms-fts-submit [option|cfgfile] ... LFN sourceSE targetSE
Arguments:
    LFN: Logical File Name or file containing LFNs
    sourceSE: Valid DIRAC SE
    targetSE: Valid DIRAC SE
General options:
    -o: --option= : Option=value to add
    -s: --section= : Set base section for relative parsed options
    -c: --cert= : Use server certificate to connect to Core Services
    -d --debug : Set debug mode (~dd is extra debug)
    -h --help : Shows this help
```

```
$ dirac-dms-fts-submit /lhcb/user/r/rgracian/fts_test CNAF-USER PIC-USER
Submitted b3c7c25a-1d14-11e1-abe9-dc229ac9908c @ https://fts.pic.es:8443/glite-data-transfer-fts/services/FileTransfer
|====================================================================================================| 100.0% Finished
```

2.7. DIRAC Systems 173
Using this script, the request to the FTS server will be formulated following the information configured in DIRAC, and will be submitted form your client to the selected FTS server with your local credential. Make sure you are using a proxy that is authorized at your FTS server (usually only some specific users in the VO are allowed, contact the administrators of the site offering you this server in case of doubts).

Requests' Processing Agents

**author**  
Krzysztof Daniel Ciba <Krzysztof.Ciba@NOSPAMgmail.com>

**date**  
Tue, 13th Mar 2012

**version**  
first and final

Requests types and operations

There are three agents in the DIRAC DataManagementSystem dealing with requests, one for each request’s type: TransferAgent, processing transfer requests, RemovalAgent, processing removal requests and RegistrationAgent for handling register requests. The current implementation is able to handle the following operations:

- **transfer**
  - putAndRegister
  - replicateAndRegister
- **removal**
  - removeFile
  - replicaRemoval
  - reTransfer
- **register**
  - registerFile

The putAndRegister and reTransfer operations are only used in LHCb online environment, while all the others are of general use and can be executed for any DIRAC user.

Design

**Base classes and inheritance**  
The common functionality in all agents has been grouped and implemented inside a base class RequestAgentBase. This class is responsible for creating and managing of the ProcessPool and tasks, requests’ retrieval and creation, execution of tasks and their callbacks. The inheritance diagram is presented below.
As you can see the TransferAgent is somehow special, as depending of configuration, it is able to execute requests in two modes:

- task execution using sub-processes,
- files scheduling for FTS.

Same story applies to tasks, there are three specialized classes to handling particular requests types:

- TransferTask for execution of transfer requests
- RegistrationTask for execution of register requests
- RemovalTask for execution of removal requests

and all of them are inherited from base class RequestTask, allowing common functionality, like operation dispatcher, common tools, logging, monitoring, request execution and updating etc. The inheritance diagram for tasks classes is shown below:

As all RequestTask-inherited objects are executed in separate sub-processes and hence in pure and empty python environment, the most important and common global DIRAC objects are created and globalized in RequestTask.
constructor. This includes gLogger, gConfig, gProxyManager, S_OK and S_ERROR. The constructor also imports a set of common modules: os, sys, re, time and everything from types module.

This globalization is made by attaching them to the __builtins__ module symbols, using :makeGlobal: function:

```python
def makeGlobal( self, objName, objDef ):
    """ export :objDef: to global name space using :objName: name
    
    :param self: self reference
    :param str objName: symbol name
    :param mixed objDef: symbol definition
    :throws: NameError if symbol of that name is already in 
    """
    if objName not in __builtins__:
        if type( __builtins__) == type( {}):
            __builtins__[objName] = objDef
        else:
            setattr( __builtins__, objName, objDef )
    return True
```

where objName is a symbol’s name used together with reference to its definition objDef. This technique will allow to use them in all functions of inherited classes like a global objects defined on a module level.

All other DIRAC tools and clients (i.e. RequestManager) are private in RequestTask class and will be imported and instantiated on demand during first call. They are accessible using facades - a proxied public methods, i.e. if you are going to use ReplicaManager just call:

```python
self.replicaManager().someMethod()
```

The main reason for using facades is keeping the memory consumption low as modules are imported and objects are created only on demand. All currently proxied tools are:

- DataLoggingClient – self.dataLoggingClient()
- ReplicaManager – self.replicaManager()
- RequestClient – self.requestClient()
- StorageFactory – self.storageFactory()

The logger message handlers for all levels are also proxied, so you can directly use them in your code, i.e.:

```python
self.info("An info message")
self.debug("This will be shown only in debug")
```

Concerning MonitoringClient (or better known its global instance gMonitor), if someone wants to send some metric over there, she has to put in agent’s code registration of activity and then in a particular task use RequestTask.addMark to save monitoring data. All monitored activities are held in RequestTask.__monitor dict which at the end of processing is returned from RequestTask.__call__. The values are then pushed to then the default callback function defined in RequestAgentBase, from where all of them are exported to gMonitor instance created in agent’s module.

**Request execution**  For handling sub-request one has to register their actions handlers using RequestTask.addOperationAction method. This method checks if handler is defined as a method of inherited class and then puts its definition into internal operation dispatcher dictionary with a key of sub-request’s operation name.

Each operation handler should follow the signature:

```python
def operationName( self, index, requestObj, subRequestAttrs, subRequestFiles )
```
where index is a sub-request counter, requestObj is a RequestContainer instance, subRequestAttrs is a dict with sub-request attributes and subRequestFiles is a dict with files attached to the sub-request. Those handlers should always return S_OK with value of (modified or not) requestObj or S_ERROR with some error message otherwise.

The processing of request os done automatically in RequestTask.__call__, one doesn’t have to worry about changing credentials, looping over sub-requests or request finalizing – only sub-request processing matters in all child classes.

Let’s follow the path of request processing using ‘removal’ request example (there is no difference for ‘registration’ and ‘transfer’ in task execution mode):

Execution of request starts with retrieval of ‘Waiting’ request out of RequestClient using RequestAgent-Base.getRequest.

The request is then serialized to XML string and put into a kwargs dictionary of ProcessPool.createAndQueueTask. Copy of that information is also stored inside agent’s requestHolder dictionary, used to clean up assigned requests during agent’s finalization.

Once there is a free slot in ProcessPool task queue, ProcessTask is enqueued and lands in WorkingProcess instance, which is managing task execution in a separate sub-process. Inside its run method, an instance of ProcessTask is created together with its callable object – our RemovalTask and immediately executed.
Inside `RequestTask.__call__` function request is checked against its owner: if `OwnerDN` and `OwnerGroup` attributes are set, proxy for this user is created and swapped with the default (`DataManager`) proxy. After that processing enters the loop over sub-requests. If particular sub-request has ‘Waiting’ state and its operation matches registered one, the sub-request files and attributes are passed as parameters to the operation handler. After the handler’s execution, the request is again serialized to the XML and updated using `RequestClient`. At this stage finalization of request is also triggered, but only if JobID attribute is different from 0.

The results of `RequestTask.__call__` are stored inside of `ProcessTask` instance, which again is enqueued into results queue of `ProcessPool`. Now processing is shifted to the `ProcessPool` instance again. The results are picked up and sent to the callback function, which is used to copy over the monitored records to the `gMonitor` and to clean up reference of original request from `requestHolder`.

The motivation for adding `requestHolder` dictionary is to have in place a fail-over mechanism for tasks that have never returned any values from their sub-process execution (i.e. sub-process is stuck waiting for a signal, there is a never released lock in the underlying library, the sub-process has been killed by `ProcessPool` or evil and mad person from outside world). In that case, no matter what was the original occasion, the assigned requests are put back to the `RequestClient` at the end of last cycle of the agent, when `RequestAgentBase.finalize` method is called.

**TransferAgent and FTS scheduling** The `TransferAgent` has two modes of operation:

- standalone, when all requests are handled using `ProcessPool` and `TransferTask`,
• scheduling for FTS with fail-back TransferTask functionality.

By default FTS scheduling is disabled and all requests are processed using tasks. The fail-back mechanism is triggered in case that:

• the FTS channels between SourceSE and TargetSE is not defined,
• there is a trouble to define correct replication tree,
• request’s owner is different from DataManager,
• sub-request operation isn’t a ‘replicateAndRegister’.
Execution of the FTS mode is following (see above diagram):

1. The request is checked against its owner, if OwnerDN and OwnerGroup attributes are set, request is passed to the task execution mode.
2. The value Waiting sub-requests operation attributes are compared with the only one handled ‘replicateAndRegister’, if any of sub-request operation is different from ‘replicateAndRegister’, the request is passed to the task execution mode.
3. Request is sent to `TransferAgent.schedule` function, where subsequent methods are called:
   - `TransferTask.checkReadyReplicas`: all files and their registered replicas are compared, if a particular file has been already replicated, its status is set to ‘Done’
   - `TransferAgent.registerFiles`: fail-over registration mechanism for files that have been already replicated (in `FTSSubmitAgent`) but not registered (error in registration in `FTSMonitorAgent`)
   - `TransferAgent.scheduleFiles`: for all ‘Waiting’ files the replication tree is constructed using `StrategyHandler`, when it is ready, `TransferDB.Channels`, `TransfeDB.FileToCat` and `TransferDB.ReplicationTree` records are pushed into `TransferDB` and file status is set to ‘Scheduled’, if for any reason `ReplicationTree` cannot be created, the request is put into task execution mode.
4. When all statuses of files are set to ‘Done’ in previous methods, sub-request status is set to ‘Done’ and the same check is repeated for all sub-requests and request itself. At this stage request is also finalized, if JobID attribute is different from 0.

The request is going to be executed many times, until all replicas are created, but please notice, that FTS scheduling is done only once, during first execution, when there are still ‘Waiting’ files in sub-request. All following operations are only repeating `TransferTask.checkReadyReplicas` and `TransferAgent.registerFiles` calls to update files statuses.

Configuration and installation
## Options common to all agents

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Logging level</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Logging handlers</td>
<td>LogBackends = std</td>
</tr>
<tr>
<td>PollingTime</td>
<td>Time period in seconds for agent’s polling</td>
<td>PollingTime = 60</td>
</tr>
<tr>
<td>ControlDirectory</td>
<td>Control directory location</td>
<td>ControlDirectory</td>
</tr>
<tr>
<td>RequestsPerCycle</td>
<td>Number of requests to process in one agent cycle</td>
<td>RequestPerCycle =</td>
</tr>
<tr>
<td>MinProcess</td>
<td>Minimal number of sub-processes running</td>
<td>MinProcess = 1</td>
</tr>
<tr>
<td>MaxProcess</td>
<td>Maximal number of sub-processes running</td>
<td>MaxProcess = 4</td>
</tr>
<tr>
<td>ProcessPoolQueueSize</td>
<td>Capacity of task queue in ProcessPool</td>
<td>ProcessPoolQueueSize =</td>
</tr>
<tr>
<td>shifterProxy</td>
<td>Default proxy used to process request</td>
<td>shifterProxy = DataProxy</td>
</tr>
<tr>
<td>RequestType</td>
<td>Request type:</td>
<td>RequestType = &lt;requestType&gt;</td>
</tr>
<tr>
<td></td>
<td>• register for RegistrationAgent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• removal for RemovalAgent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• transfer for TransferAgent</td>
<td></td>
</tr>
</tbody>
</table>

### <TaskName> subsection (TaskName = RegistrationTask, RemovalTask, TransferTask)

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLevel</td>
<td>Logging level</td>
<td>LogLevel = INFO</td>
</tr>
<tr>
<td>LogBackends</td>
<td>Logging handlers</td>
<td>LogBackends = std</td>
</tr>
</tbody>
</table>

### TransferAgent specific options

<table>
<thead>
<tr>
<th>Option name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskMode</td>
<td>Flag to disable/enable tasks for processing</td>
<td>TaskMode = True</td>
</tr>
<tr>
<td>FTSMode</td>
<td>Flag to disable/enable FTS scheduling</td>
<td>FTSMode = True</td>
</tr>
<tr>
<td>ThroughputTimescale</td>
<td>Time period used to monitor FTS transfer history</td>
<td>ThroughputTimescale =</td>
</tr>
</tbody>
</table>

### StrategyHandler subsection

<table>
<thead>
<tr>
<th>Option name</th>
<th>Acceptable time shift to start of FTS transfer</th>
<th>HopSigma = 0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchedulingType</td>
<td>Transfer speed calculation:</td>
<td>SchedulingType =</td>
</tr>
<tr>
<td></td>
<td>• number of files per hour (Files)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• amount of data per hour (Throughput)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option name</th>
<th>List of active strategies to use:</th>
<th>ActiveStrategies =</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DynamicThroughput, MinimiseTotalWait, Simple, Swarm</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option name</th>
<th>Percentage limit of success rate in monitored FTS transfers to accept/reject FTS channel from scheduling.</th>
<th>AcceptableFailureRate =</th>
</tr>
</thead>
</table>
**Default options**  Default configuration for all agents:

```plaintext
TransferAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/TransferAgent
    RequestsPerCycle = 10
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = transfer
    shifterProxy = DataManager
    TaskMode = True
    FTSMode = True
    ThroughputTimescale = 3600
    StrategyHandler {
        LogLevel = INFO
        LogBackends = stdout
        HopSigma = 0.0
        SchedulingType = File
        ActiveStrategies = MinimiseTotalWait
        AcceptableFailureRate = 75
    }
    TransferTask {
        LogLevel = INFO
        LogBackends = stdout
    }
}

RegistrationAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/RegistrationAgent
    RequestsPerCycle = 10
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = register
    shifterProxy = DataManager
    RegistrationTask {
        LogLevel = INFO
        LogBackends = stdout
    }
}

RemovalAgent {
    LogLevel = INFO
    LogBackends = stdout
    PollingTime = 60
    ControlDirectory = control/DataManagement/RemovalAgent
    RequestsPerCycle = 50
    MinProcess = 1
    MaxProcess = 4
    ProcessPoolQueueSize = 10
    RequestType = removal
    shifterProxy = DataManager
    RemovalTask {
    }
}
```
Installation procedure

1. On a fresh DIRAC installation procedure is very simple:

   • **RegistrationAgent** and **RemovalAgent**
     Follow the normal installation procedure, but make sure the new configuration sections is in place and updated.

   • **TransferAgent** in mixed mode (*FTSMode* and *TaskMode*)
     This is the default configuration that can be used i.e. in LHCb DIRAC prod system.
     Make sure FTS agents and databases are installed and properly configured (*TransferDB*, *FTSMonitorAgent*, *FTSSubmitAgent* and *FTSCleanupAgent*). Install **TransferAgent**.

   • **TransferAgent** in *TaskMode* only
     This mode should be used in LHCb online version of DIRAC or for VOs without FTS service available.
     Install **TransferAgent**, disable *FTSMode* in its configuration section.

2. Upgrading existing DIRAC installation:

   • stop **TransferAgent**, **ReplicationScheduler**, **RegistrationAgent** and **RemovalAgent** agents
   • update configuration sections by replacing **TransferAgent**, **RemovalAgent** and **RegistrationAgent** and deleting **ReplicationScheduler** fragments
   • start new agents using *runsvctrl u* command

Data Logging System

   author  Berger Corentin
   date    Thu, 20th Aug 2015

Goal

The goal of the DLS is to track all operations made on data files. The DLS is totally transparent and it is flexible.

Many methods make operations about LHCb data files and it is not possible to change all of them. A decorator is the best option for that. Moreover StorageElement and FileCatalog classes use **__getattr__** function to forward calls to the correct storage element or catalog.

Architecture

This is the architecture chosen for the Data Logging System:
The different classes are:

- **DLMethodCall**: this class is used to save the data related to a method’s call. Method’s name is saved. There is an attribute to know the parent method call and the order of call. Thank to this two attributes, it is possible to construct the sequence.
- **DLAction**: an action is a method call on one LFN. DLMethodCall can have many DLAction. For example for the method replicateAndRegister, it is possible to pass a list of LFN to replicate all LFN in the storage element. The status of the operation can be find here to know if the operation was a success or a failure.
- **DLSequence**: a sequence of method call. For example, the first method call is “A” which calls “B” and “C”, this is a sequence. Each DLMethodCall has a reference on its DLSequence and vice versa. DLSequence objects have a stack, we will see after how this stack is used.
- **DLCaller**: a name of a caller. It is needed to know “who”, i.e. which method, which script, called the first decorated method.
- **DLFile**: an LFN
- **DLStorageElement**: an SE
- **DLThreadPool**: this class contains a dictionary whose keys are thread id and whose values are DLSequence objects. This class is necessary because DLSequence object are accessed from different methods for a same thread. The system is thread safe thanks to this class.
Decorator

This section will explain how the decorator of the Data Logging System works.

The DLS decorator is a class and there are some specific features for that type of decorator. Here is the minimum for a decorator written as a class:

```python
def DataLoggingDecorator( function = None, **kwargs ):
    if function:
        return _DataLoggingDecorator( function )
    else:
        def wrapper( function ):
            return _DataLoggingDecorator( function, **kwargs )
        return wrapper

class _DataLoggingDecorator( object ):
    def __init__( self, func , **kwargs ):
        self.func = func
        functools.wraps( func )( self )
    def __get__( self, inst, owner = None ):
        return types.MethodType( self, inst )
    def __call__( self, *args, **kwargs ):
        # do something
        ret = self.func(*args, **kwargs )
        # do something else
        return ret
```

Function `DataLoggingDecorator` allows to pass some arguments to the decorator. A test is made to know if a function is passed. “function” argument is always passed in first position during the decoration, by default it is None. If function is not None, an object of class `_DataLoggingDecorator` is returned. Else, a wrapper is needed because the decoration of the function is like that: `DataLoggingDecorator(**kargs)(func)`. It is not possible to use *args because of the default value of parameter function.

In the `__init__` method of `_DataLoggingDecorator` class the function is saved as an instance attribute because there will be as many instances of `_DataLoggingDecorator` class as decorated methods. The second line in this method is a special feature. It replaces special information like docstring or function name of the object by the special information of the decorated method.

The `__get__` method is called everytime the `_DataLoggingDecorator` class is accessed as a class attribute. “types.MethodType( self, inst )” binds the new method or function to the inst object. For example, we replace the `replicateAndRegister` method in class `DataManger` by the decorated method. This avoids any change of the code for that method.

The `__call__` method is called every time that a decorated method or function is called. This is where all data will be saved, where object will be created, etc. Here is the code of the `__call__` method in the `_DataLoggingDecorator` class:

```python
def __call__( self, *args, **kwargs ):
    """ method called each time when a decorate function is called 
    get information about the function and create a sequence of method calls 
    ""
    result = None
    exception = None
    isCalled = False
    isMethodCallCreate = False
    try:
        self.setCaller()
        localArgsDecorator = self.getAttribute( args[0] )
```

Function `DataLoggingDecorator` allows to pass some arguments to the decorator. A test is made to know if a function is passed. “function” argument is always passed in first position during the decoration, by default it is None. If function is not None, an object of class `_DataLoggingDecorator` is returned. Else, a wrapper is needed because the decoration of the function is like that: `DataLoggingDecorator(**kargs)(func)`. It is not possible to use *args because of the default value of parameter function.

In the `__init__` method of `_DataLoggingDecorator` class the function is saved as an instance attribute because there will be as many instances of `_DataLoggingDecorator` class as decorated methods. The second line in this method is a special feature. It replaces special information like docstring or function name of the object by the special information of the decorated method.

The `__get__` method is called everytime the `_DataLoggingDecorator` class is accessed as a class attribute. “types.MethodType( self, inst )” binds the new method or function to the inst object. For example, we replace the `replicateAndRegister` method in class `DataManger` by the decorated method. This avoids any change of the code for that method.

The `__call__` method is called every time that a decorated method or function is called. This is where all data will be saved, where object will be created, etc. Here is the code of the `__call__` method in the `_DataLoggingDecorator` class:
The different steps are:

- Call of setCaller method: this method get the sequence from the DLThreadPool class. If there is no sequence associated to this thread id, a DLSequence object is created and we get the caller from the stack of calls.

- Call of getAttribute method: in the __init__ method, all parameters passed to the decorator are saved into a dictionary called argsDecorator. Sometimes attributes from objects, which called the decorated method, need to be saved. They will be retrieved here and added in a local argsDecorator dictionary. A local dictionary is needed because of the different calls from different thread. For example when the decorated method is a dynamic method, the real method’s name called is contained into the object as an attribute.

- Call of getMethodCallArg method: this method returns dictionary with all data about method call. For now, just the name of the function is retrieved.

- Call of getActionArg method: this method returns a list of dictionaries with all information that needs to be saved. Dictionaries have 4 keys: “file” which is a LFN, “srcSE” which is a source SE, “targetSE” which is the target SE, and “extra” which is all other information that it is possible to get from parameters passed to the decorated method when it is called.

- Call of createMethodCall method: creates a DLMethodCall with the arguments retrieved before, and adds it into the list of DLMethodCall of the DLSequence object associated to the current thread id.

- Call of initializeAction method: adds as many DLAction object to the actions list of DLMethodCall created before as dictionaries constructed. Set the status of the DLAction objects to “Unknown”.

- Call of the decorated method or function: save the return in a variable.

- Call of setActionStatus method: set the status of actions from the return of the decorated method.

- Call of popMethodCall method: pop of the DLMethodCall object from the stack of DLSequence object associated to the current thread ID, it will be explain a little bit after how the DLSequence class works.

- Call of isSequenceComplete method: to know if the sequence is complete. If it is, call of insertSequence method which passes the sequence to the data base for insertion.
All these method calls are in a block “try except” because if an exception is raise from a method of the decorator, we
do not want it to be raised to the caller. The DLS has its own exceptions. The first one is the “NoLogException” which
is raised when the system does not need to log this method. This is possible because some decorated method are very
generics and dynamic. Because it is generic, sometimes it calls a read method and it is not needed to log it.

The second type of exception that can be raised by the decorator is “DLException”. If we passed in the “except” part,
a DLException is raised with a value of the original exception.

There are two flags in the __call__ method of the _DataLoggingDecorator class:

- isMethodCallCreated: this flag is True if a DLMethodCall has been created and added to the sequence. Like
  that we know if we have to set the status of actions and to pop it from sequence.*
- isCalled: this flag is set to true when the method or function is called. Like that if there is an exception from the
  decorator, we know whether the method has already been called or not.*

The decoration

Because there are many methods to decorate and each has its own prototype, some information must be passed as
arguments of the decorator:

- argsPosition: this is a list of strings and tuples. It is used to know the name of all method’s arguments for their
  extraction:
  - String: use a string when the argument’s name is the name we want to save
  - Tuple: use a tuple when the argument can be passed both in *args and **kwargs, and when the name of the
    argument is not the name we want to save. The structure of tuples shall be (‘nameWanted’, ‘nameInPrototype’).
  - getActionArgsFunction: a string to know which function will be called to extract arguments passed ot the
    decorated method.

There are five special key-words for the argsPosition list because their names can be different for a same parameter in
prototype and because there are interesting arguments:

- dl_file: this is the keyword when the argument of a method is a LFN, a file on which operation will be made.
- dl_srcSE: the keyword when a parameter is a source Storage Element.
- dl_targetSE: the keyword when the parameter is a target Storage Element.
- dl_tuple: key_word when the parameter is an instance of tuple.
- dl_ignore_argument: key_word when the parameter has to be ignored.

These key-words are variables that can be find in DIRAC/DataManagementSystem/Client/DataLogging/DLUtilities
file.

Default case  Here is an example when the prototype of a method is simple, no tuple, no dictionary except for the lfn
parameter:

```python
@DataLoggingDecorator( argsPosition = ['self', dl_files, 'fileName', dl_targetSE, 'guid', 'path', 'checksum'] )
def putAndRegister( self, lfn, fileName, diracSE, guid = None, path = None, checksum = None ):

```

“getActionArgsFunction” is not passed to the decorator here because the default function to extract arguments is the
right one.

Here is another example:

```python
@DataLoggingDecorator( argsPosition = ['self', dl_files, dl_targetSE, ( dl_srcSE, 'sourceSE' ), 'destPath', 'localCache', 'catalog'] )
def replicateAndRegister( self, lfn, destSE, sourceSE = '', destPath = '', localCache = '' , catalog = '' ):

```
In this prototype the argument named sourceSE is a nominal one. For this parameter, the name wanted to be saved is in the variable dl_srcSE and its name in the prototype is sourceSE. The tuple in the argsPosition list for this parameter is (dl_srcSE, ‘sourceSE’).

**Tuple case** Some methods take in parameters a tuple, there is some specifics futures for this. Here is an example of a decoration:

```python
@DataLoggingDecorator( argsPosition = ['self', dl_tuple, 'catalog'], getActionArgsFunction = 'Tuple',
tupleArgsPosition = [dl_files, 'physicalFile', 'fileSize', dl_targetSE, 'fileGuid', 'checksum'] )
def registerFile( self, fileTuple, catalog = '' ):
```

It is necessary to use a special function to extract arguments. This is specify with the parameter getActionArgsFunction = ‘Tuple’.

It also necessary to specify the structure of the tuple with the argument tupleArgsPosition. This one is a list of strings like the argsPosition parameter.

**Execute File Catalog case** This class is special because the decorated method is very generic, it is the w Execute method. This method forwards the call to the right catalog. To know the name of method and their arguments, a dictionary is needed:

```python
dataLoggingMethodsToLog = {
    'addFile' :
        {'argsPosition' : ['self', dl_files],
         'keysToGet' : { 'PFN':'PFN', 'Size':'Size', dl_targetSE:'SE', 'GUID':'GUID', 'Checksum':'Checksum'},
         'setFileStatus' :
             {'argsPosition' : ['self', dl_files],
              'valueName' : 'Status'},
         'addReplica' :
             {'argsPosition' : ['self', dl_files],
              'keysToGet' : { 'PFN':'PFN', dl_targetSE:'SE' } },
         'removeReplica' :
             {'argsPosition' : ['self', dl_files],
              'keysToGet' : { 'PFN':'PFN', dl_targetSE:'SE' } },
         'removeFile' :
             {'argsPosition' : ['self', dl_files] },
         'setReplicaStatus' :
             {'argsPosition' : ['self', dl_files],
              'keysToGet' : { 'PFN':'PFN', dl_targetSE:'SE', 'Status':'Status'} },
         'setReplicaHost' :
             {'argsPosition' : ['self', dl_files],
              'keysToGet' : { 'PFN':'PFN', dl_targetSE:'NewSE', dl_srcSE:'SE', 'Status':'Status'} },
         'setReplicaProblematic' :
             {'argsPosition' : ['self', dl_files],
              'specialFunction' : 'setReplicaProblematic' },
         'createDirectory' :
             {'argsPosition' : ['self', dl_files] },
         'removeDirectory' :
             {'argsPosition' : ['self', dl_files]},
         'changePathMode' :
             {'argsPosition' : ['self', dl_files] },
         'changePathOwner' :
             {'argsPosition' : ['self', dl_files]},
         'changePathGroup' :
             {'argsPosition' : ['self', dl_files] }
}
```
Here the only arguments of all method wanted to be logged are self and dl_files. It is a dictionary in which the keys are lfn and values can be:

- String, in that case the name of the string has to be specified by using the parameter ‘valueName’.
- Dictionary, in that case, the keys needed to get value and the name of the value should be known. This is the aim of the ‘keysToGet’ argument.

Here is how the w_execute method is decorated:

```python
@DataLoggingDecorator( getActionArgsFunction = 'ExecuteFC', attributesToGet = {'methodName' : 'call'}, methods_to_log = dataLoggingMethodsToLog )
def w_execute( self, *parms, **kws):
```

The function to use for extracting arguments is a special one for the FileCatalog class.

**Execute Storage Element case**  It is more or less the same as the File Catalog class, just the function for extracting argument change.

```python
@DataLoggingDecorator( getActionArgsFunction = 'ExecuteSE', attributesToGet = {'methodName' : 'methodName', 'targetSE' : 'name'}, className = 'StorageElement', methods_to_log = dataLoggingMethodsToLog )
def __executeMethod( self, lfn, *args, **kwargs):
```

**Future features**

If there is no case for the method you want to decorate, it is possible to add new features. All arguments passed to the decorator shall be nominated. The function to extract argument shall started by ‘extractArgs’. In the decoration, you just need to pass what is after ‘extractArgs’ in the name of the function to the “getActionArgsFunction” argument.

### 2.7.4 Framework System

**Table of contents**

- Framework System
  - Static Component Monitoring

**Static Component Monitoring**

As of v6r13, DIRAC includes a Component Monitoring system that logs information about what components are being installed and uninstalled on which machines, when and by whom. This information is accessible from both the system administration CLI and the Component History page in the Web Portal.

Using the CLI, it is possible to check the information about installations by using the ‘show installations’ command. This command accepts the following parameters:

- list: Changes the display mode of the results
- current: Show only the components that are still installed
- -n <name>: Show only installations of the component with the given name
- -h <host>: Show only installations in the given host
- -s <system>: Show only installations of components from the given system
- -m <module>: Show only installations of the given module
• `-t <type>`: Show only installations of the given type
• `-itb <date>`: Show installations made before the given date (`dd-mm-yyyy`) 
• `-ita <date>`: Show installations made after the given date (`dd-mm-yyyy`)
• `-utb <date>`: Show installations of components uninstalled before the given date (`dd-mm-yyyy`)
• `-uta <date>`: Show installations of components uninstalled after the given date (`dd-mm-yyyy`)

It is also possible to retrieve the installations history information by using the ‘Component History’ app provided by the Web Portal. The app allows to set a number of filters for the query. It is possible to filter by:

• Name: Actual name which the component/s whose information should be retrieved was installed with
• Host: Machine/s in which to look for installations
• System: System/s to which the components should belong. e.g: Framework, Bookkeeping ...
• Module: Module/s of the components. e.g: SystemAdministrator, BookkeepingManager, ...
• Type: Service, agent, executor, ...
• Date and time: It is possible to select a timespan during which the components should have been installed (it is possible to fill just one of the two available fields)

By pressing the ‘Submit’ button, a list with all the matching results will be shown (or all the possible results if no filters were specified).

### 2.7.5 Request Management System

**Request Management System**

**author** Krzysztof Daniel Ciba `<Krzysztof.Ciba@NOSPAMgmail.com>`

**date** Fri, 28th May 2013

**version** v6r9

**System Overview**

The Request Management System (RMS) is designed for management of simple operations that are performed asynchronously on behalf of users - owners of the requests. The RMS is used for multiple purposes: failure recovery (failover system), data management tasks and some others. It is designed as an open system easily extendible for new types of operations.

**Architecture and functionality**

The core of the Request Management System is a ReqDB database which holds requests records together with all related data: operations that have to be performed in the defined order and possibly a set of files attached. All available and useful queries to the ReqDB are exposed to the request client (ReqClient) through ReqManager service.
Each table in the ReqDB has a corresponding class in the new API fully supporting CRUD operations. Each table column is exposed as a property in the related class.
The record class is instrumented with the internal observer for its children (a Request instance is observing states for all defined Operations, each Operation is observing states of all its Files) and built in state machine, which automatizes state propagation:

- state machine for Request
• state machine for Operation
• state machine for *File*
User is allowed to change only File statuses and in case of specific Operation’s types - Operation statuses, as Request builtin observers will automatically propagate and update statues of parent objects.

CRUD

Create  Construction of a new request is quite simple, one has to create a new Request instance:

```python
>>> from DIRAC.RequestManagementSystem.Client.Request import Request
>>> from DIRAC.RequestManagementSystem.Client.Operation import Operation
>>> from DIRAC.RequestManagementSystem.Client.File import File
>>> request = Request()  # create Request instance
>>> request.RequestName = "foobarbaz"
>>> operation = Operation()  # create new operation
>>> operation.Type = "ReplicateAndRegister"
>>> operation.TargetSE = ["CERN-USER", "PIC-USER"]
```
Invoking `Request.addOperation` method will enqueue operation to the end of operations list in the request. If you need to modify execution order, you can use `Request.insertBefore` or `Request.insertAfter` methods. Please notice there is no limit of `Operations` per `Request`, but it is not recommended to keep over there more than a few. In case of `Files` in a single `Operation` the limit is set to one hundred, which seems to be a reasonable number. In case you think this is not enough (or too much), please patch the code (look for `MAX_FILES` in `Operation` class).

The `Request` and `Operation` classes are behaving as any iterable python object, i.e. you can loop over operations in the request using:

```python
>>> for op in request: print op.Type
ReplicateAndRegister
>>> for opFile in operation: print opFile.LFN, opFile.Status, opFile.Checksum
/foo/bar/baz Waiting 123456
```

Once the request is ready, you can insert it to the `ReqDB`:

```python
>>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
>>> rc = ReqClient() # # create client
>>> rc.putRequest( request ) # # put request to ReqDB
```

**Read**  
Reading request back can be done using two methods defined in the `ReqClient`:

- for reading:
  ```python
  >>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
  >>> rc = ReqClient() # # create client
  >>> rc.peekRequest( "foobaz" ) # # get request from ReqDB for reading
  ```

- for execution (request status on DB side will flip to ‘Assigned’):
  ```python
  >>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
  >>> rc = ReqClient() # # create client
  >>> rc.getRequest( "foobaz" ) # # get request from ReqDB for execution
  ```

If you don’t specify request name in `ReqClient.getRequest` or `ReqClient.peekRequest`, the one with “Waiting” status and the oldest `Request.LastUpdate` value will be chosen.

**Update**  
Updating the request can be done by using methods that modify operation list:
To make those changes persistent you should of course put modified and say dirty request back to the `ReqDB` using `ReqClient.putRequest`.

**Delete**  Nothing special here, just execute `ReqClient.deleteRequest( requestName )` to remove whole request from `ReqDB`.

**Request validation**

The validation of a new Request that is about to enter the system for execution is checked at two levels:

- **low-level**: each property in `Request`, `Operation` and `File` classes is instrumented to check if value provided to its setter has a meaningful type and value:

```
>>> opFile.LFN = 1
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "DIRAC/RequestManagementSystem/private/Record.py", line 52, in __setattr__
    object.__setattr__( self, name, value )
 File "DIRAC/RequestManagementSystem/Client/File.py", line 137, in LFN
    raise TypeError( "LFN has to be a string!" )
TypeError: LFN has to be a string!
>>> operation.SubmitTime = False
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "DIRAC/RequestManagementSystem/private/Record.py", line 52, in __setattr__
    object.__setattr__( self, name, value )
 File "DIRAC/RequestManagementSystem/Client/Operation.py", line 370, in SubmitTime
    raise TypeError( "SubmitTime should be a datetime.datetime!" )
TypeError: SubmitTime should be a datetime.datetime!
```

- **high-level**: additionally there is also a request validator helper class (`RequestValidator` or its global instance `gRequestValidator`) - a gatekeeper checking if request is properly defined. The validator is blocking insertion of a new record to the `ReqDB` in case of missing or malformed attributes and returning `S_ERROR` describing the reason for rejection, i.e.:

```
>>> from DIRAC.RequestManagementSystem.private.RequestValidator import gRequestValidator
>>> from DIRAC.RequestManagementSystem.Client.Request import Request
>>> invalid = Request()
>>> gRequestValidator.validate( invalid )
{'Message': 'RequestName not set', 'OK': False}
>>> invalid.RequestName = "foobarbaz"
>>> gRequestValidator.validate( invalid )
{'Message': 'Operations not present in request 'foobarbaz"", 'OK': False}
>>> from DIRAC.RequestManagementSystem.Client.Operation import Operation
>>> invalid.addOperation( Operation() )
{'OK': True, 'Value': ''}
>>> gRequestValidator.validate( invalid )
{'Message': 'Operation #0 in request 'foobarbaz' hasn't got Type set", 'OK': False}
>>> invalid[0].Type = "ForwardDISET"
```
A word of caution has to be clearly stated over here: both low- and high-level validation is not checking if actual value provided during Request definition makes sense, i.e. if you put to the Operation.TargetSE unknown name of target storage element from the validation point of view your request will be OK, but it will miserably fail during execution.

**Request execution**

The execution of the all possible requests is done in only one agent: RequestExecutingAgent using special set of handlers derived from OperationHandlerBase helper class. What is different from the previous attempt is the way the request is treated: the agent will try to execute request as a whole in one go, while in the old RMS there was several different agents in place, each trying to execute one sub-request type. This approach was a horrible complication for maintain request’s state machine.

The RequestExecutingAgent is using the ProcessPool utility to create slave workers (subprocesses running RequestTask) designated to execute requests read from ReqDB. Each worker is processing request execution using following steps:

- downloading and setting up request’s owner proxy
- loop over waiting operations in the request
- creating on-demand and executing specific operation handler
- if operation status is not updated after treatment inside the handler, worker jumps out the loop otherwise tries to pick up next waiting Operation

Outside the main execution loop worker is checking request status and depending of its value finalizes request and puts it back to the ReqDB.

**Extending**

At the moment of writing following operation types are supported:

- DataManagement (under DMS/Agent/RequestOperations):
  - PhysicalRemoval
  - PutAndRegister
This of course does not cover all possible needs for a specific VO, hence all developers are encouraged to create and keep new operation handlers in VO spin-off projects. Definition of a new operation type should be easy within the context of the new RequestManagementSystem. All you need to do is to put in place operation handler (inherited from OperationHandlerBase) and/or extend RequestValidator to cope with the new type. The handler should be a functor and should override two methods: constructor (__init__) and () operator (__call__):

```python
from DIRAC import gMonitor
from DIRAC.RequestManagementSystem.private.OperationHandlerBase import OperationHandlerBase
import random

class KillParrot( OperationHandlerBase ):
    """ operation handler for 'KillParrot' operation type
    see OperationHandlerBase for list of methods and DIRAC tools exposed
    please notice that all CS options defined for this handler will be exposed there as read-only properties
    ""
    def __init__( self, request = None, csPath = None ):
        """ constructor -- DO NOT CHANGE its arguments list ""
        # # AND ALWAYS call BASE class constructor (or it won't work at all)
        OperationHandlerBase.__init__(self, request, csPath )
        # # put there something more if you need, i.e. gMonitor registration
        gMonitor.registerActivity( "ParrotsDead", ... )
        gMonitor.registerActivity( "ParrotsAlive", ... )

    def __call__( self ):
        """ this has to be defined and should return S_OK/S_ERROR ""
        self.log.info( "log is here" )
        # # and some higher level tools like ReplicaManager
        self.replicaManager().doSomething()
        # # request is there as a member
        self.request
        # # ...as well as Operation with type set to Parrot
        self.operation
        # # do something with parrot
        if random.random() > 0.5:
            self.log.error( "Parrot is still alive" )
            self.operation.Error = "It's only sleeping"
            self.operation.Status = "Failed"
            gMonitor.addMark( "ParrotsAlive", 1 )
        else:
            self.log.info( "Parrot is stone dead" )
            self.operation.Status = "Done"
            gMonitor.addMark( "ParrotsDead", 1 )
```

Once the new handler is ready you should also update config section for the `RequestExecutingAgent`:

```
RequestExecutingAgent {
  OperationHandlers {
    # # Operation.Type
    KillParrot {
      # # add Location for new handler w.r.t. PYTHONPATH settings
      Location = VODIRAC/RequestManagementSystem/Agent/RequestOperations/KillParrot
      ParrotsFoo = True
      ParrotsBaz = 1,2,3
    }
  }
}
```

Please notice that all CS options defined for each handler is exposed in it as read-only property. In the above example `KillParrot` instance will have boolean `ParrotsFoo` set to `True` and `ParrotsBaz` list set to `[1,2,3]`. You can access them in the handler code using `self.ParrotsFoo` and `self.ParrotsBaz`, nothing special, except you can only read their values. Any write attempt will raise `AttributeError` bailing out from request execution chain.

From now on you can put the new request to the `ReqDB`:

```python
>>> request = Request()
>>> operation = Operation()
>>> operation.Type = "KillParrot"
>>> request.addOperation( operation )
>>> reqClient.putRequest( request )
```

and your brand new request with a new operation type would be eventually picked up and executed by the agent.

### Installation

1. Login to host, install `ReqDB`:

   ```
   dirac-install-db ReqDB
   ```

2. Install `ReqProxyHandler`:

   ```
   dirac-install-service RequestManagement/ReqProxy
   ```

Modify CS by adding:

```
Systems {
  RequestManagement {
    URLs {
      ReqProxyURLs = dips://<hostA>:9191/RequestManagement/RequestProxy
    }
  }
}
```

You need at least one of these - they are backing up new requests in case the `ReqManagerHandler` is down. Full description can be found in `ReqManager and ReqProxies`.

3. Install `ReqManagerHandler`:

   ```
   dirac-install-service RequestManagement/ReqManager
   ```
4. Install CleanReqDBAgent:

   dirac-install-agent RequestManagement/CleanReqDBAgent

5. Install RequestExecutingAgent:

   dirac-install-agent RequestManagement/RequestExecutingAgent

If one RequestExecutingAgent is not enough (and this is a working horse replacing DISETFowardingAgent, TransferAgent, RemovalAgent and RegistrationAgent), clone it several times.

1. If VO is using FTS system, install FTSDB:

   dirac-install-db FTSDB

2. Stop DataManagement/TransferDBMonitor service and install FTSManagerHandler:

   runsvctrl d runit/DataManagement/TransferDBMonitor
dirac-install-service DataManagement/FTSManager

3. Configure FTS sites using command dirac-dms-add-ftssite:

   dirac-dms-add-ftssite SITENAME FTSSERVERURL

   In case of LHCb VO:

   dirac-admin-add-ftssite CNAP.it https://fts.cr.cnaf.infn.it:8443/glite-data-transfer-fts/services/FileTransfer
   dirac-admin-add-ftssite RAL.uk https://lcgfts.gridpp.rl.ac.uk:8443/glite-data-transfer-fts/services/FileTransfer

4. Install CleanFTSDBAgent:

   dirac-install-agent DataManagement/CleanFTSDBAgent

5. Install FTSAgent:

   dirac-install-agent DataManagement/FTSAgent

Again, as in case of RequestExecutingAgent, if one instance is not enough, you should probably clone it several times.

7. Once all requests from old version of system are processed, shutdown and remove agents:

   RequestManagement/DISETFowardingAgent
   RequestManagement/RequestCleaningAgent
   DataManagement/TransferAgent
   DataManagement/RegistrationAgent
   DataManagement/RemovalAgent

and services:

   RequestManagement/RequestManager
   RequestManagement/RequestProxy
   DataManagement/TransferDBMonitor

and dbs:
Overview

The **ReqManager** service is a handler for **ReqDB** using DISET protocol. It exposes all CRUD operations on requests (creating, reading, updating and deleting) plus several helper functions like getting requests/operation attributes, exposing some useful information to the web interface/scripts and so on.

The **ReqProxy** is a simple service which start to work only if **ReqManager** is down for some reason and newly created requests cannot be inserted to the **ReqDB**. In such case the **ReqClient** is sending them to one of the **ReqProxies**, where the request is serialized and dumped to the file in the local file system for further processing. A separate background thread in the **ReqProxy** is periodically trying to connect to the **ReqManager**, forwarding saved requests to the place they can be eventually picked up for execution.

Installation

For the proper request processing there should be only one central instance of the **ReqManager** service up and running - preferably close to the hosts on which request processing agents are running.

For the **RequestProxies** situation is quite opposite: they should be installed in the several different places all over the world, preferably close to the biggest CEs or SEs used by the community. Take the LHCb VO as an example, where...
each of Tier1 is running its own ReqProxy. Notice that you have to have at least one ReqProxy running somewhere for normal operation, preferably not sharing the host used by the ReqManager service.

Example configuration:

```plaintext
Systems {
    RequestManagement {
        Services {
            RequestManager {
                LogLevel = INFO
                HandlerPath = DIRAC/RequestManagementSystem/Service/RequestManagerHandler.py
                Port = 9143
                Protocol = dips
                Backend = mysql
                Authorization {
                    Default = authenticated
                }
            }
            RequestProxy {
                LogLevel = INFO
                HandlerPath = DIRAC/RequestManagementSystem/Service/RequestProxyHandler.py
                Port = 9161
                Protocol = dips
                Authorization {
                    Default = authenticated
                }
            }
        }
        URLs {
            ## the only instance of RequestManagerHandler
            RequestManager = dips://<central>:9143/RequestManagement/RequestManager
            ## comma separated list to all RequestProxyHandlers
            RequestProxyURLs = dips://<hostA>:9161/RequestManagement/RequestProxy, dips://<hostB>:9161/RequestProxy
        }
    }
}
```

Don’t forget to put correct FQDNs instead of <central>, <hostA>, <hostB> in above example!

**Upgrading from DIRAC v6r5**

The DIRAC releases prior to v6r6 were using different model for request forwarding: each CE was able to run its own RequestManager (local), preferably with the file backend (which BTW is obsolete now). Requests created by jobs were put to the local RequestDB using local RequestManager URL. A separate locally running ZuziaAgent was picking them up and sending to the central RequestManager service.

For upgrading from the old to the new forwarding model you should follow this procedure:

- install and configure the RequestProxy service in all the places where ZuziaAgent is running, make sure the port used by this service (9161) is visible to the outside world
- stop ZuziaAgent when no more requests are held in the local RequestDB
- stop local RequestManager
- update DIRAC configuration by adding RequestProxy FQDN to the RequestProxyURLs and removing ZuziaAgent and local RequestManager sections

---

1 Zuzia is a little Susan in Polish, the given name of a daughter of DIRAC team developer who left the project a few years ago.
### 2.7.6 Resource Status System

#### Introduction

The Resource Status System, from now RSS, is an autonomous policy system acting as a central status information point for Grid Elements. Due its complexity, it has been split into two major sections:

1. **Status Information Point**
2. **Monitoring System**

On this section, the Status Information Point for grid elements is documented.

Looking backwards, there were two end-points where information regarding Grid Elements statuses was stored. The first one, the Configuration System (CS) stored the Storage Element (SE) status information mixed with static information like the SE description among other things. The second one, the Workload Management System (WMS) (WMS) stored the Site status information (more specifically, on a dedicated table on JobDB called SiteMask).

The case of the SEs was particularly inconvenient due to the changing nature of a SE status stored on a almost dynamic container as it is the CS. In spite of being a working solution, it was pointing out the bounds of the system. The CS had not been designed for such purpose.

With that problem in hand, it was very easy to abstract it and include the site status information stored on the SiteMask. And that was just the beginning... Nowadays the DIRAC interware offers a formal description to describe grid elements and their status information using two complementary systems:

- CS, which holds the descriptions and hierarchy relationships (no need to say they are static)
- RSS, which takes care of the status information.

You can find the details on the RFC5.

#### Element

An Element in the RSS world represents a Grid Element as described on the RFC5. It can be any of the following:

- Node
Elements are the information unit used on RSS. Everything is an Element, and all are treated equally, simplifying the design and reducing the complexity of the system. If all are treated equally, the reader may be wondering why three flavors instead of just an Element type. The answer for that question is simply to keep them separated. On the RSS they are treated equally, but in Real they have very different significance. Marking as unusable a Site or a CE on the RSS requires the same single and unique operation. However, the consequences of marking as unusable a Site instead of one if its CEs by mistake are not negligible. So, you can also add “safety” as a secondary reason.

**ElementType**

The Grid topology is not part of the RSS itself, but is worth mentioning the relations underneath to have a full picture. The Grid is composed by a “un”certain number of Sites. Those sites are registered with their respective descriptions on the DIRAC CS as follows:

```
/Resources/Sites
  /CERN.ch
  ...  
  /IN2P3.fr
  /Domains = EGI, LCG
  /ContactEmail = someone@somewhere
  /MoreDetails = blah, blah, blah
  /Computing
  /...
  /Storage
  /...
  /PIC.es
  ...  
```

Each Site can have any number of Resources, grouped into categories. In terms of RSS, those categories are **ElementType**. For the Resources Element, we have the following Element Types:

- ComputingElement
- StorageElement
- ...

And if we take a look to the ComputingElement Resources, we can see the pattern happening again.

```
.../Computing/some.cream.ce
  /CETYPE = CREAM
  /Host = some.cream.ce
  /Queues
    /cream-sge-long
      /Communities = VO1, VO2
      /Domains = Grid1, Grid2
      /MaxCPUTime =
      /SI00 =
      /MaxWaitingJobs =
      /MaxTotalJobs =
      /OutputURL =
      ...
```

Each CE Resource has any number of Nodes, in this case of the ElementType Queue.

The list of ElementTypes per Element may vary depending on the CS/Resources section!
Each Element has an associated State, which is what will be used to mark the Element as usable or not. In principle, looks like a binary flag would solve the problem, either ON or OFF. On practice, a fine-grained granularity for the States has been implemented.

There are four major states, plus two corner-cases states which do not apply on the basic implementation:

- **Active**, it is 100% operative.
- **Degraded**, its performance is affected by X reason, but still usable.
- **Probing**, is recovering from a Banned period, but still has not been certified to be Ok.
- **Banned**, is basically down.

**StatusType**

It may happen that an Element requires more than one Status. A very clear example are the StorageElement Resources, which require several Statuses in order to specify the different data accesses (ReadAccess, WriteAccess, etc...).

By default, every Element has only one StatusType - “all”. However, this can be modified on the CS to have as many as needed. Please, take a look to RSS Configuration for further (setup) details.

**Ownership**

RSS includes and extends the concept of ownership, already in place for the mentioned SiteMask. It makes use of tokens, which are simply a tuple composed with the **username** and a **timestamp**.

They have two main functions:

- identify who has put his / her hands on that particular Element.
- bind the Status of that Element to the user.

By default, RSS is the owner of all Elements with an ALWAYS timestamp and username **rs_svc**. However, if there is a manual - “human” - intervention, the Element will get a 1-day-valid token for that user, and it will be recorded like that.

The second function is new in what respects the SiteMask implementation, but its purpose is not part of the basic usage of RSS. Please continue reading here: Ownership II.
Parenthood

As it was already explained on *ElementType*, Elements of different flavors are linked as stated on the CS. As it can be incredibly tedious getting those relations constantly, the most common operations have been instrumented inside the Resources() helper.

**Resources() Helper**

Warning: The Resources() Helper still needs to be developed.

Database schema

The database used for the basic operations is *ResourceStatusDB* and consists on three sets of identical tables, one for *Site*, another for *Resource* and the last one for *Node* Elements (as explained on *Element*).

On each set there is a main table, called *<element>Status* (replace *<element>* with Site, Resource or Node), which contains all status information regarding that Elements family. The Status tables are enough to start running the RSS. However, if we need to keep track of the History of our Elements, the next two tables come into scene: *<element>Log* and *<element>History*. 
Every change on Status is automatically recorded on Log and kept for a configurable amount of time. The last table, History summarizes Log table, removing consecutive entries where the Status for a given tuple (ElementName, StatusType) has not changed.

**Note:** There are no Foreign Keys on the ResourceStatusDB tables.

**Synchronizer**

The Synchronizer is the code linking the CS and the RSS (in that direction, not viceversa!). Every change on the CS in terms of Element addition or deletion is reflected on the RSS. With other words, it populates the ResourceStatusDB
Status tables with the information in the CS. In order to do so, it makes use of the Resources() Helper, which is aware of the CS structure. Every time there is an update of the CS information, this object will look for discrepancies between the database and what is on the CS and fix them consequently.

Architecture

DIRAC in general has a client-server architecture, but (almost) every system has a different approach to that model. That architecture has clients, servers and databases. In fact, the client name can be misleading sometimes, but that is a different business.

The approach used by RSS is to give full access to the data through the client. In practice this means your life is easy if you do not care about details, and just want your thing working quickly. As the image shows, the client acts as a big black box. The idea is to ALWAYS access the RSS databases through the clients, independently of your condition: human being, DIRAC agent, etc...
Most of the users do not care about how data is accessed, making the client good enough for them. If you are one of those that do not like black boxes, here is what happens behind the scenes: the client establishes a connection - either a MySQL connection with the database or a RPC procedure with the server. By default, it connects through the server.

**Note:** We encourage you to use the client, but if you prefer to connect directly to the db or the server, you can do it as well.

The fact of connecting either to the server or the database triggers the following question: how do we connect to the server and the database without fattening our code every time we add something to the API?

Easy, we just expose the same methods the server and db wrapper exposing. That keeps the interfaces clean and tidied. However, every time a new functionality is added to the system, a new set of methods must be written in the db & service modules... or maybe not! Database and service are provided by 4 low level methods:

- `insert`
- `select`
- `update`
- `delete`

plus three little bit smarter methods making use of the first four:

- `addOrModify`
- `addIfNotThere`
- `modify`

The first four methods are the abstraction of the MySQL statements `INSERT`, `SELECT`, `UPDATE` and `DELETE`. The last three include few extras:

- log the status to the `<element>Log tables`
- `addOrModify` behaves as ‘`INSERT ... ON DUPLICATE KEY UPDATE’`
- `addIfNotThere` is an insert logging to the `<element>Log tables.`
- `modify` is an update logging to the `<element>Log tables.`

**Note:** In most cases, you will only need the methods `addOrModify`, `modify` and `select`.

**Installation**

**Setup for BASIC USAGE**

This page describes the basic steps to install, configure, activate and start using the ResourceStatus system of DIRAC. **WARNING:** If you have doubts about the success of any step, DO NOT ACTIVATE RSS. **WARNING:** REPORT FIRST to the DIRAC FORUM!

**CS Configuration**

The configuration for RSS sits under the following path on the CS following the usual `/Operations` section convention:
DIRAC Documentation, Release integration

Please, make sure you have the following schema:

```plaintext
/Operations/Defaults/ResourceStatus
/Config
    State = InActive
    Cache = 300
    CacheHistory = 24
    RecordLogs = Active
/StatusTypes
    default = all
    StorageElement = ReadAccess, WriteAccess, CheckAccess, RemoveAccess
```

For a more detailed explanation, take a look to the official documentation: *RSS Configuration*.

**Fresh DB**

Needs a fresh DB installation. *ResourceStatusDB* and *ResourceManagementDB* are needed. Information on former ResourceStatusDB can be discarded. Delete the old database tables. If there is no old database, just install a new one, either using the dirac-admin-sysadmin-cli or directly from the machine as follows:

```bash
$ dirac-install-db ResourceStatusDB
$ dirac-install-db ResourceManagementDB
```

**Generate DB tables**

The DB tables will be created when the services are started for the first time.

**Run service(s)**

RSS - basic - needs the following services to be up and running: ResourceStatusSystem/ResourceStatus, ResourceStatusSystem/ResourceManagement please install them using the dirac-admin-sysadmin-cli command, and make sure it is running:

```bash
install service ResourceStatusSystem ResourceManagement
install service ResourceStatusSystem ResourceStatus
```

In case of any errors, check that you have the information about DataBase ‘Host’ in the configuration file.

The host(s) running the RSS services or agents need the ‘SiteManager’ property.

**Populate tables**

First check that your user has ‘SiteManager’ privilege, otherwise it will be “Unauthorized query” error. Let’s do it one by one to make it easier:

```bash
$ dirac-rss-sync --element Site -o LogLevel=VERBOSE
$ dirac-rss-sync --element Resource -o LogLevel=VERBOSE
$ dirac-rss-sync --element Node -o LogLevel=VERBOSE
```
Initialize Statuses for StorageElements

Copy over the values that we had on the CS for the StorageElements:

```bash
$ dirac-rss-sync --init -o LogLevel=VERBOSE
```

**WARNING**: If the StorageElement does not have a particular StatusType declared on the CS, this script will set it to Banned. If that happens, you will have to issue the `dirac-rss-status` script over the elements that need to be fixed.

Set statuses by HAND

In case you entered the **WARNING**! on point 4, you may need to identify the status of your StorageElements. Try to detect the Banned SEs using the following:

```bash
$ dirac-rss-list-status --element Resource --elementType StorageElement --status Banned
```

If there is any SE to be modified, you can do it as follows:

```bash
$ dirac-rss-set-status --element Resource --name CERN-USER --statusType ReadAccess --status Active --reason "Why not?"
```

# This matches all StatusTypes

```bash
$ dirac-rss-set-status --element Resource --name CERN-USER --status Active --reason "Why not?"
```

Activate RSS

If you did not see any problem, activate RSS by setting the CS option:

```bash
/Operations/Defaults/ResourceStatus/Config/State = Active
```

Agents

There are few agents that are required:

- SummarizeLogsAgent

Please, install them and make sure they are up and running. Old agents, like the `InspectorAgent(s)` can be safely removed.

RSS Configuration

The basic configuration for the RSS is minimal, and must be placed under the Operations section, preferably on Defaults subsection.

```bash
/Operations/Defaults/ResourceStatus
    /Config
    State = Active
    Cache = 720
    FromAddress = email@address
    /StatusTypes
    default = all
    StorageElement = ReadAccess,WriteAccess,CheckAccess,RemoveAccess
```

2.7. DIRAC Systems
Config section

This section is all you need to get the RSS working. The parameters are the following:

State | Active || InActive ( default if not specified ) > is the flag used on the ResourceStatus helper to switch between CS and RSS. If Active, RSS is used.

Cache | < <int> || 300 ( default if not specified ) > [ seconds ] sets the lifetime for the cached information on RSSCache.

FromAddress | < <string> || ( default dirac mail address ) > email used to send the emails from ( sometimes a valid email address is needed ).

StatusTypes | if a ElementType has more than one StatusType ( aka StorageElement ), we have to specify them here. Otherwise, “all” is taken as StatusType.

Usage

Table of contents

- Usage
  - scripts
    * dirac-rss-list-status
    * dirac-rss-set-status
  - interactive shell
    * Helper

scripts

There are two main scripts to get and set statuses on RSS:

- dirac-rss-list-status
- dirac-rss-set-status

dirac-rss-list-status | This command can be issued by everyone in possession of a valid proxy.

dirac-rss-set-status | This command CANNOT be issued by everyone. You need the SiteManager property to use it. Appart from setting a new status, it will set the token owner for the elements modified to the owner of the proxy used for a duration of 24 hours.

interactive shell

This is a quick reference of the basic usage of RSS from the python interactive shell.

There are two main components that can be used to extract information:

- the client : ResourceStatusSystem
- the helper : SiteStatus, ResourceStatus, NodeStatus

The second is a simplification of the client with an internal cache. Unless you want to access not-only status information, please use the second. Nevertheless, bear in mind that both require a valid proxy.
Let’s get some statuses.

```python
helper = ResourceStatus()

# Request all status types of CERN-USER SE
helper.getStorageElementStatus( 'CERN-USER' )[ 'Value' ]
{'CERN-USER': {'ReadAccess': 'Active', 'RemoveAccess': 'Active', 'WriteAccess': 'Active', 'CheckAccess': 'Active'}}

# Request ReadAccess status type of CERN-USER SE
helper.getStorageElementStatus( 'CERN-USER', statusType = 'ReadAccess' )[ 'Value' ]
{'CERN-USER': {'ReadAccess': 'Active'}}

# Request ReadAccess & WriteAccess status types of CERN-USER SE
helper.getStorageElementStatus( 'CERN-USER', statusType = [ 'ReadAccess', 'WriteAccess' ] )[ 'Value' ]
{'CERN-USER': {'ReadAccess': 'Active', 'WriteAccess': 'Active'}}

# Request ReadAccess status type of CERN-USER and PIC-USER SEs
helper.getStorageElementStatus( [ 'CERN-USER', 'PIC-USER' ], statusType = 'ReadAccess' )[ 'Value' ]
{'CERN-USER': {'ReadAccess': 'Active'}, 'PIC-USER': {'ReadAccess': 'Active'}}

# Request unknown status type for PIC-USER SE
helper.getStorageElementStatus( 'PIC-USER', statusType = 'UnknownAccess' )
Cache misses: [('PIC-USER', 'UnknownAccess')]
{'Message': "Cache misses: [('PIC-USER', 'UnknownAccess')]", 'OK': False}

# Request unknown and a valid status type for PIC-USER SE
helper.getStorageElementStatus( 'PIC-USER', statusType = [ 'UnknownAccess', 'ReadAccess' ] )
Cache misses: [('PIC-USER', 'UnknownAccess')]
{'Message': "Cache misses: [('PIC-USER', 'UnknownAccess')]", 'OK': False}
```

Similarly, let’s set some statuses.

```python
helper = ResourceStatus()

# Are you sure you have a proxy with SiteManager property ? If not, this is what you will see.
helper.setStorageElementStatus( 'PIC-USER', 'ReadAccess', 'Active', reason = 'test' )[ 'Message' ]
'Unautorized query'

# Let's try again with the right proxy
_= helper.setStorageElementStatus( 'PIC-USER', 'ReadAccess', 'Bad', reason = 'test' )
helper.getStorageElementStatus( 'PIC-USER', 'ReadAccess' )
{'OK': True, 'Value': {'PIC-USER': {'ReadAccess': 'Bad'}}}

# Or banning all SE. For the time being, we have to do it one by one!
helper.setStorageElementStatus( 'PIC-USER', [ 'ReadAccess', 'WriteAccess' ], 'Bad', reason = 'test' )
False
```

**Monitoring**

The monitoring part is the other half of RSS, and where most of the complexity lies. This part handles the automatic status assessment for any Element registered on RSS.
State Machine

The state machine forces the transitions between valid states: Unknown, Active, Bad, Probing, Banned and Error. In principle, the first and last states of the list should not be visible. They are used to manage corner cases and crashes. The only restriction is the following:

*any transition from Banned to Unknown, Active or Bad will be forced to go to Probing first*

The idea is that after a downtime, we check the health of the Element before setting it as Active.

**Note:** The order in which statuses have been introduced is not trivial. Active is more restrictive than Unknown, which is less restrictive than Bad, and so on. This detail is crucial on *Policy Decision Point*.

Element Inspector Agents

There is one InspectorAgent per family of elements: Site, Resource and Node. They run frequently and get from the DB the elements that have not been checked recently. With other words, they take elements following:

\[
\text{LastCheckTime} + \text{lifetime( Status )} < \text{now()}
\]

where \(\text{LastCheckTime}\) is a timestamp column of the tables storing the element statuses and \(\text{lifetime( Status )}\) corresponds to the next table. The healthier the element is, the less often is checked.

<table>
<thead>
<tr>
<th>Status</th>
<th>Lifetime ( min )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>60</td>
</tr>
<tr>
<td>Degraded</td>
<td>30</td>
</tr>
<tr>
<td>Probing</td>
<td>30</td>
</tr>
<tr>
<td>Banned</td>
<td>30</td>
</tr>
<tr>
<td>Unknown</td>
<td>15</td>
</tr>
<tr>
<td>Error</td>
<td>15</td>
</tr>
</tbody>
</table>

When checked, it is passed as a dictionary that looks like the following to the Policy System. This dictionary is the python representation of one row in the table.

```python
decisionParams = {
    'element' : 'Resource',
    'elementType' : 'CE',
    'name' : 'some.ce',
    'statusType' : 'all',
    'status' : 'Active',
    'reason' : 'This is the AlwaysActive policy ###',
    'dateEffective' : datetime.datetime( ... ),
    'lastCheckTime' : datetime.datetime( ... ),
    'tokenOwner' : 'rs_svc',
    'tokenExpiration' : datetime.datetime( ... )
}
```

Policy System

The Policy System is comprised by several modules (listed by order of execution).

- Policy Enforcement Point (PEP)
- Policy Decision Point (PDP)
- Info Getter (IG)
• Policy Caller (PC)

Policy Enforcement Point The Policy Enforcement Point is the main object, which will orchestrate the status assessment and the actions taken. In order to do so, it will make use of the Policy Decision Point to get the results of the policies run, and the actions that must be taken. These are returned on a dictionary, resDecisions (which will be returned to the Element Inspector Agent as well).

Note: running a policy does not trigger any update on the database. Are the actions which perform changes on the database, send alerts, etc.

Let’s understand it with a sequence diagram:

Firstly, the Element Inspector Agent calls the PEP with a dictionary like the one shown above – decisionParams. The PEP will setup the PDP, which among other things will sanitize the input. Once done, the PDP will take a decision with the decision parameters provided. Its reply is a dictionary consisting on three key-value pairs, that looks like the one below, resDecisions. Once resDecisions is known, the PEP runs the actions suggested by the PDP and exits.

```python
resDecisions = {
    'decisionParams': decisionParams,
    'policyCombinedResult': {
        'Status': 'Active',
        'Reason': 'A Policy that always returns Active and
        'PolicyAction': [
            {'policyActionName1', 'policyActionType1'},
        ],
    'singlePolicyResults': [
        {'Status': 'Active',
        'Reason': 'blah',
        'Policy': {
            'name': 'AlwaysActiveForResource',
            'type': 'AlwaysActive',
            'module': 'AlwaysActivePolicy',
            'description': 'A Policy that always returns Active',
            'command': None,
            'args': {}},
    }
}
```
Complex? Not really, just big (can be quite). What does it mean? It has three keys:

- **decisionParams**: input dictionary received from the Element Inspector Agent.
- **policyCombinedResult**: dictionary with the computed result of the policies - **Status** and **Reason** - and a list of actions to be run - **PolicyAction**.
- **singlePolicyResults**: list with dictionaries, one per policy run. Explained on **Policy Caller**

The PEP will iterate over the tuples in the PolicyAction value and executing the actions defined there.

**Actions**  
DIRAC.RSS has the following actions:

- **EmailAction**: sends an email notification
- **SMSAction**: sends a sms notification (not certified yet).
- **LogStatusAction**: updates the <element>-Status table with the new computed status
- **LogPolicyResultAction**: updates the PolicyResult table with the results of the policies in singlePolicyResults.

The last action should always run, otherwise there is no way to track what happened with the policies execution. The others, depend on what we want to achieve. At the moment, there is no user case where LogStatusAction is not run.

**Policy Decision Point**  
The Policy Decision Point is the instance that will collect all results from the policies and decide what to do. However, it will not do anything, that is the task of the PEP. You can see the PDP as a advisory service.

Its flow is depicted on the following sequence diagram:
Firstly it sanitizes the input parameters `decisionParams` into `decisionParams'`

```python
# cannot name decisionParams' ( is not a valid python name ), decisionParams2 instead
decisionParams2 = {
    'element': decisionParams.get('element', None),
    'elementType': decisionParams.get('elementType', None),
    'name': decisionParams.get('name', None),
    'statusType': decisionParams.get('statusType', None),
    'status': decisionParams.get('status', None),
    'reason': decisionParams.get('reason', None),
    'tokenOwner': decisionParams.get('tokenOwner', None),
    '#dateEffective': datetime.datetime(...),
    '#lastCheckTime': datetime.datetime(...),
    '#tokenExpiration': datetime.datetime(...),
    'active': True
}
```

**Note:** the timestamps are not taken into account on decisionParams'. However, a new parameter is added `active`. Its meaning will be explained on **Info Getter**.

which will be used internally by the PDP instead of the input dictionary. It contacts the Info Getter to find the policies that match the decision parameters (`decisionParams'`). This means, `decisionParams'` is compared with all the policies metadata defined on the CS. Once PDP knows which policies apply, it runs them, obtaining a list of dictionaries `singlePolicyResults`. Each dictionary contains the `Status` and `Reason` proposed by a particular policy.

```python
singlePolicyResults = [ { 'Status': 'Active', 'Reason': 'reasonActive' }, { 'Status': 'Bad', 'Reason': 'reasonBad' }, { 'Status': 'Bad', 'Reason': 'reasonBad2' } ]
```

Knowing all the proposed statuses by the policies, they are sorted by status and picked the most restrictive ones (as explained on **State Machine**. In this sense, the most restrictive status is Error). As a result of the policies result computing, we have a dictionary `combinedPolicyResults` with the most restrictive status as `Status` and the concatenation of reasons paired with that most restrictive status separated by `###` as `Reason`.

```python
combinedPolicyResults = { 'Status': 'Bad', 'Reason': 'reasonBad ### reasonBad2' }
```

More or less the same principle applies to get the actions that apply. The only difference is that the single policy results are taken into account (perhaps, no matter what we want to send an alert based on a policy), as well as the combined results (actions triggered based on the proposed final result). Once the PDP has a list of action tuples (actionName, actionType), builds the `resDecisions` dictionary and returns it to the PEP.

**Info Getter** Info getter is the piece of code that decides which policies and actions match. It reads from the CS (`/Operation/ResourceStatus/Policies`) and gets a dictionary per policy defined there. The matching algorithm works as follows:

```python
for key in decisionParams:
    # first case
    if not key in policyParams:
        # if the policy has not defined the parameter `key`, it is ignored for the matching
        continue

    # second case
    if decisionParams[ key ] is None:
        # None is assumed to be a wildcard (*)
        continue

    # Convert to list before intersection ( the algorithm is slightly different at this
# point, but does not really matter in the InfoGetter explanation).

dParameter = [ decisionParams[ key ] ]

# third case
# At this point, we know that 'key' is a parameter in decisionParams and policyParams.
# if dParameter is not included in the list policyParams[ 'key' ], we have a False
# match.

if not set( dParameter ).intersection( set( policyParams[ key ] ) ) :
    return False

return True

Or with other words:

- a policy with empty definition in the CS, will match any resource (first case).
- a decisionParams dictionary with values None, will match any policy (second case). However, this will never happen if called from ElementInspectorAgent. It is enforced to not have None values.
- otherwise, we are on third case.

Do not forget about the Active parameter forced on the PDP! It is very handy if we want to disable a policy on the CS completely without having to delete it. We just need to set active = False. As active is set by default as True in the PDP, we will have a False match.

For the actions, same principle applies. The difference are the input and reference dictionaries. In this case, for every action we compare all dictionaries in singlePolicyResults, plus combinedPolicyResult plus decisionParams. This allows us to trigger actions based on the global result, on a particular policy result, on a kind of element, etc..

Policy Caller  Given a policyDict, the Policy Caller imports the policy <Extensions>DIRAC.ResourceStatusSystem.Policy.<policyDict[ 'module' ]> and run it. In case there is a command specified, it will be run using policyDict[ ‘args’ ] and decisionParams' as inputs.

```python
policyDict = {
    'name' : 'AlwaysActiveResource',
    'type' : 'AlwaysActive',
    'args' : None,
    'description' : 'This is the AlwaysActive policy',
    'module' : 'AlwaysActivePolicy',
    'command' : None
}
```

Policy  A Policy is a simple piece of code which returns a dictionary like:

```python
policyRes = { 'Status' : 'Active', 'Reason' : 'This is the AlwaysActive policy' }
```

If defined, it evaluates a command firstly, which will fetch information from the database cache of fresh from the source of information. To change the behavior, add to policyDict the key-value pair (this is done on the code: DIRAC.ResourceStatusSystem.Policy.Configurations).

- ‘args’: { ‘onlyCache’ : True }

Command  Commands are the pieces of code in charge of getting the information from different information sources or caches in bulk queries, getting it individually and storing it.

Commands are used with two purposes:
• Request with bulk queries the information to fill the cache tables (commands issued by an agent called CacheFeederAgent). This is the master mode.

• Provide policies with the information concerning the element they are evaluating.

Their basic usage is:

```python
argsDict = { .. }
# this command will query XYZ cache in RSS to get information about a particular element,
# if there is nothing, it will query the original source of information
CommandXYZ( argsDict ).doCommand()

# this command will ONLY query XYZ cache about a particular element. This is the suggested
# operation mode for policies to avoid hammering sources of information
argsDict[ 'onlyCache' ] = True
CommandXYZ( argsDict ).doCommand()

# However, if we want to fill the caches, we use the master mode of the Command.
# It will get the information and store it where it belongs
C = CommandXYZ()
C.masterMode = True
C.doCommand()
```

Ownership II

So far, so good. But what if we want to take the control out from RSS for a given element. This is done making use of the token ownership. By default, every element belongs to RSS (token rs_svc). However, we can request the token for a set of elements (by default, it is one day). During that period, RSS will not apply any policy on them. If by the end of the 24 hours the owner of the token has not extended its duration, RSS will gain again control of the element.

Advanced Configuration

The full RSS configuration comprises 4 main sections

- **Config**
- **Policies**
- **PolicyActions**
- **Notification**

Config

Already described in config section.

Policies

This section describes the policies and the conditions to match elements.

```
/Operations/Defaults/ResourceStatus
  /Policies
    /PolicyName
      policyType = policyType
      doNotCombineResult = something
```
This is the complete definition of a policy. Let’s go one by one.

- **PolicyName**: this must be a human readable name explaining what the policy is doing (mandatory).

- **policyType**: is the name of the policy we want to run as defined in DIRAC.ResourceStatusSystem.Policy.Configurations (mandatory).

- **doNotCombineResult**: if this option is present, the status will not be merged with the rest of statuses (but actions on this policy will apply).

- **matchParams**: is the dictionary containing the policy metadata used by *Info Getter* to match policies. Any of them can be a CSV.

**Note**: Remember, declare ONLY the parameters in match params that want to be taken into account.

There is one caveat. If we want to match the following SEs: CERN-USER for ReadAccess and PIC-USER for WriteAccess, we cannot define the following matchParams:

```plaintext
.../matchParams
  element = Resource
  elementType = StorageElement
  name = CERN-USER, PIC-USER
  statusType = ReadAccess, WriteAccess
```

**Warning**: This setting will match the cartesian product of name x statusType. We will match CERN-USER for WriteAccess and PIC-USER for ReadAccess as well. We will need two separate policies.

**PolicyActions**

It applies the same idea as in *Policies*, but the number of options is larger.
Note: Mind the upper / lower case ( to be fixed )

- PolicyActionName: must be a human readable name explaining what the action will do (mandatory).
- actionType: is one of the following actions (mandatory).
- notificationGroups: if required by the actionType, one of Notification.
- matchParams: as explained in Policies.
- combinedResult: this is the computed final result after merging the single policy results.
- policyResults: allows to trigger an action based on a single policy result, where policyName follows Policies.

Notification

This section defines the notification groups (right now, only for EmailAction).

/Operations/Defaults/ResourceStatus
   /Notification
      /NotificationGroupName
         users = email@address, email@address

- NotificationGroupName: human readable of what the group represents
- users: CSV with email addresses

Advanced Usage

Table of contents

- Advanced Usage
  - scripts

scripts

Will come soon.

2.7.7 Storage Management System

Table of contents

- Storage Management System
2.7.8 Transformation System

The Transformation System (TS) is used to automatise common tasks related to production activities. Just to make some basic examples, the TS can handle the generation of Simulation jobs, or Data Re-processing jobs as soon as a ‘pre-defined’ data-set is available, or Data Replication to ‘pre-defined’ SE destinations as soon as the first replica is registered in the Catalog.

The lingo used here needs a little explanation: throughout this document the terms “transformation” and “production” are often used to mean the same thing:

- A “production” is a transformation managed by the TS that is a “Data Processing” transformation (e.g. Simulation, Merge, DataReconstruction...). A Production ends up creating jobs in the WMS.
- A “Data Manipulation” transformation replicates, or remove, data from storage elements. A “Data Manipulation” transformation ends up creating requests in the RMS (Request Management System).

For each high-level production task, the production manager creates a transformation. Each transformation can have different parameters. The main parameters of a Transformation are the following:

- Type (e.g. Simulation, DataProcessing, Removal, Replication)
- Plugin (Standard, BySize, etc.)
- The possibility of having Input Files.

Within the TS a user can (for example):

- Generate several identical tasks, differing by few parameters (e.g. Input Files list)
- Extend the number of tasks
- have one single high-level object (the Transformation) associated to a given production for global monitoring

Disadvantages:

- For very large installations, the submission may be perceived as slow, since there is no use (not yet) of Parametric jobs.

Several improvements have been made in the TS to handle scalability, and extensibility issues. While the system structure remains intact, “tricks” like threading and caching have been extensively applied.

It’s not possible to use ISB (Input Sandbox) to ship local files as for ‘normal’ Jobs (this should not be considered, anyway, a disadvantage).
Architecture

The TS is a standard DIRAC system, and therefore it is composed by components in the following categories: Services, DBs, Agents. A technical drawing explaining the interactions between the various components follow.

- **Services**
  - TransformationManagerHandler: DISET request handler base class for the TransformationDB

- **DB**
  - TransformationDB: it's used to collect and serve the necessary information in order to automate the task of job preparation for high level transformations. This class is typically used as a base class for more specific data processing databases. Here below the DB tables:
Note that since version v6r10, there are important changes in the TransformationDB, as explained in the release notes (for example the Replicas table can be removed). Also, it is highly suggested to move to InnoDB. For new installations, all these improvements will be installed automatically.

**Agents**

- **TransformationAgent**: it processes transformations found in the TransformationDB and creates the associated tasks, by connecting input files with tasks given a plugin. It’s not useful for MCSimulation type.

- **WorkflowTaskAgent**: it takes workflow tasks created in the TransformationDB and submits to the WMS. Since version v6r13 there are some new capabilities in the form of TaskManager plugins.

- **RequestTaskAgent**: it takes request tasks created in the TransformationDB and submits to the RMS. Both RequestTaskAgent and WorkflowTaskAgent inherits from the same agent, “TaskManagerAgentBase”, whose code contains large part of the logic that will be executed. But, TaskManagerAgentBase should not be run standalone.

- **MCExtensionAgent**: it extends the number of tasks given the Transformation definition. To work it needs to know how many events each production will need, and how many events each job will produce. It is only used for ‘MCSimulation’ type.

- **TransformationCleaningAgent**: it cleans up the finalised Transformations.

- **InputDataAgent**: it updates the transformation files of active Transformations given an InputDataQuery fetched from the Transformation Service.

- **ValidateOutputDataAgent**: it runs few integrity checks prior to finalise a Production.

The complete list can be found in the DIRAC project GitHub repository.

**Clients**

- **TaskManager**: it contains WorkflowsTasks and RequestTasks modules, for managing jobs and requests tasks, i.e. it contains classes wrapping the logic of how to ‘transform’ a Task in a job/request. WorkflowTaskAgent uses WorkflowTasks, RequestTaskAgent uses RequestTasks.

- **TransformationClient**: class that contains client access to the transformation DB handler (main client to the service/DB). It exposes the functionalities available in the DIRAC/TransformationHandler. This inherits the DIRAC base Client for direct execution of server functionality.

- **Transformation**: it wraps some functionalities mostly to use the ‘TransformationClient’ client.
Configuration

• Operations
  – In the Operations/[VO]/Transformations section, Transformation Types must be added
  – By default, the WorkflowTaskAgent will treat all the DataProcessing transformations and the RequestTaskAgent all the DataManipulation ones
  – An example of working configuration is give below:

    Transformations
    {
      DataProcessing = MCSimulation
      DataProcessing += CorsikaRepro
      DataProcessing += Merge
      DataProcessing += Analysis
      DataProcessing += DataReprocessing
      DataManipulation = Removal
      DataManipulation += Replication
    }

• Agents
  – Agents must be configured in the Systems/Transformation/[VO]/Agents section
  – The Transformation Types to be treated by the agent must be configured if and only if they are different from those set in the ‘Operations’ section. This is useful, for example, in case one wants several agents treating different transformation types, e.g.: one WorkflowTaskAgent for DataReprocessing transformations, a second for Merge and MCStripping, etc. Advantage is speedup.
  – For the WorkflowTaskAgent and RequestTaskAgent some options must be added manually
  – An example of working configuration is give below, where 2 specific WorkflowTaskAgents, each treating a different subset of transformation types have been added. Also notice the shifterProxy set by each one.

    WorkflowTaskAgent
    {
      #Transformation types to be treated by the agent
      TransType = MCSimulation
      TransType += DataReconstruction
      TransType += DataStripping
      TransType += MCStripping
      TransType += Merge
      TransType += DataReprocessing
      #Task statuses considered transient that should be monitored for updates
      TaskUpdateStatus = Submitted
      TaskUpdateStatus += Received
      TaskUpdateStatus += Waiting
      TaskUpdateStatus += Running
      TaskUpdateStatus += Matched
      TaskUpdateStatus += Completed
      TaskUpdateStatus += Failed
      shifterProxy = ProductionManager
      #Flag to enable task submission
      SubmitTasks = yes
      #Flag for checking reserved tasks that failed submission
      CheckReserved = yes
      #Flag to enable task monitoring
      MonitorTasks = yes
      PollingTime = 120
    }
MonitorFiles = yes
}
WorkflowTaskAgent-RealData
{
    #@@-phicharp@lhcb_admin - 2015-06-05 16:44:11
    TransType = DataReconstruction
    TransType += DataStripping
    shifterProxy = DataProcessing
    LoadName = WorkflowTaskAgent-RealData
    Module = WorkflowTaskAgent
}
WorkflowTaskAgent-Simulation
{
    #@@-phicharp@lhcb_admin - 2015-06-05 16:44:11
    TransType = Simulation
    TransType += MCSimulation
    shifterProxy = SimulationProcessing
    LoadName = WorkflowTaskAgent-RealData
    Module = WorkflowTaskAgent
}
RequestTaskAgent
{
    PollingTime = 120
    SubmitTasks = yes
    CheckReserved = yes
    MonitorTasks = yes
    MonitorFiles = yes
    TaskUpdateStatus = Submitted
    TaskUpdateStatus += Received
    TaskUpdateStatus += Waiting
    TaskUpdateStatus += Running
    TaskUpdateStatus += Matched
    TaskUpdateStatus += Completed
    TaskUpdateStatus += Failed
    TransType = Removal
    TransType += Replication
}

Plugins

There are two different types of plugins, i.e. TransformationAgent plugins and TaskManager plugins. The first are used to ‘group’ the input files of the tasks according to different criteria, while the latter are used to specify the tasks destinations.

TransformationAgent plugins

- Standard: group files by replicas (tasks create based on the file location)
- BySize: group files until they reach a certain size (Input size in Gb)
- ByShare: group files given the share (specified in the CS) and location
- Broadcast: take files at the source SE and broadcast to a given number of locations (used for replication)
TaskManager plugins

By default the standard plugin (BySE) sets job’s destination depending on the location of its input data. Starting from v6r13 a new ByJobType TaskManager plugin has been introduced, so that different rules for site destinations can be specified for each JobType. In order to use the ByJobType plugin, one has to:

- Set CS section Operations/Transformations/DestinationPlugin = ByJobType
- Set the JobType in the job workflow of the transformation, e.g.:

```python
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.Interfaces.API.Job import Job

t = Transformation()
job = Job()
...

job.setType('DataReprocessing')
t.setBody ( job.workflow.toXML() )
```

- Define the actual rules for each JobType in the CS section Operation/JobTypeMapping, as in the following example:

```python
JobTypeMapping
{
    AutoAddedSites = LCG.CERN.ch
    AutoAddedSites += LCG.IN2P3.fr
    AutoAddedSites += LCG.CNAF.it
    AutoAddedSites += LCG.PIC.es
    AutoAddedSites += LCG.GRIDKA.de
    AutoAddedSites += LCG.RAL.uk
    AutoAddedSites += LCG.SARA.nl
    AutoAddedSites += LCG.RRCKI.ru
    DataReprocessing
    {
        Exclude = ALL
        Allow
        {
            LCG.NIKHEF.nl = LCG.SARA.nl
            LCG.UKI-LT2-QMUL.uk = LCG.RAL.uk
            LCG.CPPM.fr = LCG.SARA.nl
            LCG.USC.es = LCG.PIC.es
            LCG.LAL.fr = LCG.CERN.ch
            LCG.LAL.fr += LCG.IN2P3.fr
            LCG.BariRECAS.it = LCG.CNAF.it
            LCG.CBPF.br = LCG.CERN.ch
            VAC.Manchester.uk = LCG.RAL.uk
        }
    }
    Merge
    {
        Exclude = ALL
        Allow
        {
            LCG.NIKHEF.nl = LCG.SARA.nl
        }
    }
}
```
By default, all sites are allowed to do every job

“AutoAddedSites” contains the list of sites allowed to run jobs with files in their local SEs

Sections under “JobTypeMapping” correspond to the different JobTypes one may want to define, e.g.: DataReprocessing, Merge, etc.

For each JobType one has to define:

* “Exclude”: the list of sites that will be removed as destination sites (“ALL” for all sites)
* “Allow”: the list of ‘helpers’, specifying sites helping another site

In the example above all sites in “AutoAddedSites” are allowed to run jobs with input files in their local SEs. These sites won’t be excluded, even if set in the Exclude list. For DataReprocessing jobs, jobs having input files at LCG.NIKHEF.nl local SEs can run both at LCG.NIKHEF.nl and at LCG.SARA.nl, etc.

Use-cases

MC Simulation

Generation of many identical jobs which don’t need Input Files and having as varying parameter a variable built from @JOB_ID.

• Agents

WorkflowTaskAgent, MCExtensionAgent (optional)

The WorkflowTaskAgent uses the TaskManager client to transform a ‘Task’ into a ‘Job’.

• Example:

```python
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.Interfaces.API.Job import Job

j = myJob()
...
t = Transformation()
t.setTransformationName("MCProd")  # This must be unique
t.setTransformationGroup("Group1")
t.setType("MCReprocessing")
t.setDescription("MC pro example")
t.setLongDescription("This is the long description of my production")  # mandatory
t.setBody(j.workflow.toXML())
t.addTransformation()  # transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
```

Re-processing

Generation of identical jobs with Input Files.

• Agents

TransformationAgent, WorkflowTaskAgent, InputDataAgent (used for DFC query)

• Example with Input Files list
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.TransformationSystem.Client.TransformationClient import TransformationClient
from DIRAC.Interfaces.API.Job import Job
j = myJob()
...
t = Transformation()
tc = TransformationClient()
t.setTransformationName("Reprocessing_1") # This must be unique
t.setType("DataReprocessing")
t.setDescription("repro example")
t.setLongDescription("This is the long description of my reprocessing") #mandatory
t.addTransformation() #transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
transID = t.getTransformationID()
tc.addFilesToTransformation(transID['Value'], infileList) # Files are added here

• Example with Input Files as a result of a DFC query. Just replace the above example with a DFC query (example taken from CTA):

   tc.createTransformationInputDataQuery(transID['Value'], {'particle': 'proton','prodName': 'ConfigtestCorsika','outputType': 'corsikaData'})

Note:
• Transformation Type = ‘DataReprocessing’
• If the ‘MonitorFiles’ option is enabled in the agent configuration, failed jobs are automatically rescheduled

Data management transformations

Generation of bulk data removal/replication requests from a fixed file list or as a result of a DFC query

• Agents

   TransformationAgent, RequestTaskAgent, InputDataAgent (for DFC query)

Requests are then treated by the RMS (see RequestManagement):

   – Check the logs of RequestExecutingAgent, e.g.:

      2014-07-08 08:27:33 UTC RequestManagement/RequestExecutingAgent/00000188_00000001 INFO: request '00000188_00000001' is done

   – Query the ReqDB to check the requests

• Example of data removal

   from DIRAC.TransformationSystem.Client.Transformation import Transformation
   from DIRAC.TransformationSystem.Client.TransformationClient import TransformationClient

   infileList = []
   ...
   t = Transformation()
tc = TransformationClient()
t.setTransformationName("DM_Removal") # Must be unique
   #t.setTransformationGroup("Group1")
t.setType("Removal")
t.setPlugin("Standard") # Not needed. The default is 'Standard'
t.setDescription("dataset1 Removal")
t.setLongDescription( "Long description of dataset1 Removal" ) # Mandatory
t.setGroupSize(2) # Here you specify how many files should be grouped within the same request, e.g. 100
t.setBody ( "Removal;RemoveFile" ) # Mandatory (the default is a ReplicateAndRegister operation)
t.addTransformation() # Transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
transID = t.getTransformationID()
tc.addFilesToTransformation(transID['Value'],infileList) # Files are added here

Note:
- It’s not needed to set a Plugin, the default is ‘Standard’
- It’s mandatory to set the Body, otherwise the default operation is ‘ReplicateAndRegister’
- It’s not needed to set a SourceSE nor a TargetSE
- This script remove all replicas of each file. We should verify how to remove only a subset of replicas (SourceSE?)
- If you add non existing files to a Transformation, you won’t get any particular status, the Transformation just does not progress

Data replication based on Catalog Query

- Example of data replication (file list as a result of a DFC query, example taken from CTA)

```python
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.TransformationSystem.Client.TransformationClient import TransformationClient

t = Transformation()
tc = TransformationClient()
t.setTransformationName("DM_ReplicationByQuery1") # This must vary
#t.setTransformationGroup("Group1")
t.setType("Replication")
t.setSourceSE(['CYF-STORM-Disk','DESY-ZN-Disk']) # A list of SE where at least 1 SE is the valid one
#t.setTargetSE(['CEA-Disk'])
t.setDescription("data Replication")
t.setLongDescription("data Replication") # mandatory
t.setGroupSize(1)
t.setPlugin("Broadcast")
t.addTransformation() #transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
transID = t.getTransformationID()
tc.createTransformationInputDataQuery(transID['Value'], {'particle': 'gamma','prodName':'Config_test300113','outputType':'Data','simtelArrayProdVersion':'prod-2_21122012_simtel','runNumSeries':'0'})
```

Actions on transformations

- **Start**
- **Stop**
- **Flush**: It has a meaning only depending on the plugin used, for example the ‘BySize’ plugin, used e.g. for merging productions, creates a task if there are enough files in input to have at least a certain size: ‘flush’ will make the ‘BySize’ plugin to ignore such requirement
- **Complete**: The transformation can be archived by the TransformationCleaningAgent. Archived means that the data produced stay, but not the entries in the TransformationDB
• **Clean**: The transformation is cleaned by the TransformationCleaningAgent: jobs are killed and removed from WMS. Produced and stored files are removed from the Storage Elements, when “OutputDirectories” parameter is set for the transformation.

# 2.7.9 Workload Management System (WMS)

The DIRAC WMS is a Pilot based Workload Management System. It provides high user jobs efficiency, hiding the heterogeneity of the underlying computing resources.

Jobs are not sent directly to the Computing Elements, or to any Computing resource. Instead, their description and requirements are stored in the DIRAC WMS (in a JDL, Job Description Language). JDLs are then matched by pilots running on the Worker Nodes.

## DIRAC pilots

This page describes what are DIRAC pilots, and how they work. To know how to develop DIRAC pilots, please refer to the Developers documentation

The current version of pilots are sometimes dubbed as “Pilots 2.0”, or “the pilots to fly in all the skies”.

### What's a DIRAC Pilot

First of all, a definition: - A **pilot** is what creates the possibility to run jobs on a worker node. Or, in other words: - a script that, at a minimum, setup (VO)DIRAC, sets the local DIRAC configuration, launches the an entity for matching jobs (e.g. the JobAgent)

A pilot can be sent, as a script to be run. Or, it can be fetched.

A pilot can run on every computing resource, e.g.: on CREAM Computing elements, on DIRAC Computing elements, on Virtual Machines in the form of contextualization script, or IAAC (Infrastructure as a Client) provided that these machines are properly configured.

A pilot has, at a minimum, to:

- install DIRAC
- configure DIRAC
- run the JobAgent

A pilot has to run on each and every computing resource type, provided that:

- Python 2.6+ on the WN
- It is an OS onto which we can install DIRAC

The same pilot script can be used everywhere.
Definitions that help understanding what’s a pilot

- **TaskQueue**: a queue of JDLs with similar requirements.
- **JobAgent**: a DIRAC agent that matches a DIRAC local configuration with a TaskQueue, and extracts a JDL from it (or more than one).
- **pilot wrapper**: a script that wraps the pilot script with conditions for running the pilot script itself (maybe multiple times).
- **pilot job**: a pilot wrapper sent to a computing element (e.g. CREAM, ARC).

The *pilot* is a “standardized” piece of code. The *pilot wrapper* is not.

An agent like the “SiteDirector” encapsulates the *pilot* in a *pilot wrapper*, then sends it to a Computing Element as a *pilot job*. But, if you don’t have the possibility to send a pilot job (e.g. the case of a Virtual Machine in a cloud), you can still find a way to start the pilot script by encapsulating it in a pilot wrapper that will be started at boot time, e.g. by supplying the proper contextualization to the VM.

**Administration**

The following CS section is used for administering the version of DIRAC that the pilot will install:

```
Operations/<Setup>/Pilot
```

Inside this section, you should define the following options, and give them a meaningful value (here, an example is give):

```
#Needed by the SiteDirector:
Version = v6r15p31  #Version to install. Add the version of your extension if you have one.
Project = myVO  #Your project name
Extensions = myVO  #The Pilot extension (if any)
Installation = mycfg.cfg  #For an optional configuration file
#For the Matcher
CheckVersion = False  #True by default, if false any version would be accepted at matching level
```
Pilot Commands

The system works with “commands”, as explained in the RFC 18. Any command can be added. If your command is executed before the “InstallDIRAC” command, pay attention that DIRAC functionalities won’t be available.

Beware that, to send pilot jobs containing a specific list of commands using the SiteDirector agents, you’ll need a SiteDirector extension.

Basically, pilot commands are an implementation of the command pattern. Commands define a toolbox of pilot capabilities available to the pilot script. Each command implements one function, like:

- Check the environment
- Get the pilot version to install
- Install (VO)DIRAC
- Configure (VO)DIRAC
- In fact, there are several configuration commands
- Configure CPU capabilities
- the famous “dirac-wms-cpu-normalization”
- Run the JobAgent

A custom list of commands can be specified using the –commands option, but if nothing is selected then the following list will be run:

```
'GetPilotVersion', 'CheckWorkerNode', 'InstallDIRAC', 'ConfigureBasics', 'CheckCECapabilities',
'CheckWNCapabilities', 'ConfigureSite', 'ConfigureArchitecture', 'ConfigureCPURequirements',
'LaunchAgent'
```

Communities can easily extend the content of the toolbox, adding more commands. If necessary, different computing resources types can run different commands.

Pilot options

The pilot can be configured to run in several ways. Please, refer to https://github.com/DIRACGrid/DIRAC/blob/rel-v6r15/WorkloadManagementSystem/PilotAgent/pilotTools.py#L395 for the full list.

Pilot extensions

In case your VO only uses Grid resources, and the pilots are only sent by SiteDirector or TaksQueueDirector agents, and you don’t plan to have any specific pilot behaviour, you can stop reading here.

Instead, in case you want, for example, to install DIRAC in a different way, or you want your pilot to have some VO specific action, you should carefully read the RFC 18, and what follows.

Pilot commands can be extended. A custom list of commands can be added starting the pilot with the -X option.

Pilots started when not controlled by the SiteDirector

You should keep reading if your resources include IAAS and IAAC type of resources, like Virtual Machines.

We have introduced a special command named “GetPilotVersion” that you should use, and possibly extend, in case you want to send/start pilots that don’t know beforehand the (VO)DIRAC version they are going to install. In this
case, you have to provide a json file freely accessible that contains the pilot version. This is tipically the case for VMs in IAAS and IAAC.

The files to consider are in https://github.com/DIRACGrid/DIRAC/blob/rel-v6r15/WorkloadManagementSystem/PilotAgent The main file in which you should look is https://github.com/DIRACGrid/DIRAC/blob/rel-v6r15/WorkloadManagementSystem/PilotAgent/dirac-pilot.py that also contains a good explanation on how the system works.

You have to provide in this case a pilot wrapper script (which can be written in bash, for example) that will start your pilot script with the proper environment. If you are on a cloud site, often contextualization of your virtual machine is done by supplying a script like the following: https://gitlab.cern.ch/mcnab/temp-diracpilot/raw/master/user_data (this one is an example from LHCb)

A simpler example is the following:

```bash
#!/bin/sh
#
# Runs as dirac. Sets up to run dirac-pilot.py
#

date --utc +"%Y-%m-%d %H:%M:%S %Z" vm-pilot Start vm-pilot

for i in "$@"
do
case $i in
    --dirac-site=*)
        DIRAC_SITE="${i#*=}"
        shift
    ;;
    --lhcb-setup=*)
        LHCBDIRAC_SETUP="${i#*=}"
        shift
    ;;
    --ce-name=*)
        CE_NAME="${i#*=}"
        shift
    ;;
    --vm-uuid=*)
        VM_UUID="${i#*=}"
        shift
    ;;
    --vmtype=*)
        VMTYPE="${i#*=}"
        shift
    ;;
    *)
        # unknown option
        ;;
esac
done

# Default if not given explicitly
LHCBDIRAC_SETUP=${LHCBDIRAC_SETUP:-LHCb-Production}

# JOB_ID is used by when reporting LocalJobID by DIRAC watchdog
#export JOB_ID="$VMTYPE:$VM_UUID"

# We might be running from cvmfs or from /var/spool/checkout
export CONTEXTDIR=`readlink -f \ dirname 50`
```

236 Chapter 2. Administrator Guide
DIRAC Documentation, Release integration

export TMPDIR=/scratch/
export EDG_WL_SCRATCH=$TMPDIR

# Needed to find software area
export VO_LHCB_SW_DIR=/cvmfs/lhcb.cern.ch

# Clear it to avoid problems (be careful if there is more than one agent!)
rm -rf /tmp/area/*

# URLs where to get scripts
DIRAC_INSTALL='https://raw.githubusercontent.com/DIRACGrid/DIRAC/raw/integration/Core/scripts/dirac-install.py'
DIRAC_PILOT='https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/WorkloadManagementSystem/PilotAgent/dirac-pilot.py'
DIRAC_PILOT_TOOLS='https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/WorkloadManagementSystem/PilotAgent/pilotTools.py'
DIRAC_PILOT_COMMANDS='https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/WorkloadManagementSystem/PilotAgent/pilotCommands.py'
LHCbDIRAC_PILOT_COMMANDS='http://svn.cern.ch/guest/dirac/LHCbDIRAC/trunk/LHCbDIRAC/WorkloadManagementSystem/PilotAgent/LHCbPilotCommands.py'

echo "Getting DIRAC Pilot 2.0 code from lhcbproject for now..."

# #get the necessary scripts
wget --no-check-certificate -O dirac-install.py $DIRAC_INSTALL
wget --no-check-certificate -O dirac-pilot.py $DIRAC_PILOT
wget --no-check-certificate -O pilotTools.py $DIRAC_PILOT_TOOLS
wget --no-check-certificate -O pilotCommands.py $DIRAC_PILOT_COMMANDS
wget --no-check-certificate -O LHCbPilotCommands.py $LHCbDIRAC_PILOT_COMMANDS

# run the dirac-pilot script
python dirac-pilot.py \
  --debug \ 
  --setup $LHCBDIRAC_SETUP \ 
  --project LHcb \ 
  -o '/LocalSite/SubmitPool=Test' \ 
  --configurationServer dips://lhcb-conf-dirac.cern.ch:9135/Configuration/Server \ 
  --Name "$CE_NAME" \ 
  --MaxCycles 1 \ 
  --name "$1" \ 
  --cert \ 
  --certLocation=/scratch/dirac/etc/grid-security \ 
  --commandExtensions LHCbPilot \ 
  --commands LHCbGetPilotVersion,CheckWorkerNode,LHCbInstallDIRAC,LHCbConfigureBasics,LHCbConfigureSite,LHCbConfigureArchitecture,LHCbConfigureCPURequirements,LaunchAgent

DIRAC jobs

Some definitions for DIRAC jobs:

- **payload** or **workflow**: the executed code. A payload describes how to run one or more application step.
- **payload executor**: a script that runs the payload (e.g. dirac-jobexec)
- **JDL**: a container of payload requirements
- **DIRAC job**: a JDL to which it is assigned a unique identifier inside the DIRAC WMS
- **JobWrapper**: a software module for running a DIRACJob in a controlled way

2.7. DIRAC Systems 237
Applications properties are reflected in payload properties.

**Job Priority Handling**

This page describes how DIRAC handles job priorities.

**Scenario**

There are two user profiles:

- Users that submit jobs on behalf of themselves. For instance normal analysis users.
- Users that submit jobs on behalf of the group. For instance production users.

In the first case, users are competing for resources, and on the second case users share them. But this two profiles also compete against each other. DIRAC has to provide a way to share the resources available. On top of that users want to specify a “UserPriority” to their jobs. They want to tell DIRAC which of their own jobs should run first and which should ran last.

DIRAC implements a priority schema to decide which user gets to run in each moment so a fair share of CPU is kept between the users.

**Priority implementation**

DIRAC handles jobs using *TaskQueues*. Each *TaskQueue* contains all the jobs that have the same requirements for a user/group combination. To prioritize user jobs, DIRAC only has to prioritize *TaskQueues*.

To handle the users competing for resources, DIRAC implements a group priority. Each DIRAC group has a priority defined. This priority can be shared or divided amongst the users in the group depending on the group properties. If the group has the *JOB_SHARING* property the priority will be shared, if it doesn’t have it the group priority will be divided amongst them. Each *TaskQueue* will get a priority based on the group and user it belongs to:

- If it belongs to a *JOB_SHARING* group, it will get 1/N of the priority being N the number of *TaskQueues* that belong to the group.
- If it does NOT, it will get 1/(N*U) being U the number of users in the group with waiting jobs and N the number of *TaskQueues* of that user/group combination.

On top of that users can specify a “UserPriority” to their jobs. To reflect that, DIRAC modifies the *TaskQueues* priorities depending on the “UserPriority” of the jobs in each *TaskQueue*. Each *TaskQueue* priority will be P*J being P the *TaskQueue* priority. J is the sum of all the “UserPriorities” of the jobs inside the *TaskQueue* divided by the sum of sums of all the “UserPriorities” in the jobs of all the *TaskQueues* belonging to the group if it has *JOB_SHARING* or to that user/group combination.

**Dynamic share corrections** DIRAC includes a priority correction mechanism. The idea behind it is to look at the past history and alter the priorities assigned based on it. It can have multiple plugins but currently it only has one. All correctors have a CS section to configure themselves under */Operations/<vo>/<setup>/JobScheduling/ShareCorrections*. The option */Operations/<vo>/<setup>/JobScheduling/ShareCorrections/ShareCorrectorsToStart* defines which correctors will be used in each iteration.
**WMSHistory corrector**  This corrector looks the running jobs for each entity and corrects the priorities to try to maintain the shares defined in the CS. For instance, if an entity has been running three times more jobs than it's current share, the priority assigned to that entity will be one third of the corresponding priority. The correction is the inverse of the proportional deviation from the expected share.

Multiple time spans can be taken into account by the corrector. Each time span is weighted in the final correction by a factor defined in the CS. A max correction can also be defined for each time span. The next example defines a valid WMSHistory corrector:

```plaintext
ShareCorrections
{
  ShareCorrectorsToStart = WMSHistory
  WMSHistory
  {
    GroupsInstance
    {
      MaxGlobalCorrectionFactor = 3
      WeekSlice
      {
        TimeSpan = 604800
        Weight = 80
        MaxCorrection = 2
      }
      HourSlice
      {
        TimeSpan = 3600
        Weight = 20
        MaxCorrection = 5
      }
    }
    lhcb_userInstance
    {
      Group = lhcb_user
      MaxGlobalCorrectionFactor = 3
      WeekSlice
      {
        TimeSpan = 604800
        Weight = 80
        MaxCorrection = 2
      }
      HourSlice
      {
        TimeSpan = 3600
        Weight = 20
        MaxCorrection = 5
      }
    }
  }
}
```

The previous example will start the WMSHistory corrector. There will be two instances of the WMSHistory corrector. The only difference between them is that the first one tries to maintain the shares between user groups and the second one tries to maintain the shares between users in the _lhcb_user_ group. It makes no sense to create a third corrector for the users in the _lhcb_prod_ group because that group has **JOB_SHARING**, so the priority is assigned to the whole group, not to the individuals.

Each WMSHistory corrector instance will correct at most \( x \cdot \frac{3}{3 - \frac{1}{3}} \) the priorities. That’s defined by the _MaxGlobalCorrectionFactor_. Each instance has two time spans to check. The first one being the last week and the second one being the last hour. The last week time span will weight 80% of the total correction, the last hour will weight the
remaining 20%. Each time span can have its own max correction. By doing so we can boost the first hour of any new entity but then try to maintain the share for longer periods. The final formula would be:

\[
\begin{align*}
\text{hourCorrection} &= \max\left( \min(\text{hourCorrection}, \text{hourMax}), \frac{1}{\text{hourMax}} \right) \\
\text{weekCorrection} &= \max\left( \min(\text{weekCorrection}, \text{weekMax}), \frac{1}{\text{weekMax}} \right) \\
\text{finalCorrection} &= \text{hourCorrection} \times \text{hourWeight} + \text{weekCorrection} \times \text{weekWeight} \\
\text{finalCorrection} &= \max\left( \min(\text{finalCorrection}, \text{globalMax}), \frac{1}{\text{globalMax}} \right)
\end{align*}
\]

2.8 Managing DIRAC Sites

DIRAC can incorporate resources provided by sites not integrated in any grid infrastructure still contributing their computing and storage capacity available as conventional clusters or file servers. In this section, the procedures to give access to the site resources are described.

2.8.1 DIRAC Computing Elements

Direct access to the site computing clusters is done by sending pilot jobs in a similar way as it is done for the grid sites. The pilot jobs are sent by a specialized agent called SiteDirector.

The SiteDirector is usually serving one or several sites and can run as part of the central service installation or as an on-site component. At the initialization phase it gets description of the site’s capacity and then runs in a loop performing the following operations:

- Check if there are tasks in the DIRAC TaskQueue eligible for running on the site;
- If there are tasks to run, check the site current occupancy in terms of numbers of already running or waiting pilot jobs;
- If there is a spare capacity on the site, submit a number of pilot jobs corresponding to the number of user jobs in the TaskQueue and the number of slots in the site computing cluster;
- Monitor the status of submitted pilot jobs, update the PilotAgentsDB accordingly;
- Retrieve the standard output/error of the pilot jobs.

SiteDirector is submitting pilot jobs with credentials of a user entitled to run generic pilots for the given user community. The generic pilots are called so as they are capable of executing jobs on behalf of community users.

SiteDirector Configuration

The SiteDirector configuration is defined in the standard way as for any DIRAC agent. It belongs to the Workload-Management System and its configuration section is:

```
/Systems/WorkloadManagement/<instance>/Agents/SiteDirector
```

The following parameters are defined in the SiteDirector configuration with some example values:

```
SiteDirector {
    # agent cycle length in seconds
    PollingTime = 60

    # one or more site names served by the agent
    Site = LCG.CNAF.it, LCG.NIKHEF.it

    # one or more CE names served by the agent. All CEs defined for the sites by default
    CEs = LCG.CNAF.it, LCG.NIKHEF.it
```

240 Chapter 2. Administrator Guide
Computing Elements

DIRAC can use different computing resources via specialized clients called *ComputingElements*. Each computing resource is accessed using an appropriate *ComputingElement* class derived from a common base class.

The *ComputingElements* should be properly described to be useful. The configuration of the *ComputingElement* is located in the inside the corresponding site section in the /Resources section. An example of a site description is given below:

```plaintext
CEs = torqsrv.ihep.ac.cn

# Types of the CEs served by the agent
CETypes = SSHTorque

# The DN of the user with whose credentials the pilots are submitted
GenericPilotDN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev

# The group of the user with whose credentials the pilots are submitted
GenericPilotGroup = lhcb_pilot

# The agent working directory
WorkDirectory = /opt/dirac/work/SiteDirector/SSHTorque

# Flag to perform pilot status updates
UpdatePilotStatus = True

# Flag to perform pilot output retrieval
GetPilotOutput = True

# Flag to send pilot info to the Accounting service
SendPilotAccounting = True
}
```

2.8. Managing DIRAC Sites
Submission mode should be "direct" in order to work with SiteDirector. Otherwise, the CE will be eligible for the use with third party broker, e.g., gLite WMS.

```sh
SubmissionMode = direct
```

Section to describe various queue in the CE:

```sh
Queues
{
  long
  {
    ...
  }
}
```

This is the general structure in which specific CE descriptions are inserted. The CE configuration is part of the general DIRAC configuration. It can be placed in the general Configuration Service or in the local configuration of the DIRAC installation. Some CE parameters are confidential, e.g., password of the account used for the SSH tunnel access to a site. The confidential parameters should be stored in the local configuration in protected files.

The SiteDirector is getting the CE descriptions from the configuration and uses them according to their specified capabilities and preferences. Configuration options specific for different types of CEs are described in the subsections below.

**CREAM Computing Element**

A commented example follows:

```sh
# Section placed in the */Resources/Sites/<domain>/<site>/CEs* directory
ce01.infn.it
{
  CEType = CREAM
  SubmissionMode = direct

  Queues
  {
    # The queue section name should be the same as in the BDII description
    long
    {
      # Max CPU time in HEP'06 unit secs
      CPUPTime = 10000
      # Max total number of jobs in the queue
      MaxTotalJobs = 5
      # Max number of waiting jobs in the queue
      MaxWaitingJobs = 2
    }
  }
}
```
Torque Computing Element

A commented example follows:

```plaintext
# Section placed in the */Resources/Sites/<domain>/site/CEs* directory
ce01.infn.it
{
    CEType = Torque
    SubmissionMode = direct

    Queues
    {
        # The queue section name should be the same as the name of the actual batch queue
        long
        {
            # Max CPU time in HEP'06 unit secs
            CPUTime = 10000
            # Max total number of jobs in the queue
            MaxTotalJobs = 5
            # Max number of waitin jobs in the queue
            MaxWaitingJobs = 2
            # Flag to include pilot proxy in the payload sent to the batch system
            BundleProxy = True
            # Directory on the CE site where the pilot standard output stream will be stored
            BatchOutput = /home/dirac_ssh/localsite/output
            # Directory on the CE site where the pilot standard output stream will be stored
            BatchError = /home/dirac_ssh/localsite/error
            # Directory where the payload executable will be stored temporarily before
            # submission to the batch system
            ExecutableArea = /home/dirac_ssh/localsite/submission
            # Flag to remove the pilot output after it was retrieved
            RemoveOutput = True
        }
    }
}
```

SSHBatch Computing Element

This is an extension of the SSHComputingElement capable of submitting several jobs on one host.

Like all SSH Computing Elements, it's defined like the following:

```plaintext
# Section placed in the */Resources/Sites/<domain>/site/CEs* directory
pc.farm.ch
{
    CEType = SSHBatch
    SubmissionMode = direct

    # Parameters of the SSH conecction to the site. The /2 indicates how many cores can be used on that
    # It's equivalent to the number of jobs that can run in parallel.
    SSHHost = pc.domain.ch/2
    SSHUser = dirac_ssh
    # if SSH password is not given, the public key connection is assumed.
    # Do not put this in the CS, put it in the local dirac.cfg of the host.
    # You don't want external people to see the password.
    SSHPassword = XXXXXXXXXX
```
# If no password, specify the key path
SSHKey = /path/to/key.pub

# In case your SSH connection requires specific attributes (see below) available in late v6r10 versions (TBD).
SSHOptions = -o option1=something -o option2=something else

Queues
{
    # The queue section name should be the same as the name of the actual batch queue
    long
    {
        # Max CPU time in HEP'06 unit secs
        CPUtilime = 10000
        # Max total number of jobs in the queue
        MaxTotalJobs = 5
        # Max number of waiting jobs in the queue
        MaxWaitingJobs = 2
        # Flag to include pilot proxy in the payload sent to the batch system
        BundleProxy = True
        # Directory on the CE site where the pilot standard output stream will be stored
        BatchOutput = /home/dirac_ssh/localsite/output
        # Directory on the CE site where the pilot standard output stream will be stored
        BatchError = /home/dirac_ssh/localsite/error
        # Directory where the payload executable will be stored temporarily before
        # submission to the batch system
        ExecutableArea = /home/dirac_ssh/localsite/submission
        # Extra options to be passed to the qsub job submission command
        SubmitOptions =
        # Flag to remove the pilot output after it was retrieved
        RemoveOutput = True
    }
}

New in version >: v6r10 The SSHOptions option.

The SSHOptions is needed when for example the user used to run the agent isn’t local and requires access to afs. As the way the agents are started isn’t a login, they do not have access to afs (as they have no token), so no access to the HOME directory. Even if the HOME environment variable is replaced, ssh still looks up the original home directory. If the ssh key and/or the known_hosts file is hosted on afs, the ssh connection is likely to fail. The solution is to pass explicitly the options to ssh with the SSHOptions option. For example:

```bash
SSHOptions = -o UserKnownHostsFile=/local/path/to/known_hosts
```

allows to have a local copy of the known_hosts file, independent of the HOME directory.

**SSHTorque Computing Element**

A commented example follows:

```bash
# Section placed in the */Resources/Sites/<domain>/<site>/CEs* directory
ce01.infn.it
{
    CEType = SSHTorque
    SubmissionMode = direct

    # Parameters of the SSH connection to the site
    SSHHost = lphe01c1.epfl.ch
```
DIRAC Documentation, Release integration

```
SSHUser = dirac_ssh
# if SSH password is no given, the public key connection is assumed
SSHPassword = XXXXXXXXXX
# specify the SSHKey if needed (like in the SSHBatchComputingElement above)
Queues
{
    # The queue section name should be the same as the name of the actual batch queue
    long
    {
        # Max CPU time in HEP '06 unit secs
        CPUTime = 10000
        # Max total number of jobs in the queue
        MaxTotalJobs = 5
        # Max number of waitin jobs in the queue
        MaxWaitingJobs = 2
        # Flag to include pilot proxy in the payload sent to the batch system
        BundleProxy = True
        # Directory on the CE site where the pilot standard output stream will be stored
        BatchOutput = /home/dirac_ssh/localsite/output
        # Directory on the CE site where the pilot standard output stream will be stored
        BatchError = /home/dirac_ssh/localsite/error
        # Directory where the payload executable will be stored temporarily before
        # submission to the batch system
        ExecutableArea = /home/dirac_ssh/localsite/submission
        # Extra options to be passed to the job submission command
        SubmitOptions =
        # Flag to remove the pilot output after it was retrieved
        RemoveOutput = True
    }
    }
}
```

Similar to SSHTorqueComputingElement is the SSHCondorComputingElement, the
SSHGEComputingElement, the SSHLSFComputingElement, and the SSHOARComputingElement.
They differ in the final backend, respectively Condor, GridEngine, LSF, and OAR.

### 2.8.2 DIRAC Storage Elements

### 2.9 Multi-VO DIRAC

**author** Bruno Santeramo <bruno.santeramo@ba.infn.it>

**date** 3rd May 2013

**version** 1.0

In this chapter a guide to install and configure DIRAC for multi-VO usage.
2.9.1 Before to start with this tutorial ...

In this tutorial

- Server hostname is: dirac.ba.infn.it
- first VO configured is: superbvo.org
- second VO configured is: pamela
- adding more VOs can be done following instructions for the second one
- foreach VO a <vo_name>_user group is configured to allow normal user operations
- foreach VO a Pool_<vo_name> submit pool is configured

Limits of this guide

- This guide must be considered as a step-by-step tutorial, not intended as documentation for DIRAC’s multi-VO capabilities.
- Please, feel free to send me via email any suggestion to improve this chapter.
2.9.2 DIRAC server installation

First step is to install DIRAC. Procedure is the same for a single VO installation, but avoiding VirtualOrganization parameter in configuration file:

```plaintext
...  
# VO name (not mandatory, useful if DIRAC will be used for a VO)  
#VirtualOrganization = superbvo.org  
...  
```

2.9.3 DIRAC client installation

Second step is to install a dirac client and configure it for new installation.

2.9.4 Configuring first VO (e.g. superbvo.org)

**Registry**

Add superb_user group

Registry

```plaintext
{  
  DefaultGroup = superb_user, user  
}
```

**Registry/VO**

Registry

```plaintext
{  
  VO  
  {  
    superbvo.org  
    {  
      SubmitPools = Pool_superbvo.org  
      VOAdmin = bsanteramo  
      VOMSName = superbvo.org  
      VOMSServers  
      {  
        voms2.cnaf.infn.it  
        {  
          DN = /C=IT/O=INFN/OU=Host/L=CNAF/CN=voms2.cnaf.infn.it  
          CA = /C=IT/O=INFN/CN=INFN CA  
          Port = 15009  
        }  
        voms-02.pd.infn.it  
        {  
          DN = /C=IT/O=INFN/OU=Host/L=Padova/CN=voms-02.pd.infn.it  
          CA = /C=IT/O=INFN/CN=INFN CA  
          Port = 15009  
        }  
      }  
    }  
  }  
}
```
Registry/Groups

Registry
{
  Groups
  {
    superb_user
    {
      Users = bsanteramo
      Properties = NormalUser
      VOMSRole = /superbvo.org
      VOMSVO = superbvo.org
      VO = superbvo.org
      SubmitPool = Pool_superbvo.org
      AutoAddVOMS = True
      AutoUploadProxy = True
      AutoUploadPilotProxy = True
    }
  }
}

Registry/VOMS

Registry
{
  VOMS
  {
    Mapping
    {
      superb_user = /superbvo.org
    }
    Servers
    {
      superbvo.org
      {
        voms2.cnaf.infn.it
        {
          DN = /C=IT/O=INFN/OU=Host/L=CNAF/CN=voms2.cnaf.infn.it
          CA = /C=IT/O=INFN/CN=INFN CA
          Port = 15009
        }
        voms-02.pd.infn.it
        {
          DN = /C=IT/O=INFN/OU=Host/L=Padova/CN=voms-02.pd.infn.it
          CA = /C=IT/O=INFN/CN=INFN CA
          Port = 15009
        }
      }
    }
  }
}

$HOME/.glite/vomses

DIRAC search for VOMS data in $HOME/.glite/vomses folder. For each VO create a file with the same name of VO and fill it in this way for every VOMS server. (Take data from http://operations-portal.egi.eu/vo)
"<VO_name>" "<VOMS_server>" "<vomses_port>" "<DN>" "<VO_name>" "<https_port>"

For example:

[managai@dirac vomses]$ cat /usr/etc/vomses/superbvo.org
"superbvo.org" "voms2.cnaf.infn.it" "15009" "/C=IT/O=INFN/OU=Host/L=CNAF/CN=voms2.cnaf.infn.it" "superbvo.org" 
"superbvo.org" "voms-02.pd.infn.it" "15009" "/C=IT/O=INFN/OU=Host/L=Padova/CN=voms-02.pd.infn.it" "superbvo.org"

Systems/Configuration - CE2CSAgent

CE2CSAgent retrieve CE info from BDII. For each VO should be an instance of the CE2CSAgent:

Systems
{
  Configuration
  {
    Production
    {
      Agents
      {
        CE2CSAgent
        {
          BannedCSs =
          MailTo =
          MailFrom =
          VirtualOrganization = superbvo.org
        }
      }
    }
  }
}

Operations - Shifter

Operations
{
  SuperB-Production
  {
    Shifter
    {
      SAMManager
      {
        User = bsanteramo
        Group = superb_user
      }
      ProductionManager
      {
        User = bsanteramo
        Group = superb_user
      }
      DataManager
      {
        User = bsanteramo
        Group = superb_user
      }
    }
  }
}
Operations/JobDescription

Add new Pool to SubmitPools

Operations
{
JobDescription
{
AllowedJobTypes = MPI
AllowedJobTypes += User
AllowedJobTypes += Test
SubmitPools = Pool_superbvo.org
}
}

Resources/FileCatalog

Configure DIRAC File Catalog (DFC)

Resources
{
FileCatalogs
{
FileCatalog
{
AccessType = Read-Write
Status = Active
Master = True
}
}
}

Resources/StorageElements/ProductionSandboxSE

Resources
{
StorageElements
{
ProductionSandboxSE
{
BackendType = DISET
AccessProtocol.1
{
Host = dirac.ba.infn.it
Port = 9196
ProtocolName = DIP
Protocol = dips
Path = /WorkloadManagement/SandboxStore
Access = remote
}
}
WorkloadManagement - PilotStatusAgent

Option value could be different, it depends on UI installed on server

Systems/WorkloadManagement/<setup>/Agents/PilotStatusAgent/GridEnv = /etc/profile.d/grid-env

Systems/WorkloadManagement - TaskQueueDirector

Systems
{
  WorkloadManagement
  {
    Production
    {
      Agents
      {
        TaskQueueDirector
        {
          DIRACVersion = v6r11p1
          Status = Active
          ListMatchDelay = 10
          extraPilotFraction = 1.0
          extraPilots = 2
          pilotsPerIteration = 100
          maxThreadsInPool = 8
          PollingTime = 30
          MaxCycles = 500
          SubmitPools = Pool_superbvo.org
          AllowedSubmitPools = Pool_superbvo.org
          Pool_superbvo.org
          {
            GridMiddleware = gLite
            ResourceBrokers = wms-multi.grid.cnaf.infn.it
            Failing =
            PrivatePilotFraction = 1.0
            MaxJobsInFillMode = 5
            Rank = ( other.GlueCEStateWaitingJobs == 0 ? ( other.GlueCEStateFreeCPUs * 10 / other.GlueCEInfoTotalCPUs + other.GlueCEInfoTotalCPUs / 500 ) : -other.GlueCEStateWaitingJobs * 4 / (other.GlueCEStateRunningJobs + 1 ) - 1 )
            GenericPilotDN = /C=IT/O=INFN/OU=Personal Certificate/L=Bari/CN=Bruno Santeramo
            GenericPilotGroup = superb_user
            GridEnv = /etc/profile.d/grid-env
            VirtualOrganization = superbvo.org
          } DIRAC
          {
            GridMiddleware = DIRAC
          } DIRAC
          {
            GridMiddleware = DIRAC
          } DIRAC
        } TaskQueueDirector
      } Agents
    } Production
  } WorkloadManagement
} Systems

2.9. Multi-VO DIRAC 251
DONE

First VO configuration finished... Upload shifter certificates, add some CE and test job submission works properly (webportal Job Launchpad is useful for testing purpose)

2.9.5 Configuring another VO (e.g. pamela)

$HOME/.glite/vomses

Add the other VO following the same convention as above.

Registry

```
Registry
{
  DefaultGroup = pamela_user, superb_user, user
}
```

Registry/VO

Add pamela

```
Registry
{
  VO
  {
    pamela
    {
      SubmitPools = Pool_pamela
      VOAdmin = bsanteramo
      VOMSName = pamela
      VOMSServers
      {
        voms.cnaf.infn.it
        {
          DN = /C=IT/O=INFN/OU=Host/L=CNAF/CN=voms.cnaf.infn.it
          CA = /C=IT/O=INFN/CN=INFN CA
          Port = 15013
        }
        voms-01.pd.infn.it
        {
          DN = /C=IT/O=INFN/OU=Host/L=Padova/CN=voms-01.pd.infn.it
          CA = /C=IT/O=INFN/CN=INFN CA
          Port = 15013
        }
      }
    }
  }
}
```

Registry/Groups

Add pamela_user
Registry
{
    Groups
    {
        pamela_user
        {
            Users = bsanteramo
            Properties = NormalUser
            VOMSRole = /pamela
            VOMSVO = pamela
            VO = pamela
            SubmitPool = Pool_pamela
            AutoAddVOMS = True
            AutoUploadProxy = True
            AutoUploadPilotProxy = True
        }
    }
}

Registry/VOMS

Add pamela parameters...

Registry
{
    VOMS
    {
        Mapping
        {
            pamela_user = /pamela
        }
        Servers
        {
            pamela
            {
                voms.cnaf.infn.it
                {
                    DN = /C=IT/O=INFN/OU=Host/L=CNAF/CN=voms.cnaf.infn.it
                    CA = /C=IT/O=INFN/CN=INFN CA
                    Port = 15013
                }
                voms-01.pd.infn.it
                {
                    DN = /C=IT/O=INFN/OU=Host/L=Padova/CN=voms-01.pd.infn.it
                    CA = /C=IT/O=INFN/CN=INFN CA
                    Port = 15013
                }
            }
        }
    }
}
Systems/Configuration - CE2CSAgent

Systems
{
  Configuration
  {
    Production
      {
        Agents
          {
            CE2CSAgent
            {
              PollingTime = 86400
              Status = Active
              MaxCycles = 500
              LogLevel = INFO
              BannedCSs =
              MailTo =
              MailFrom =
              VirtualOrganization = superbvo.org
            }
            CE2CSAgent_pamela
            {
              Module = CE2CSAgent
              #This parameter overwrites the default value
              VirtualOrganization = pamela
            }
          }
        }
      }
  }
}

As dirac_admin group member, enter dirac-admin-sysadmin-cli

(dirac.ba.infn.it)> install agent Configuration CE2CSAgent_pamela -m CE2CSAgent -p VirtualOrganization=pamela
agent Configuration_CE2CSAgent_pamela is installed, runit status: Run

Operations - adding pamela section

Operations
{
  EMail
  {
    Production = bruno.santeramo@ba.infn.it
    Logging = bruno.santeramo@ba.infn.it
  }
  SuperB-Production
  {
    Shifter
    {
      SAMManager
      {
        User = bsanteramo
        Group = superb_user
      }
      ProductionManager
      {
      }
  }
}


Systems/WorkloadManagement - TaskQueueDirector

Systems
{
  WorkloadManagement
  {
    Production
    {
      Agents
      {
        TaskQueueDirector
        {
          DIRACVersion = v6r11p1
        }
      }
    }
  }
}
2.10 Administrator Command Reference

In this subsection all the dirac-admin commands available are explained. You can get up-to-date documentation by using the -h switch on any of them. The following command line flags are common to all DIRAC scripts making use of the parseCommandLine method of the base Script class:

General options:
- o: --option= : Option=value to add
-s: --section= : Set base section for relative parsed options
-c: --cert= : Use server certificate to connect to Core Services
-d --debug : Set debug mode (-dd is extra debug)
-h --help : Shows this help

General information:

2.10.1 dirac-admin-externals-versions

Usage:

dirac-admin-externals-versions.py (<options>|<cfgFile>)*

Example:

$ dirac-admin-externals-versions

2.10.2 dirac-admin-service-ports

Print the service ports for the specified setup

Usage:

dirac-admin-service-ports [option|cfgfile] ... [Setup]

Arguments:

Setup: Name of the setup

Example:

$ dirac-admin-service-ports
{'Accounting/DataStore': 9133,
 'Accounting/ReportGenerator': 9134,
 'DataManagement/FileCatalog': 9197,
 'DataManagement/StorageElement': 9148,
 'DataManagement/StorageElementProxy': 9149,
 'Framework/BundleDelivery': 9158,
 'Framework/Monitoring': 9142,
 'Framework/Notification': 9154,
 'Framework/Plotting': 9157,
 'Framework/ProxyManager': 9152,
 'Framework/SecurityLogging': 9153,
 'Framework/SystemAdministrator': 9162,
 'Framework/SystemLogging': 9141,
 'Framework/SystemLoggingReport': 9144,
 'Framework/UserProfileManager': 9155,
 'RequestManagement/RequestManager': 9143,
 'WorkloadManagement/JobManager': 9132,
 'WorkloadManagement/JobMonitoring': 9130,
 'WorkloadManagement/JobStateUpdate': 9136,
 'WorkloadManagement/MPIService': 9171,
 'WorkloadManagement/Matcher': 9170,
 'WorkloadManagement/SandboxStore': 9196,
 'WorkloadManagement/WMSAdministrator': 9145}
2.10.3 dirac-platform

Linux_x86_64_glibc-2.5

Example:

$ dirac-platform
Linux_x86_64_glibc-2.5

Managing Registry:

2.10.4 dirac-admin-add-group

Add or Modify a Group info in DIRAC

Usage:

dirac-admin-add-group [option|cfgfile] ... Property=<Value> ...

Arguments:

Property=<Value>: Other properties to be added to the User like (VOMSRole=XXXX)

Options:

-G: --GroupName: : Name of the Group (Mandatory)
-U: --UserName: : Short Name of user to be added to the Group (Allow Multiple instances or None)
-P: --Property: : Property to be added to the Group (Allow Multiple instances or None)

Example:

$ dirac-admin-add-group -G dirac_test

2.10.5 dirac-admin-add-host

Add or Modify a Host info in DIRAC

Usage:

dirac-admin-add-host [option|cfgfile] ... Property=<Value> ...

Arguments:

Property=<Value>: Other properties to be added to the User like (Responsible=XXXX)

Options:

-H: --HostName: : Name of the Host (Mandatory)
-D: --HostDN: : DN of the Host Certificate (Mandatory)
-P: --Property: : Property to be added to the Host (Allow Multiple instances or None)

Example:

2.10.6  dirac-admin-add-user

Add or Modify a User info in DIRAC

Usage:

```plaintext
dirac-admin-add-user [option|cfgfile] ... Property=<Value> ...
```

Arguments:

Property=<Value>: Properties to be added to the User like (Phone=XXXX)

Options:

- `-N`: --UserName: : Short Name of the User (Mandatory)
- `-D`: --UserDN: : DN of the User Certificate (Mandatory)
- `-M`: --UserMail: : eMail of the user (Mandatory)
- `-G`: --UserGroup: : Name of the Group for the User (Allow Multiple instances or None)

Example:

```plaintext
$ dirac-admin-add-user -N vhamar -D /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar -M hamar@cppm.in2p3.fr -G dirac_user
```

2.10.7  dirac-admin-delete-user

Remove User from Configuration

Usage:

```plaintext
dirac-admin-delete-user [option|cfgfile] ... User ...
```

Arguments:

User: User name

Example:

```plaintext
$ dirac-admin-delete-user vhamar
```

2.10.8  dirac-admin-list-hosts

Usage:

```plaintext
dirac-admin-list-hosts.py (<options>|<cfgFile>)*
```

Options:

- `-e`: --extended : Show extended info

Example:

```plaintext
$ dirac-admin-list-hosts
dirac.in2p3.fr
host-dirac.in2p3.fr
```
### 2.10.9 dirac-admin-list-users

Lists the users in the Configuration. If no group is specified return all users.

**Usage:**

```
dirac-admin-list-users [option|cfgfile] ... [Group] ...
```

**Arguments:**

- **Group:** Only users from this group (default: all)

**Options:**

- `-e` --extended : Show extended info

**Example:**

```
$ dirac-admin-list-users
All users registered:
vhamar
msapunov
atsareg
```

### 2.10.10 dirac-admin-modify-user

Modify a user in the CS.

**Usage:**

```
dirac-admin-modify-user [option|cfgfile] ... user DN group [group] ...
```

**Arguments:**

- **user:** User name
- **DN:** DN of the User
- **group:** Add the user to the group

**Options:**

- `-p` --property= : Add property to the user `<name>=<value>`
- `-f` --force : create the user if it doesn't exist

**Example:**

```
$ dirac-admin-modify-user vhamar group dirac_user
```

### 2.10.11 dirac-admin-sync-users-from-file

Sync users in Configuration with the cfg contents.

**Usage:**

```
dirac-admin-sync-users-from-file [option|cfgfile] ... UserCfg
```
Arguments:

UserCfg: Cfg FileName with Users as sections containing DN, Groups, and other properties as options.

Options:

- `--test`: Only test. Don't commit changes.

Example:

```
$ dirac-admin-sync-users-from-file file_users.cfg
```

### 2.10.12 dirac-admin-user-quota

Show storage quotas for specified users or for all registered users if nobody is specified.

Usage:

```
dirac-admin-user-quota [user1 ...]
```

Example:

```
$ dirac-admin-user-quota

<table>
<thead>
<tr>
<th>Username</th>
<th>Quota (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>atsareg</td>
<td>None</td>
</tr>
<tr>
<td>msapunov</td>
<td>None</td>
</tr>
<tr>
<td>vhamar</td>
<td>None</td>
</tr>
</tbody>
</table>
```

### 2.10.13 dirac-admin-users-with-proxy

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

- `--valid=`: Required HH:MM for the users.

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

- `--valid=`: Required HH:MM for the users.

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

- `--valid=`: Required HH:MM for the users.

Example:
Managing Resources:

### 2.10.14 dirac-admin-add-site

Add a new DIRAC SiteName to DIRAC Configuration, including one or more CEs

Usage:
```
dirac-admin-add-site [option|cfgfile] ... DIRACSiteName GridSiteName CE [CE] ...
```

Arguments:
- **DIRACSiteName**: Name of the site for DIRAC in the form GRID.LOCATION.COUNTRY (ie: LCG.CERN.ch)
- **GridSiteName**: Name of the site in the Grid (ie: CERN-PROD)
- **CE**: Name of the CE to be included in the site (ie: ce111.cern.ch)

Example:
```
$ dirac-admin-add-site LCG.IN2P3.fr IN2P3-Site
```

### 2.10.15 dirac-admin-allow-catalog

Enable usage of the File Catalog mirrors at given sites

Usage:
```
dirac-admin-allow-catalog site1 [site2 ...]
```

### 2.10.16 dirac-admin-allow-se

Enable using one or more Storage Elements

Usage:
```
dirac-admin-allow-se SE1 [SE2 ...]
```

Options:
-r --AllowRead : Allow only reading from the storage element
-w --AllowWrite : Allow only writing to the storage element
-k --AllowCheck : Allow only check access to the storage element
-m --Mute : Do not send email
-S: --Site= : Allow all SEs associated to site

Example:
$ dirac-admin-allow-se M3PEC-disk
$

2.10.17 dirac-admin-allow-site

Add Site to Active mask for current Setup

Usage:

dirac-admin-allow-site [option|cfgfile] ... Site Comment

Arguments:

Site: Name of the Site

Comment: Reason of the action

Options:

-E: --email= : Boolean True/False (True by default)

Example:

$ dirac-admin-allow-site LCG.IN2P3.fr 'FRANCE'

2.10.18 dirac-admin-ban-catalog

Ban the File Catalog mirrors at one or more sites

Usage:

dirac-admin-ban-catalog site1 [site2 ...]

Example:

$ dirac-admin-ban-catalog LCG.IN2P3.fr

2.10.19 dirac-admin-ban-se

Ban one or more Storage Elements for usage

Usage:
dirac-admin-ban-se SE1 [SE2 ...]

Options:

- **-r --BanRead**: Ban only reading from the storage element
- **-w --BanWrite**: Ban writing to the storage element
- **-k --BanCheck**: Ban check access to the storage element
- **-m --Mute**: Do not send email
- **-S: --Site=**: Ban all SEs associate to site (note that if writing is allowed, check is always allowed)

Example:

```
$ dirac-admin-ban-se M3PEC-disk
```

### 2.10.20 dirac-admin-ban-site

Remove Site from Active mask for current Setup

Usage:

```
dirac-admin-ban-site [option|cfgfile] ... Site Comment
```

Arguments:

- **Site**: Name of the Site
- **Comment**: Reason of the action

Options:

- **-E: --email=**: Boolean True/False (True by default)

Example:

```
$ dirac-admin-ban-site LCG.IN2P3.fr 'Pilot installation problems'
```

### 2.10.21 dirac-admin-bdii-ce-state

Check info on BDII for CE state

Usage:

```
dirac-admin-bdii-ce-state [option|cfgfile] ... CE
```

Arguments:

- **CE**: Name of the CE (ie: ce111.cern.ch)

Options:

- **-H: --host=**: BDII host
- **-V: --vo=**: vo
### 2.10.22 dirac-admin-bdii-ce-voview

Check info on BDII for VO view of CE

Usage:

```
dirac-admin-bdii-ce-voview [option|cfgfile] ... CE
```

Arguments:

- **CE**: Name of the CE (e.g., `cell1.cern.ch`)

Options:

- `-H`: `--host=` : BDII host
- `-V`: `--vo=` : vo

Example:

```
$ dirac-admin-bdii-ce-voview LCG.IN2P3.fr
```

### 2.10.23 dirac-admin-bdii-ce

Check info on BDII for CE

Usage:

```
dirac-admin-bdii-ce [option|cfgfile] ... CE
```

Arguments:

- **CE**: Name of the CE (e.g., `cell1.cern.ch`)

Options:

- `-H`: `--host=` : BDII host

Example:

```
$ dirac-admin-bdii-ce LCG.IN2P3.fr
```

### 2.10.24 dirac-admin-bdii-cluster

Check info on BDII for Cluster

Usage:

```
dirac-admin-bdii-cluster [option|cfgfile] ... CE
```

Arguments:

- **CE**: Name of the CE (e.g., `cell1.cern.ch`)

Options:

- `-H`: `--host=` : BDII host

Example:
2.10.25 dirac-admin-bdii-sa

Check info on BDII for SA

Usage:
```
dirac-admin-bdii-sa [option|cfgfile] ... Site
```

Arguments:
```
Site: Name of the Site (ie: CERN-PROD)
```

Options:
```
-H: --host= : BDII host
-V: --vo= : vo
```

Example:
```
$ dirac-admin-bdii-sa CERN-PROD
```

2.10.26 dirac-admin-bdii-site

Check info on BDII for Site

Usage:
```
dirac-admin-bdii-site [option|cfgfile] ... Site
```

Arguments:
```
Site: Name of the Site (ie: CERN-PROD)
```

Options:
```
-H: --host= : BDII host
```

Example:
```
$ dirac-admin-bdii-site CERN-PROD
```

2.10.27 dirac-admin-ce-info

Retrieve Site Associated to a given CE

Usage:
```
dirac-admin-ce-info [option|cfgfile] ... CE ...
```

Arguments:
```
CE: Name of the CE
```

Options:
-G: --Grid= : Define the Grid where to look (Default: LCG)

Example:

$ dirac-admin-ce-info LCG.IN2P3.fr

2.10.28 dirac-admin-get-banned-sites

Usage:

dirac-admin-get-banned-sites.py (<options>|<cfgFile>)*

Example:

$dirac-admin-get-banned-sites.py
LCG.IN2P3.fr Site not present in logging table

2.10.29 dirac-admin-get-site-mask

Get the list of sites enabled in the mask for job submission

Usage:

dirac-admin-get-site-mask [options]

Example:

$ dirac-admin-get-site-mask
LCG.CGG.fr
LCG.CPPM.fr
LCG.GRIF.fr
LCG.IBCP.fr
LCG.IN2P3.fr
LCG.IPNL.fr
LCG.IPSL-IPGP.fr
LCG.IRES.fr
LCG.LAPP.fr
LCG.LPSC.fr
LCG.M3PEC.fr
LCG.MSFG.fr

2.10.30 dirac-admin-get-site-protocols

Check the defined protocols for all SEs of a given site

Usage:

dirac-admin-get-site-protocols [option|cfgfile] ... PilotID ...

Options:

- --Site= : Site for which protocols are to be checked (mandatory)

Example:
### 2.10.31 dirac-admin-set-site-protocols

Defined protocols for each SE for a given site.

**Usage:**
```
dirac-admin-set-site-protocols [option|cfgfile] ... Protocol ...
```

**Arguments:**
- Protocol: SE access protocol (mandatory)

**Options:**
- **--Site=** : Site for which protocols are to be set (mandatory)

**Example:**
```
$ dirac-admin-set-site-protocols
```

### 2.10.32 dirac-admin-site-info

Print Configuration information for a given Site

**Usage:**
```
dirac-admin-site-info [option|cfgfile] ... Site ...
```

**Arguments:**
- Site: Name of the Site

**Example:**
```
$ dirac-admin-site-info LCG.IN2P3.fr
{'CE': 'cclcgceli01.in2p3.fr, cclcgceli03.in2p3.fr, sbgc1.in2p3.fr, clrlcgce01.in2p3.fr, clrlcgce02.in2p3.fr, clrlcgce03.in2p3.fr, grid10.lal.in2p3.fr, polgrid1.in2p3.fr', 'Coordinates': '4.8655:45.7825', 'Mail': 'grid.admin@cc.in2p3.fr', 'MoUTierLevel': '1', 'Name': 'IN2P3-CC', 'SE': 'IN2P3-disk, DIRAC-USER'}
```

### 2.10.33 dirac-admin-site-mask-logging

Retrieves site mask logging information.

**Usage:**
```
```

---

```bash
$ dirac-admin-get-site-protocols --Site LCG.IN2P3.fr
Summary of protocols for StorageElements at site LCG.IN2P3.fr
StorageElement ProtocolsList
IN2P3-disk   file, root, rfio, gsiftp
```
dirac-admin-site-mask-logging [option|cfgfile] ... Site ...

Arguments:
Site: Name of the Site

Example:
$ dirac-admin-site-mask-logging LCG.IN2P3.fr
Site Mask Logging Info for LCG.IN2P3.fr
Active 2010-12-08 21:28:16 (atsareg) ""

2.10.34 dirac-config-ce

Configure a Local CE to be used in a DIRAC Site

Usage:
dirac-config-ce [option]... [cfgfile]

Arguments:
cfgfile: DIRAC Cfg with description of the configuration (optional)

Options:
-N: --Name= : Computing Element Name (Mandatory)
-T: --Type= : Computing Element Type (Mandatory)
-D --Director : Setup a Director Using this CE

Workload management commands:

2.10.35 dirac-admin-get-job-pilot-output

Retrieve the output of the pilot that executed a given job

Usage:
dirac-admin-get-job-pilot-output [option|cfgfile] ... JobID ...

Arguments:
JobID: DIRAC ID of the Job

Example:
$ dirac-admin-get-job-pilot-output 34

2.10.36 dirac-admin-get-job-pilots

Retrieve info about pilots that have matched a given Job

Usage:
2.10.37 dirac-admin-get-pilot-info

Retrieve available info about the given pilot

Usage:

dirac-admin-get-pilot-info [option|cfgfile] ... PilotID ...

Arguments:

PilotID: Grid ID of the pilot

Options:

-e --extended : Get extended printout

Example:

$ dirac-admin-get-pilot-info https://marlb.in2p3.fr:9000/26KCLKBttxXKHIf4_2rQjkw

{'AccountingSent': 'False',
'BenchMark': 0.0,
'Broker': 'marwms.in2p3.fr',
'DestinationSite': 'cclgcclci01.in2p3.fr',
'GridSite': 'LCG.IN2P3.fr',
'GridType': 'gLite',
'LastUpdateTime': datetime.datetime(2011, 2, 21, 12, 49, 14),
'OutputReady': 'False',
'OwnerDN': '/O=GRID-FR/C=FR/O=CNRS/OU=LPC/CN=Sebastien Guizard',
'OwnerGroup': '/biomed',
'ParentID': 0L,
'PilotID': 2247L,
'SubmissionTime': '',
'Status': 'Done',
'TaskQueueID': 399L}
2.10.38 **dirac-admin-get-pilot-logging-info**

Retrieve logging info of a Grid pilot

Usage:

```
dirac-admin-get-pilot-logging-info [option|cfgfile] ... PilotID ...
```

Arguments:

- **PilotID**: Grid ID of the pilot

Example:

```
Pilot Reference: %s https://marlb.in2p3.fr:9000/26KCLKBFtxXKH4ZrQjkw
===================== glite-job-logging-info Success =====================
LOGGING INFORMATION:
Printing info for the Job : https://marlb.in2p3.fr:9000/26KCLKBFtxXKH4ZrQjkw
---
Event: RegJob
- Arrived = Mon Feb 21 13:27:50 2011 CET
- Host = marwms.in2p3.fr
- Jobtype = SIMPLE
- Level = SYSTEM
- Ns = https://marwms.in2p3.fr:7443/glite_wms_wmproxy_server
- Nsubjobs = 0
- Parent = https://marlb.in2p3.fr:9000/WQHVOB1mI4oqrlYz2ZKtgA
- Priority = asynchronous
- Source = NetworkServer
```
2.10.40 dirac-admin-kill-pilot

Kill the specified pilot

Usage:

dirac-admin-kill-pilot <pilot reference>

2.10.41 dirac-admin-pilot-summary

Usage:

dirac-admin-pilot-summary.py (<options>|<cfgFile>)*

Example:

```
$ dirac-admin-pilot-summary

<table>
<thead>
<tr>
<th>CE</th>
<th>Status</th>
<th>Count</th>
<th>Status</th>
<th>Count</th>
<th>Status</th>
<th>Count</th>
<th>Status</th>
<th>Count</th>
<th>Status</th>
<th>Count</th>
<th>Status</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbgce1.in2p3.fr</td>
<td>Done</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lpsc-ce.in2p3.fr</td>
<td>Done</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lyogrid02.in2p3.fr</td>
<td>Done</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>egee-ce.datagrid.jussieu.fr</td>
<td>Aborted</td>
<td>81</td>
<td>Done</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc1cgceli03.in2p3.fr</td>
<td>Done</td>
<td>275</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>marce01.in2p3.fr</td>
<td>Done</td>
<td>156</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node07.datagrid.cea.fr</td>
<td>Done</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc1cgceli01.in2p3.fr</td>
<td>Aborted</td>
<td>1</td>
<td>Done</td>
<td>235</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ce0.m3pec.u-bordeaux1.fr</td>
<td>Done</td>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grive11.ibcp.fr</td>
<td>Aborted</td>
<td>3</td>
<td>Done</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lptace01.msfg.fr</td>
<td>Aborted</td>
<td>3</td>
<td>Aborted_Day</td>
<td>3</td>
<td>Done</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ipnls2001.in2p3.fr</td>
<td>Done</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Aborted</td>
<td>89</td>
<td>Done</td>
<td>1423</td>
<td>Ready</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lapp-ce01.in2p3.fr</td>
<td>Aborted</td>
<td>1</td>
<td>Done</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

2.10.42 dirac-admin-reoptimize-jobs

Usage:

dirac-admin-reoptimize-jobs.py (<options>|<cfgFile>)*

Example:

```
$ dirac-admin-reoptimize-jobs
```

2.10.43 dirac-admin-reset-job

Reset a job or list of jobs in the WMS

Usage:

dirac-admin-reset-job [option|cfgfile] ... JobID ...

Arguments:

JobID: DIRAC ID of the Job

Example:
2.10.44 dirac-admin-show-task-queues

Usage:

dirac-admin-show-task-queues.py (<options>|<cfgFile>)*

Example:

$ dirac-admin-show-task-queues
Getting TQs..
* TQ 401
  CPULtime: 360
  Jobs: 3
  OwnerDN: /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
  OwnerGroup: dirac_user
  Priority: 1.0
  Setup: Dirac-Production

2.10.45 dirac-admin-submit-pilot-for-job

Submit a DIRAC pilot for the given DIRAC job. Requires access to taskQueueDB and PilotAgentsDB

Usage:

dirac-admin-submit-pilot-for-job [option|cfgfile] ... JobID ...

Arguments:

JobID: DIRAC Job ID

Example:

$ dirac-admin-submit-pilot-for-job 1847

2.10.46 dirac-jobexec

Usage:

dirac-jobexec.py (<options>|<cfgFile>)*

Options:

-p: --parameter= : Parameters that are passed directly to the workflow

Transformation management commands:

2.10.47 dirac-transformation-archive

Usage:
dirac-transformation-archive.py (<options>|<cfgFile>)*

2.10.48 dirac-transformation-clean

Usage:

dirac-transformation-clean.py (<options>|<cfgFile>)*

2.10.49 dirac-transformation-cli

Launch the Transformation shell

Usage:

dirac-transformation-cli [option]

2.10.50 dirac-transformation-remove-output

Usage:

dirac-transformation-remove-output.py (<options>|<cfgFile>)*

2.10.51 dirac-transformation-resolve-problematics

Resolve problematic files for the specified transformations

Usage:

dirac-transformation-resolve-problematics [options] TransID [TransID]

2.10.52 dirac-transformation-verify-outputdata

Usage:

dirac-transformation-verify-outputdata.py (<options>|<cfgFile>)*

Managing DIRAC installation:

2.10.53 dirac-framework-ping-service

Ping the given DIRAC Service

Usage:

dirac-framework-ping-service [option|cfgfile] ... System Service|System/Agent

Arguments:

System: Name of the DIRAC system (ie: WorkloadManagement)

Service: Name of the DIRAC service (ie: Matcher)
Example:

$ dirac-framework-ping-service WorkloadManagement MPIService
{'OK': True,
 'Value': {'cpu times': {'children system time': 0.0,
 'children user time': 0.0,
 'elapsed real time': 8778481.7200000007,
 'system time': 54.85999999999999,
 'user time': 361.06999999999999},
 'host uptime': 4485212L,
 'load': '3.44 3.90 4.02',
 'name': 'WorkloadManagement/MPIService',
 'service start time': datetime.datetime(2011, 2, 21, 8, 58, 35, 521438),
 'service uptime': 85744,
 'service url': 'dips://dirac.in2p3.fr:9171/WorkloadManagement/MPIService',
 'time': datetime.datetime(2011, 3, 14, 11, 47, 40, 394957),
 'version': 'v5r12-pre9'),
 'rpcStub': (('WorkloadManagement/MPIService',
 {'delegatedDN': '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
 'delegatedGroup': 'dirac_user',
 'skipCACheck': True,
 'timeout': 120}),
 'ping',
 ()))

2.10.54 dirac-install-agent


Do the initial installation and configuration of a DIRAC agent

Usage:

dirac-install-agent [option|cfgfile] ... System Agent|System/Agent

Arguments:

System: Name of the DIRAC system (ie: WorkloadManagement)

Agent: Name of the DIRAC agent (ie: JobCleaningAgent)

Options:

-w --overwrite : Overwrite the configuration in the global CS
-m: --module= : Python module name for the agent code
-p: --parameter= : Special agent option

2.10.55 dirac-install-db

2013-02-06 12:30:30 UTC Framework NOTICE: DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7

Create a new DB on the local MySQL server

Usage:
dirac-install-db [option|cfgFile] ... DB ...

Arguments:

DB: Name of the Database (mandatory)

2.10.56 dirac-install-service

Do the initial installation and configuration of a DIRAC service

Usage:

dirac-install-service [option|cfgfile] ... System Service|System/Service

Arguments:

System: Name of the DIRAC system (ie: WorkloadManagement)

Service: Name of the DIRAC service (ie: Matcher)

Options:

-w    --overwrite : Overwrite the configuration in the global CS

-m:    --module= : Python module name for the service code

-p:    --parameter= : Special service option

2.10.57 dirac-install-web-portal

Do the initial installation of a DIRAC Web portal

Usage:

dirac-install-web-portal [option|cfgfile] ...

2.10.58 dirac-install

2013-02-06 12:30:27 UTC dirac-install [NOTICE] Processing installation requirements

Usage:

r:    release= : Release version to install

l:    project= : Project to install

e:    extraModules= : Extra modules to install (comma separated)

t:    installType= : Installation type (client/server)

i:    pythonVersion= : Python version to compile (25/24)
p:  platform=  : Platform to install
P:  installationPath=  : Path where to install (default current working dir)
b  build  : Force local compilation
g:  grid=  : lcg tools package version
B  noAutoBuild  : Do not build if not available
v  useVersionsDir  : Use versions directory
u:  baseURL=  : Use URL as the source for installation tarballs
V:  installation=  : Installation from which to extract parameter values
X  externalsOnly  : Only install external binaries
M:  defaultsURL=  : Where to retrieve the global defaults from
T:  Timeout=  : Timeout for downloads (default = %s)

Known options and default values from /defaults section of releases file

Release =
  Project = DIRAC
  ModulesToInstall = []
  ExternalsType = client
  PythonVersion = 26
  LcgVer =
  UseVersionsDir = False
  BuildExternals = False
  NoAutoBuild = False
  Debug = False
  Timeout = 300

2.10.59 dirac-restart-component

Restart DIRAC component using runsvctrl utility

Usage:

dirac-restart-component [option|cfgfile] ... [System [Service|Agent]]

Arguments:

System: Name of the system for the component (default *: all)
Service|Agent: Name of the particular component (default *: all)
### 2.10.60 dirac-restart-mysql

Restart DIRAC MySQL server

**Usage:**

```
dirac-restart-mysql [option|cfgfile] ...
```

### 2.10.61 dirac-start-component

Start DIRAC component using runsvctrl utility

**Usage:**

```
dirac-start-component [option|cfgfile] ... [system [service|agent]]
```

**Arguments:**

- **system:** Name of the system for the component (default *: all)
- **service|agent:** Name of the particular component (default *: all)

### 2.10.62 dirac-start-mysql

Start DIRAC MySQL server

**Usage:**

```
dirac-start-mysql [option|cfgfile] ...
```

### 2.10.63 dirac-status-component

Status of DIRAC components using runsvstat utility

**Usage:**

```
dirac-status-component [option|cfgfile] ... [system [service|agent]]
```

**Arguments:**

- **system:** Name of the system for the component (default *: all)
- **service|agent:** Name of the particular component (default *: all)

**Example:**

```
$ dirac-status-component
DIRAC Root Path = /vo/dirac/versions/Lyon-HEAD-1296215324

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Uptime</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkloadManagement_PilotStatusAgent</td>
<td>Run</td>
<td>4029</td>
<td>1697</td>
</tr>
<tr>
<td>WorkloadManagement_JobHistoryAgent</td>
<td>Run</td>
<td>4029</td>
<td>1679</td>
</tr>
<tr>
<td>Framework_CAUpdateAgent</td>
<td>Run</td>
<td>4029</td>
<td>1658</td>
</tr>
<tr>
<td>Framework_SecurityLogging</td>
<td>Run</td>
<td>4025</td>
<td>2111</td>
</tr>
<tr>
<td>WorkloadManagement_Matcher</td>
<td>Run</td>
<td>4029</td>
<td>1692</td>
</tr>
<tr>
<td>WorkloadManagement_StalledJobAgent</td>
<td>Run</td>
<td>4029</td>
<td>1704</td>
</tr>
<tr>
<td>WorkloadManagement_JobCleaningAgent</td>
<td>Run</td>
<td>4029</td>
<td>1676</td>
</tr>
</tbody>
</table>
```
2.10.64 dirac-stop-component

Stop DIRAC component using runsvctrl utility

Usage:

```
dirac-stop-component [option|cfgfile] ... [system [service|agent]]
```

Arguments:

```
system: Name of the system for the component (default *: all)

service|agent: Name of the particular component (default *: all)
```

2.10.65 dirac-stop-mysql

Stop DIRAC MySQL server

Usage:

```
dirac-stop-mysql [option|cfgfile] ...
```
2.10.66 dirac-monitoring-get-components-status

Usage:

dirac-monitoring-get-components-status.py (<options>|<cfgFile>)*

2.10.67 dirac-service

2013-02-06 13:06:27 UTC Framework FATAL: You must specify which server to run!

2.10.68 dirac-setup-site

Initial installation and configuration of a new DIRAC server (DBs, Services, Agents, Web Portal,...)

Usage:

dirac-setup-site [option] ... [cfgfile]

Arguments:

cfgfile: DIRAC Cfg with description of the configuration (optional)

2.10.69 dirac-configure

Main script to write dirac.cfg for a new DIRAC installation and initial download of CAs and CRLs

Usage:

dirac-configure [option|cfgfile] ...

Options:

-S: --Setup= : Set <setup> as DIRAC setup
-C: --ConfigurationServer= : Set <server> as DIRAC configuration server
-I --IncludeAllServers : include all Configuration Servers
-n: --SiteName= : Set <sitename> as DIRAC Site Name
-N: --CEName= : Determiner <sitename> from <cname>
-V: --VO= : Set the VO name
-W: --gateway= : Configure <gateway> as DIRAC Gateway for the site
-U --UseServerCertificate : Configure to use Server Certificate
-H --SkipCAChecks : Configure to skip check of CAs
-D --SkipCADownload : Configure to skip download of CAs
-v --UseVersionsDir : Use versions directory
-A: --Architecture= : Configure /Architecture=<architecture>
2.10.70 dirac-admin-get-CAs

Refresh the local copy of the CA certificates and revocation lists. Connects to the BundleDelivery service to obtain the tar balls. Needed when proxies appear to be invalid.

Usage:

dirac-admin-get-CAs.py (<options>|<cfgFile>)*

Example:

$ dirac-admin-externals-versions

2.10.71 dirac-info

Report info about local DIRAC installation

Usage:

dirac-info [option|cfgfile] ... Site

Example:

$ dirac-info
  DIRAC version : v5r12
  Setup : Dirac-Production
  VirtualOrganization : vo.formation.idgrilles.fr

2.10.72 dirac-version

Return the current dirac version used by the client.

Example:

$ dirac-version
  v5r12-pre9

Managing DIRAC software:

2.10.73 dirac-create-distribution-tarball

Create tarballs for a given DIRAC release

Usage:

dirac-create-distribution-tarball <option> ...

A source, name and version are required to build the tarball

For instance:
**dirac-create-distribution-tarball**

-n DIRAC -v v1r0 -z svn -u http://svnweb.cern.ch/guest/dirac/DIRAC/tags/DIRAC/v1r0

Options:
- **-v**: --version= : version to tar
- **-u**: --source= : VCS path to retrieve sources from
- **-D**: --destination= : Destination where to build the tar files
- **-n**: --name= : Tarball name
- **-z**: --vcs= : VCS to use to retrieve the sources (try to find out if not specified)
- **-b**: --branch= : VCS branch (if needed)
- **-p**: --path= : VCS path (if needed)
- **-K**: --releasenotes= : Path to the release notes
- **-A**: --notesoutside : Leave a copy of the compiled release notes outside the tarball

### 2.10.74 dirac-create-svn-branch

Create a new branch in svn

Usage:

```
dirac-create-svn-branch [option|cfgfile] ...
```

Specifying a version is mandatory

Options:
- **-p**: --package= : package to branch (default = DIRAC)
- **-v**: --version= : version to branch from
- **-u**: --username= : svn username to use
- **-l**: --devel= : Create a development branch with name
- **-e**: --pre= : Create a pre branch with name
- **-r**: --release= : Create a release branch with name

### 2.10.75 dirac-create-svn-tag

Tag a new release in SVN

Usage:

```
dirac-create-svn-tag [option|cfgfile] ...
```

Options:
### 2.10.76 dirac-deploy-scripts

Scripts will be deployed at /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7/scripts

Inspecting DIRAC module

Example:

```bash
$ dirac-deploy-scripts
Scripts will be deployed at /afs/in2p3.fr/home/h/hamar/DIRAC-v5r12/scripts
Inspecting DIRAC module
Inspecting EELADIRAC module
```

### 2.10.77 dirac-distribution

Create tarballs for a given DIRAC release

Usage:

```
dirac-distribution [option|cfgfile] ...
```

Options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r: --releases=</td>
<td>releases to build (mandatory, comma separated)</td>
</tr>
<tr>
<td>-l: --project=</td>
<td>Project to build the release for (DIRAC by default)</td>
</tr>
<tr>
<td>-D: --destination</td>
<td>Destination where to build the tar files</td>
</tr>
<tr>
<td>-i: --pythonVersion</td>
<td>Python version to use (25/26)</td>
</tr>
<tr>
<td>-P --ignorePackages</td>
<td>Do not make tars of python packages</td>
</tr>
<tr>
<td>-C: --relcfg=</td>
<td>Use &lt;file&gt; as the releases.cfg</td>
</tr>
<tr>
<td>-b --buildExternals</td>
<td>Force externals compilation even if already compiled</td>
</tr>
<tr>
<td>-B --ignoreExternals</td>
<td>Skip externals compilation</td>
</tr>
<tr>
<td>-t: --buildType=</td>
<td>External type to build (client/server)</td>
</tr>
<tr>
<td>-x: --externalsLocation=</td>
<td>Use externals location instead of downloading them</td>
</tr>
<tr>
<td>-j: --makeJobs=</td>
<td>Make jobs (default is 1)</td>
</tr>
<tr>
<td>-M: --defaultsURL=</td>
<td>Where to retrieve the global defaults from</td>
</tr>
</tbody>
</table>
2.10.78  dirac-externals-requirements

Usage:

`dirac-externals-requirements.py (<options>|<cfgFile>)*`

Options:

`-t: --type= : Installation type. 'server' by default.`

2.10.79  dirac-fix-ld-library-path

Usage:

`dirac-fix-ld-library-path.py (<options>|<cfgFile>)*`

2.10.80  dirac-install-executor

Install an executor.

Usage:

`dirac-install-executor [option|cfgfile] ... System Executor|System/Executor`

Arguments:

`System: Name of the DIRAC system (ie: WorkloadManagement)`

`Service: Name of the DIRAC executor (ie: JobPath)`

Options:

`-w: --overwrite : Overwrite the configuration in the global CS`

`-m: --module= : Python module name for the executor code`

`-p: --parameter= : Special executor option`

2.10.81  dirac-install-mysql

Install MySQL. The clever way to do this is to use the dirac-admin-sysadmin-cli.

User convenience:

2.10.82  dirac-accounting-report-cli

Command line interface to DIRAC Accounting ReportGenerator Service

Usage:

`dirac-accounting-report-cli [option|cfgfile] ...`
## 2.10.83 dirac-accounting-decode-fileid

Decode Accounting plot URLs

**Usage:**

```
dirac-accounting-decode-fileid [option|cfgfile] ... URL ...
```

**Arguments:**

- **URL:** encoded URL of a DIRAC Accounting plot

## 2.10.84 dirac-cert-convert.sh

From a p12 file, obtain the pem files with the right access rights. Needed to obtain a proxy. Creates the necessary directory, `$HOME/.globus`, if needed. Backs-up old pem files if any are found.

**Usage:**

```
dirac-cert-convert.sh CERT_FILE_NAME
```

**Arguments:**

- **CERT_FILE_NAME:** Path to the p12 file.

## 2.10.85 dirac-myproxy-upload

**Usage:**

```
dirac-myproxy-upload.py (<options>|<cfgFile>)*
```

**Options:**

- `-f`  `--file=` : File to use as proxy
- `-D`  `--DN` : Use DN as myproxy username
- `-i`  `--version` : Print version

## 2.10.86 dirac-utils-file-adler

Calculate alder32 of the supplied file

**Usage:**

```
dirac-utils-file-adler [option|cfgfile] ... File ...
```

**Arguments:**

- **File:** File Name

**Example:**

```
$ dirac-utils-file-adler Example.tgz
Example.tgz 88b4ca8b
```
2.10.87 dirac-utils-file-md5

Calculate md5 of the supplied file

Usage:

dirac-utils-file-md5 [option|cfgfile] ... File ...

Arguments:

File: File Name

Example:

$ dirac-utils-file-md5 Example.tgz
Example.tgz 5C1A1102-EAFD-2CBA-25BD-0EFCCFC3623E

Other commands:

2.10.88 dirac-admin-accounting-cli

Command line administrative interface to DIRAC Accounting DataStore Service

Usage:

dirac-admin-accounting-cli [option|cfgfile] ...

2.10.89 dirac-admin-get-proxy

Retrieve a delegated proxy for the given user and group

Usage:

dirac-admin-get-proxy [option|cfgfile] ... <DN|user> group

Arguments:

DN: DN of the user
user: DIRAC user name (will fail if there is more than 1 DN registered)
group: DIRAC group name

Options:

-v: --valid= : Valid HH:MM for the proxy. By default is 24 hours
-l: --limited : Get a limited proxy
-u: --out= : File to write as proxy
-a: --voms : Get proxy with VOMS extension mapped to the DIRAC group
-m: --vomsAttr= : VOMS attribute to require

Example:
2.10.90 dirac-admin-proxy-upload

Usage:

```
$ dirac-admin-proxy-upload <options>|<cfgFile>*
```

Options:

- `-v`: `--valid=`, : Valid HH:MM for the proxy. By default is one month
- `-g`: `--group=`, : DIRAC Group to embed in the proxy
- `-C`: `--Cert=`, : File to use as user certificate
- `-K`: `--Key=`, : File to use as user key
- `-P`: `--Proxy=`, : File to use as proxy
- `-f`: `--onthefly`, : Generate a proxy on the fly
- `-p`: `--pwstdin`, : Get passwd from stdin
- `-i`: `--version`, : Print version

2.10.91 dirac-admin-upload-proxy

Upload a proxy to the Proxy Manager using delegation

Usage:

```
$ dirac-admin-upload-proxy [option|cfgfile] ... Group
```

Arguments:

- `Group`: Group name in the uploaded proxy

Example:

```
$ dirac-admin-upload-proxy dirac_test
```

2.10.92 dirac-proxy-get-uploaded-info

Usage:

```
$ dirac-proxy-get-uploaded-info.py <options>|<cfgFile>*
```

Options:

- `-u`: `--user=`, : User to query (by default oneself)

Example:
2.10.93 dirac-proxy-info

Obtain detailed info about user proxies.

Usage:

dirac-proxy-info.py (<options>|<cfgFile>)*

Options:

- **f**: --file= : File to use as user key
- **i**: --version : Print version
- **n**: --novoms : Disable VOMS
- **v**: --checkvalid : Return error if the proxy is invalid
- **x**: --ncs : Disable CS
- **e**: --steps : Show steps info
- **j**: --noclockcheck : Disable checking if time is ok
- **m**: --uploadedinfo : Show uploaded proxies info

Example:

$ dirac-proxy-info
subject : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy/CN=proxy
issuer : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar/CN=proxy
identity : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar

2.10.94 dirac-proxy-init

Obtain a user proxy.

Usage:

dirac-proxy-init.py (<options>|<cfgFile>)*

Options:
-v: --valid= : Valid HH:MM for the proxy. By default is 24 hours
-g: --group= : DIRAC Group to embed in the proxy
-b: --strength= : Set the proxy strength in bytes
-1 --limited : Generate a limited proxy
-t --strict : Fail on each error. Treat warnings as errors.
-S --summary : Enable summary output when generating proxy
-C: --Cert= : File to use as user certificate
-K: --Key= : File to use as user key
-u: --out= : File to write as proxy
-x --nocs : Disable CS check
-p --pwstdin : Get passwd from stdin
-i --version : Print version
-j --noclockcheck : Disable checking if time is ok
-U --upload : Upload a long lived proxy to the ProxyManager
-P --uploadPilot : Upload a long lived pilot proxy to the ProxyManager
-M --VOMS : Add voms extension

Example:

```
$ dirac-proxy-init -g dirac_user
Enter Certificate password:
$ 
```

### 2.10.95 dirac-admin-request-summary

Usage:

```
dirac-admin-request-summary.py (<options>|<cfgFile>)*
```

Example:

```
$ dirac-admin-request-summary.py (<options>|<cfgFile>)*
{'diset': {'Waiting': 7}, 'register': {'Waiting': 2}}
```

### 2.10.96 dirac-admin-select-requests

Select requests from the request management system

Usage:
### 2.10.97 dirac-admin-sysadmin-cli

**Usage:**

```bash
dirac-admin-sysadmin-cli.py (<options>|<cfgFile>)
```

**Options:**

- `-H`: **--host=** : Target host

**Example:**

```bash
$ dirac-admin-sysadmin-cli --host dirac.in2p3.fr
DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v5r12
dirac.in2p3.fr >
```
2.10.98 dirac-admin-sort-cs-sites

Sort site names at CS in “/Resources” section. Sort can be alphabetic or by country postfix in a site name.

Usage:

```
dirac-admin-sort-cs-sites [option|cfgfile] <Section>
```

Optional arguments:

- **Section**: Name of the subsection in ‘/Resources/Sites/’ for sort (i.e. LCG DIRAC)

Example:

```
dirac-admin-sort-cs-sites -C CLOUDS DIRAC
```

sort site names by country postfix in ‘/Resources/Sites/CLOUDS’ and ‘/Resources/Sites/DIRAC’ subsection.

Options:

- **-C --country**: Sort site names by country postfix (i.e. LCG.IHEP.cn, LCG.IN2P3.fr, LCG.IHEP.su)
- **-R --reverse**: Reverse the sort order

2.10.99 dirac-configuration-cli

Command line interface to DIRAC Configuration Server

Usage:

```
dirac-configuration-cli [option|cfgfile] ...
```

2.10.100 dirac-configuration-dump-local-cache

Dump DIRAC Configuration data

Usage:

```
dirac-configuration-dump-local-cache [option|cfgfile] ...
```

Options:

- **-f --file=**: Dump Configuration data into <file>
- **-r --raw**: Do not make any modification to the data

Example:

```
$ dirac-configuration-dump-local-cache -f /tmp/dump-conf.txt
```

2.10.101 dirac-configuration-shell

Usage:

```
dirac-configuration-shell.py (<options>|<cfgFile>)*
```
2.10.102 dirac-repo-monitor

Monitor the jobs present in the repository

Usage:

```sh
dirac-repo-monitor [option|cfgfile] ... RepoDir
```

Arguments:

- **RepoDir**: Location of Job Repository

2.10.103 dirac-rss-reassign-token

Re-assign a token: if it was assigned to a human, assign it to ‘RS_SVC’ and viceversa.

Usage:

```sh
dirac-rss-reassign-token [option|cfgfile] <resource_name> <token_name> <username>
```

Arguments:

- **resource_name** (string): name of the resource, e.g. "lcg.cern.ch"
- **token_name** (string): name of a token, e.g. "RS_SVC"
- **username** (string): username to reassign the token to

2.10.104 dirac-rss-renew-token

Extend the duration of given token

Usage:

```sh
dirac-rss-renew-token [option|cfgfile] <resource_name> <token_name> [hours]
```

Arguments:

- **resource_name** (string): name of the resource, e.g. "lcg.cern.ch"
- **token_name** (string): name of a token, e.g. "RS_SVC"
- **hours** (int, optional): number of hours (default: 24)

Options:

- **-e**: --Extension= : Number of hours of token renewal (will be 24 if unspecified)

2.10.105 dirac-rss-list-status

Script that dumps the DB information for the elements into the standard output.

If returns information concerning the StatusType and Status attributes.

Usage:
2.10.106 dirac-rss-set-status

Script that facilitates the modification of an element through the command line.

However, the usage of this script will set the element token to the command issuer with a duration of 1 day.

Options:
- --element= : Element family to be Synchronized (Site, Resource or Node)
- --name= : Name, name of the element where the change applies
- --statusType= : StatusType, if none applies to all possible statusTypes
- --status= : Status to be changed
- --reason= : Reason to set the Status

2.10.107 dirac-rss-sync

Script that synchronizes the resources described on the CS with the RSS.

By default, it sets their Status to Unknown, StatusType to all and reason to Synchronized. However, it can copy over the status on the CS to the RSS. Important: If the StatusType is not defined on the CS, it will set it to Banned!

Options:
- --init : Initialize the element to the status in the CS (applicable for StorageElements)
- --element= : Element family to be Synchronized (Site, Resource or Node) or 'all'

2.10.108 dirac-rss-setup

What is this doing??

2.10.109 dirac-rss-set-token

Set the token for the given element.
Usage:

```
dirac-rss-set-token [option|cfgfile] <granularity> <element_name> <token> [reason] [status_type] [duration]
```

Arguments:

- **granularity** (string): granularity of the resource, e.g. "Site"
- **element_name** (string): name of the resource, e.g. "LCG.CERN.ch"
- **token** (string, optional): token to be assigned ("RS_SVC" gives it back to RSS), e.g. "ubeda"
- **reason** (string, optional): reason for the change, e.g. "I dont like the site admin"
- **statusType** (string, optional): defines the status type, otherwise it applies to all
- **duration** (integer, optional): duration of the token.

Options:

- `-g`: --Granularity= : Granularity of the element
- `-n`: --ElementName= : Name of the element
- `-k`: --Token= : Token of the element (write 'RS_SVC' to give it back to RSS)
- `-r`: --Reason= : Reason for the change
- `-t`: --StatusType= : StatusType of the element
- `-u`: --Duration= : Duration(hours) of the token

### 2.10.110 dirac-stager-monitor-request

**Usage:**

```
dirac-stager-monitor-request [option|cfgfile] ... Request ...
```

Arguments:

**Request**: ID of the Stage request in the StorageManager

### 2.10.111 dirac-stager-stage-files

**Usage:**

```
dirac-stager-stage-files [option|cfgfile] ... SE FileName [...]
```

Arguments:

**SE**: Name of Storage Element

**FileName**: LFN to Stage (or local file with list of LFNs)
2.10.112 install_site.sh

Options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFG_file</td>
<td>is the name of the installation configuration file which contains all the</td>
</tr>
<tr>
<td></td>
<td>instructions for the DIRAC installation. See DIRAC Administrator Guide for</td>
</tr>
<tr>
<td></td>
<td>the details</td>
</tr>
</tbody>
</table>

2.10.113 dirac-agent

Script running a dirac agent. Mostly internal.

2.10.114 dirac-executor

2013-02-06 12:30:09 UTC Framework FATAL: You must specify which executor to run!

2.10.115 dirac-compile-externals

Compile DIRAC externals (does not require DIRAC code)

Usage:

```
dirac-compile-externals [options]...
```

Options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-D: --destination=</td>
<td>Destination where to build the externals</td>
</tr>
<tr>
<td>-t: --type=</td>
<td>Type of compilation (default: client)</td>
</tr>
<tr>
<td>-e: --externalsPath=</td>
<td>Path to the externals sources</td>
</tr>
<tr>
<td>-v: --version=</td>
<td>Version of the externals to compile (default will be the latest commit)</td>
</tr>
<tr>
<td>-i: --pythonVersion=</td>
<td>Python version to compile (default 26)</td>
</tr>
<tr>
<td>-f</td>
<td>--fixLinksOnly</td>
</tr>
<tr>
<td>-j: --makeJobs=</td>
<td>Number of make jobs, by default is 1</td>
</tr>
</tbody>
</table>

2.10.116 dirac-fix-mysql-script

Fixes the mysql.server script, it requires a proper /LocalInstallation section

Usage:

```
dirac-fix-mysql-script [option] ... [cfgfile]
```
The DIRAC Developer Guide is describing procedures, rules and practical details for developing new DIRAC components. The section Development Model describes the general code management procedures, building and distribution of the DIRAC releases.

To work on the code, DIRAC developers need to set up an environment to work on the software components and to test it together with other parts of the distributed system. Setting up such an environment is discussed in Development Environment.

An overview of the DIRAC software architecture is presented in the Architecture overview section. Detailed instructions on how to develop various types of DIRAC components are given in Developing DIRAC components chapter. It gives examples with explanations, common utilities are discussed as well. More details on the available interfaces can be found in the code_documentation part.

For every question, or comment, regarding specific development activities, including suggestion and comments to the RFC, the correct forum for is the dirac-develop google group. For everything operational, instead, you can write on the dirac-grid group.

### 3.1 Development Model

The DIRAC Project is advanced collectively by a distributed team of developers working in parallel on the core software as well as on various extensions. This work must be supported by a suitable development model which is described in this chapter.

The DIRAC code development is done with the help of the Git code management system. It is inherently distributed and is well suited for the project. It is outlined the Managing Code with Git subsection.

The DIRAC Development Model relies on the excellent Git capability for managing code branches which is mandatory for a distributed team of developers. The DIRAC branching model is following strict conventions described in Branching Model subsection.

The DIRAC code management is done using the Github service as the main code repository. The service provides also facilities for bug and task tracking, Wiki engine and other tools to support the group code development. Setting up the Git based development environment and instructions to contribute new code is described in Contributing code subsection.

The DIRAC releases are described using a special configuration file and tools are provided to prepare code distribution tar archives. The tools and procedures to release the DIRAC software are explained in Making DIRAC releases subsection.
3.1.1 Managing Code with Git

DIRAC uses Git to manage its source code. Git is a Distributed Version Control System (DVCS). That means that there’s no central repository like the one CVS/Subversion use. Each developer has a copy of the whole repository. Because there are lots of repositories, code changes travel across different repositories all the time by merging changes from different branches and repositories. In any centralized VCS branching/merging is an advanced topic. In Git branching and merging are daily operations. That allows to manage the code in a much more easy and efficient way. This document is heavily inspired on A successful Git branching model.

How decentralization works

Git doesn’t have a centralized repository like CVS or Subversion do. Each developer has its own repository. That means that commits, branches, tags... everything is local. Almost all Git operations are blazingly fast. By definition only one person works with one repository directly. But people don’t develop alone. Git has a set of operations to send and bring information to/from remote repositories. Users work with their local repositories and only communicate with remote repositories to publish their changes or to bring other developer's changes to their repository. In Git lingo sending changes to a repository is called push and bringing changes is pull.

Git per-se doesn’t have a central repository but to make things easier we’ll define a repository that will hold the releases and stable branches for DIRAC. Developers will bring changes from that repository to synchronize their code with the DIRAC releases. To send changes to be released, users will have to push their changes to a repository where the integration manager can pull the changes from, and send a Pull Request. A Pull Request is telling the release manager where to get the changes from to integrate them into the next DIRAC release.

Developers use the developer private repositories for their daily work. When they want something to be integrated, they publish the changes to their developer public repositories and send a Pull Request to the release manager. The release manager will pull the changes to his/her own repository, and publish them in the blessed repository where the rest of the developers can pull the new changes to their respective developer private repositories.

In practice, the DIRAC Project is using the Github service to manage the code integration operations. This will be described in subsequent chapters.

Decentralized but centralized

Although Git is a distributed VCS, it works best if developers use a single repository as the central “truth” repository. Note that this repository is only considered to be the central one. We will refer to this repository as release since all releases will be generated from this repository.

Each developer can only pull from the release repository. Developers can pull new release patches from the release repository into their private repositories, work on a new feature, bugfix.... and then push the changes to their public repository. Once there are new changes in their public repositories, they can issue a pull request so the changes can be included in central release repository.
The precise instructions on how to create local git development repository and how to contribute code to the common repository are given in subsequent sections.

### 3.1.2 Branching Model

DIRAC release branches live in the central repository of the Github service.

#### DIRAC releases nomenclature

**Release version name conventions**

The DIRAC release versions have the form $vXrYpZ$, where $X$, $Y$ and $Z$ are incrementally increasing integer numbers. For example, $v1r0p1$. $X$ corresponds to the major release number, $Y$ corresponds to the minor release number and $Z$ corresponds to the patch number (see below). It is possible that the patch number is not present in the release version, for example $v6r7$.

The version of prereleases used in the DIRAC certification procedure is constructed in a form: $vXrY$-pre$Z$, where the $vXrY$ part corresponds to the new release being tested and $Z$ denotes the prerelease number.

Release versions are used as tags in Git terms to mark the set of codes corresponding to the given release.

**Release types**

We distinguish *releases*, *patches* and *pre-releases*. Releases in turn can be *major* and *minor*.

**major release** major releases are created when there is an important change in the DIRAC functionality involving changes in the service interfaces making some of the previous major release clients incompatible with the new services. DIRAC clients and services belonging to the same major release are still compatible even if they belong to different normal releases. In the release version the major release is denoted by the number following the initial letter “v”.

**minor release** minor releases are created whenever a significant new functionality is added or important changes of the existing functionality are done. Minor releases necessitate certification step in order to make the new code available in production for users. In the release version minor releases are denoted by the number following the letter “r”.

**patch** patches are created for a minor and/or obvious bug fixes and minor functionality changes. This is the responsibility of the developer to ensure that the patch changes only fix known problems and do not introduce new bugs or undesirable side effects. Patch releases are not subject to the full release certification procedure. Patches are applied to the existing releases. In the release version patch releases are denoted by the number following the letter “p”.

**pre-release** the DIRAC certification procedure goes through a series of *pre-releases* used to test carefully the code to be released in production. The prerelease versions have a form $vXrY$-pre$Z$.

**Release branches**

The following branches are used in managing DIRAC releases:

**integration branch** this branch is used to assemble new code developments that will be eventually released as a new major or minor release.
**release branches** the release branches contain the code corresponding to a given major or minor release. This is the production code which is distributed for the DIRAC installations. The release branches are created when a new minor release is done. The patches are incorporated into the release branches. The release branch names have the form `rel-vXrY`, where `vXrY` part corresponds to the branch minor release version.

**master branch** the master branch corresponds to the current stable production code. It is a copy of the corresponding release branch.

These branches are the only ones maintained in the central Git repository by the release managers. They are used to build DIRAC releases. They also serve as reference code used by developers as a starting point for their work.

### Feature branches

These are the branches where all the actual developments are happening. They can be started from `release/integration` and will be merged back to them eventually if the contribution are accepted. Their name should reflect the feature being developed and should not be “integration” or “master” to avoid confusions.

Feature branches are used to develop new features for a future release or making patches to the already created releases. A feature branch will exist as long as the feature is in development but will eventually be merged into `release/integration` or discarded in case the feature is no longer relevant. Feature branches exist only in the developer repositories and never in the `release` repository.

Working on and contributing code to the DIRAC Project is described in *Contributing code*.

### 3.1.3 Contributing code

The Github service is providing the Git code repository as well as multiple other services to help managing complex software projects developed by large teams. It supports a certain development process fully adopted by the DIRAC Project.

#### Contributing new code

The developers are working on the new codes following the procedure below.

**Github repository developer fork**

All the DIRAC developers must register as Github users. Once registered, they create their copies of the main DIRAC code repository, so called *forks*. Now they have two remote repositories to work with: `release` and `origin`.

**Local Git environment**

The local Git repository is most easily created by cloning the user remote Github repository. Choose the local directory where you will work on the code, e.g. `devRoot`:

```
$ git clone git@github.com:<your_github_user_name>/DIRAC.git
```

This will create DIRAC directory in `devRoot` which contains the local Git repository and a checked out code of a default branch. You can now start working on the code.

In the local Git environment developers create two “remotes” (in the Git terminology) corresponding to the two remote repositories:

- **release** this remote is pointing to the main DIRAC project repository. It can be created using the following command:
origin this remote is pointing to the DIRAC project personal fork repository of the developer. It can be created using the following command:

```
$ git remote add origin git://github.com/<your_github_user_name>/DIRAC.git
```

where the <username> is the user name of the developer in the Github service. If the local repository was created by cloning the user Github remote repository as described above, the origin remote is already created.

Note that the names of the remotes, release and origin, are conventional. But it is highly recommended to follow this convention to have homogeneous environment with other developers.

### Working on the new code

The work on the new features to be incorporated eventually in a new release should start in a local feature branch created from the current integration branch of the main DIRAC repository. Let call the new development branch “newdev”, for example. It should be created with the following commands:

```
$ git fetch release
$ git checkout -b newdev release/integration
```

This will create the new newdev branch locally starting from the current status of the main DIRAC repository. The “newdev” branch becomes the working branch.

The new codes are created in the newdev branch and when they are ready to be incorporated into the main DIRAC code base, the following procedure should be followed. First, the local development branch should receive all the new changes in the main integration branch that were added since the development branch was created:

```
$ git checkout newdev
$ git fetch release
$ git rebase --no-ff release/integration
```

This might need resolving possible conflicts following Git instructions. Once the conflicts are resolved, the newdev branch should be pushed to the developer personal Github repository:

```
$ git push origin newdev
```

Now the newly developed code is in the personal Github repository and the developer can make a Pull Request (PR) to ask its incorporation into the main integration branch. This is done using the Github service web interface. This interface is changing often since the Github service is evolving. But the procedure includes in general the following steps:

- go to the personal fork of the DIRAC repository in the Github portal
- choose the newdev branch in the branch selector
- press the “Pull Request” button
- choose the integration as the target branch of the PR
- give a meaningful name to the PR describing shortly the new developments
- give a more detailed description of the new developments suitable to be included into the release notes
- press “Submit Pull Request” button

The PR is submitted. All the developers will be notified by e-mail about the new contribution proposal, they can now review it. After the PR is reviewed, it is now up to the release manager to examine the PR and to incorporate it into the new release.
After the PR is submitted and before it is merged into the main integration branch, the developer can still add new changes to the newdev branch locally and push the changes to the origin personal remote repository, for example, following comments of the reviewers. These changes will be automatically added to the PR already submitted. After the PR is merged by the release manager into the main integration branch, it is recommended to remove the newdev branch from the remote personal repository in order to avoid conflicts with later uploads of this branch. This can be done with the following command:

```bash
git push origin :newdev
```

### Working on a patch

Making a patch is very similar to contributing the new code. The only difference is that the source and the target branch for the corresponding PR is the release branch to which the patch is meant to. For the developer it is very important to choose the right target release branch. The release branches in the main project repository are containing the code that is currently in production. Different DIRAC installations may use different releases. Therefore, the target release branch for a patch is the earliest release still in production for some DIRAC installations and for which the patch is relevant.

As a matter of reminder, here is a set of commands to make a patch. First, start with the new branch to work on the patch based on the target release branch, for example rel-v6r11 ::

```bash
git fetch release
git checkout -b fix-v6r11 release/rel-v6r11
```

Make the necessary changes to the code of the branch and then push them to the developer’s fork::

```bash
git push origin fix-v6r11
```

Do the PR with the rel-v6r11 as a target branch. Once the PR is merged, scrap the patch branch from the forked repository::

```bash
git push origin :fix-v6r11
```

The patches incorporated into a release branch will be propagated to the more recent release branches and to the integration branch by the release manager. There is no need to make separate PR’s of the same patch to other branches.

### Resolving PR conflicts

It should be stressed once again that you must choose carefully the target branch where the newly developed code will go: new features must be included into the integration branch, whereas small patches are targeted to relevant release branches. Once the choice is made, start the feature branch from the eventual target branch.

Even when preparing a PR you follow the procedure described above, there is no guarantee that there will be no conflicts when merging the PR. You can check if your PR can be merged on the Github page for Pull Requests of the DIRACGrid project. In case of conflicts, the release manager will ask you to find and fix conflicts made by your PR. Assuming you have a local clone of your DIRAC repository and the new code was developed in the featurebranch, you have to try merge it by hand to find and understand the source of conflicts. For that you should first checkout your feature branch, and try to rebase your branch on the target branch, release or integration::

```bash
$ git checkout featurebranch
Switched to branch 'featurebranch'
$ git fetch release
remote: Counting objects: 1366, done.
remote: Compressing objects: 100% (528/528), done.
remote: Total 1138 (delta 780), reused 952 (delta 605)
Receiving objects: 100% (1138/1138), 334.89 KiB, done.
```
Resolving deltas: 100% (780/780), completed with 104 local objects.

From git://github.com/DIRACGrid/DIRAC
  * [new branch] integration -> DIRAC/integration
  * [new branch] master -> DIRAC/master
  * [new tag] v6r0-pre1 -> v6r0-pre1
  * [new tag] v6r0-pre2 -> v6r0-pre2

From git://github.com/DIRACGrid/DIRAC
  * [new tag] v6r0-pre3 -> v6r0-pre3

$ git rebase release/integration
First, rewinding head to replay your work on top of it...
Applying: added .metadata to .gitignore
Using index info to reconstruct a base tree...
Falling back to patching base and 3-way merge...
Auto-merging .gitignore
CONFLICT (content): Merge conflict in .gitignore
Failed to merge in the changes.
Patch failed at 0001 added .metadata to .gitignore

When you have resolved this problem run "git rebase --continue".
If you would prefer to skip this patch, instead run "git rebase --skip".
To restore the original branch and stop rebasing run "git rebase --abort".

On this stage git will tell you which changes cannot be merged automatically, in above example there is only one conflict in .gitignore file. Now you should open this file and find all conflict markers (“>>>>>>” and “<<<<<<<<<”), edit it choosing which lines are valid. Once all conflicts are resolved and necessary changes are committed, you can now push your feature branch to your remote repository:

git push origin featurebranch

The fixes will be automatically taken into account, you do not need to recreate the Pull Request.

3.1.4 DIRAC Projects

DIRAC is used by several user communities. Some of them are creating their own modules for DIRAC. These modules require a certain version of DIRAC in order to function properly. Virtual organizations have to be able to create their own releases of their modules and install them seamlessly with dirac-install. This is achieved by creating and releasing software projects in the DIRAC framework.

Preparing DIRAC distribution

Releases schema

DIRAC modules are released and distributed in projects. Each project has a releases.cfg configuration file where the releases, modules and dependencies are defined. A single releases.cfg can take care of one or more modules. releases.cfg file follows a simplified schema of DIRAC’s cfg format. It can have several sections, nested sections and options. Section Releases contains the releases definition. Each section in the Releases section defines a release. The name of the section will be the release name. Each release will contain a list of dependencies (if any) and a list of modules (if more than one). An example of a release.cfg for a single module is shown below:

DefaultModules = MyExt

Sources
{
  MyExt = git://somerepohosting/MyExt.git
}

3.1. Development Model
Releases
{
  v1r2p3
  {
    depends = DIRAC:v5r12
  }
  v1r2p2
  {
    depends = DIRAC:v5r12p1
  }
}

The DefaultModules option (outside any section) defines what modules will be installed by default if there’s nothing explicitly specified at installation time. Because there is only one module defined in DefaultModules each release will try to install the MyExt module with the same version as the release name. Each release can require a certain version of any other project (DIRAC is also an project).

An example with more than one module follows:

DefaultModules = MyExt
RequiredExtraModules = Web

Sources
{
  MyExt = git://somerepohosting/MyExt.git
  MyExtExtra = svn | http://someotherrepohosting/repos/randomname/MyExtExtra/tags
}

Releases
{
  v1r2p3
  {
    Modules = MyExt:v1r2p1, MyExtExtra:v1r1p1
    Depends = DIRAC:v5r12p1
  }
  v1r2p2
  {
    Modules = MyExt:v1r2p1, MyExtExtra:v1r1
    Depends = DIRAC:v5r12
  }
}

If a project requires a module that is not installed by default from another project to be installed, it can be defined in the RequiredExtraModules option. For instance, DIRAC project contains DIRAC and Web. But by default DIRAC project only installs DIRAC module. If another project requires the DIRAC Web module to be installed it can be defined in this option. That way, when installing this other project, Web module will also be installed.

The Modules option can define explicitly which modules (and their version) to install. This is useful if a given VO is managing more than one module. In that scenario a release can be a combination of modules that can evolve independently. By defining releases as groups of modules with their versions the VO can ensure that a release is consistent for its modules. DIRAC uses this mechanism to ensure that the DIRAC Web will always be installed with a DIRAC version that it works with.

The Sources section defines where to extract the source code from for each module. dirac-distribution will assume that there’s a tag in that source origin with the same name as the version of the module to be released. dirac-distribution knows how to handle several types of VCS. The ones supported are:
**file** A directory in the filesystem. *dirac-distribution* will assume that the directory specified contains the required module version of the module.

**cvs** The cvss root where to find the code. *dirac-distribution* will assume there’s a tag with the same name name as the module version to be tagged.

**svn** A subversion url that contains a directory with the same name as the version to be tagged. If the module version is v1r0 and the url is http://host/extName, *dirac-distribution* will check out http://host/extName/v1r0 and assume it contains the module contents.

**hg** A mercurial repository. *dirac-distribution* will check out the a tag with the same name as the module version and assume it contains the module contents.

**git** A git repository. *dirac-distribution* will clone the repository and check out to a tag with the same name as the module version and assume it contains the module contents.

Some of the VCS URLs may not explicitly define which VCS has to be used (for instance http://... it can be a subversion or mercurial repository). In that case the option value can take the form `<vcsName> | <vcsURL>`. In that case *dirac-distribution* will use that VCS to check out the source code.

When installing, a project name can be given. If it is given *dirac-install* will try to install that project instead of the DIRAC project. *dirac-install* will have a mapping to discover where to find the releases.cfg based on the project name. Any VO can modify *dirac-install* to directly include their repositories inside *dirac-install* in their module source code, and use their modified version. DIRAC developers will also maintain a project name to releases.cfg location mapping in the DIRAC repository. Any VO can also notify the DIRAC developers to update the mapping in the DIRAC repository so *dirac-install* will automatically find the project’s releases.cfg without any change to *dirac-install*.

If a project is given, all modules inside that releases.cfg have to start with the same name as the project. For instance, if *dirac-install* is going to install project LHCb, all modules inside LHCb’s releases.cfg have to start with LHCb.

*dirac-distribution* will generate a set of tarballs, md5 files and a release-<projectName>-<version>.cfg. Once generated, they have to be upload to the install project source of tarballs where *dirac-install* will try to pick them up.

**How to define how to make a project distribution**

*dirac-distribution* needs to know where to find the releases.cfg file. *dirac-distribution* will load some global configuration from a DIRAC web server. That configuration can instruct *dirac-distribution* to load the project defaults file from a URL. Those defaults will define default values for *dirac-distribution* and *dirac-install* command line options. An example of a project defaults file would be:

```
#Where to load the release.cfg file from
Releases = https://github.com/DIRACGrid/DIRAC/raw/integration/releases.cfg
#Where to download the released tarballs from
#How to upload the release tarballs to the BaseURL
UploadCommand = ( cd %OUTLOCATION% ; tar -cf - %OUTFILENAMES% ) | ssh webuser@webhost 'cd /webroot/lhcbproject/dist/DIRAC3/installSource && tar -xvf - && ls *.tar.gz > tars.list'
```

Once the tarballs and required files have been generated by *dirac-distribution* (see below), if *UploadCommand* is defined the variables will be substituted and the final command printed to be executed by the user.

*dirac-install* will download the project files from the BaseURL location.

The defaults file is defined per project and can live in any web server.

---

**3.1. Development Model**
Installation

When installing, `dirac-install` requires a release version and optionally a project name. If the project name is given, `dirac-install` will try to load the project’s versioned `release-<projectName>-<version>.cfg` instead of the DIRAC’s one (this file is generated by `dirac-distribution` when generating the release). `dirac-install` has several mechanisms on how to find the URL where the released tarballs and releases files for each project are. `dirac-install` will try the following steps:

1. Load DIRAC’s default global locations. This file contains the default values and paths for each project that DIRAC knows of and it’s maintained by DIRAC developers.
2. Load the required project’s defaults file. DIRAC’s default global locations has defined where this file is for each project. It can be in a URL that is maintained by the project’s developers/maintainers.
3. If an option called `BaseURL` is defined in the project’s defaults file then use that as the base URL to download the releases and tarballs files for the projects.
4. If it’s defined inside `dirac-install`, use it.
5. If not found then the installation is aborted.

The `release-<projectName>-<version>.cfg` file will specify which module and version to install. All modules that are defined inside a `release-<projectName>-<version>.cfg` will be downloaded from the same parent URL. For instance, if the `release-<projectName>-<version>.cfg` is in `http://diracgrid.org/releases/releases.cfg` and DIRAC v5r14 has to be installed, `dirac-install` will try to download it from `http://diracgrid.org/releases/DIRAC-v5r14.tar.gz`.

If nothing else is defined, `dirac-install` will only install the modules defined in `DefaultModules` option. To install other modules that are defined in the `release-<projectName>-<version>.cfg` the `-e` flag has to be used.

Once all the modules defined in the `release-<projectName>-<version>.cfg` are installed. `dirac-install` will try to load the dependencies. The `depends` option defines on which projects the installed project depends on. That will trigger loading that `release-<projectName>-<version>.cfg` and process it as the main one was processed. `dirac-install` will try to resolve recursively all the dependencies either until all the required modules are installed or until there’s a mismatch in the requirements. If after resolving all the `release-<projectName>-<version>.cfg` an module is required to be installed with more than one version, an error will be raised and the installation stopped.

The set of parameters used to install a project is called an installation. `dirac-install` also has support for installations. Each installation is a set of default values for `dirac-install`. If the `-V` switch is used `dirac-install` will try to load the defaults file for that installation and use those defaults for the arguments.

Reference of releases.cfg schema

```plaintext
#List of modules to be installed by default for the project
DefaultModules = MyExt

#Extra modules to be installed
RequiredExtraModules = Web

#Section containing where to find the source code to generate releases
Sources
{
    #Source URL for module MyExt
    MyExt = git://somerepohosting/MyExt.git
    MyExtExtra = svn | http://someotherrepohosting/repos/randomname/MyExtExtra/tags
}

#Section containing the list of releases
```

306 Chapter 3. Developer Guide
Releases
{
  #Release v1r2p3
  v1r2p3
  {
    # (Optional) Contains a comma separated list of modules for this release and their version in format
    # *extName(:extVersion)? (, extName(:extVersion)?)**.
    # If this option is not defined, modules defined in *DefaultExtensions* will be installed
    # with the same version as the release.
    Modules = MyExt:v1r2p1, MyExtExtra:v1r1p1
    
    # (Optional) Comma separated list of projects on which this project depends in format
    # *projectName(:projectVersion)? (, projectName(:projectVersion)?)**.
    # Defining this option triggers installation on the depended project.
    # This is useful to install the proper version of DIRAC on which a set of modules depend.
    Depends = DIRAC:v5r12p1
  }
}

v1r2p2
{
  Modules = MyExt:v1r2p1, MyExtExtra:v1r1
}

Reference of an installation's defaults file

# (Everything in here is optional) Default values for dirac-install
LocalInstallation
{
  # Install the requested project instead of this one
  # Useful for setting defaults for VOs by defining them as projects and
  # using this feature to install DIRAC instead of the VO name
  Project = DIRAC
  # Release to install if not defined via command line
  Release = v1r4
  # Modules to install by default
  ModulesToInstall = MyExt
  # Type of externals to install (client, client-full, server)
  ExternalsType = client
  # Python version to install (25/26)
  PythonVersion = 26
  # Version of lcg bundle to install
  LcgVer = 2010-11-20
  # Install following DIRAC's pro/versions schema
  UseVersionDir = False
  # Force building externals
  BuildExternals = False
  # Build externals if the required externals is not available
  BuildIfNotAvailable = False
  # Enable debug logging
  Debug = False
}

3.1. Development Model
Reference of global default's file

Global defaults is the file that `dirac-install` will try to load to discover where the each project's `defaults.cfg` file is. The schema is as follows:

```
Projects
{
    #Project name
    ProjectName
    {
        #Where to find the defaults
        DefaultsLocation = http://somehost/somepath/defaultsProject.cfg
        #Release file location
        ReleasesLocation = http://endoftheworld/releases.cfg
    }
    Project2Name
    {
        DefaultsLocation = http://someotherhost/someotherpath/chunkybacon.cfg
    }
}
Installations
{
    #Project name or installation name
    InstallationName
    {
        #Location of the defaults for this installation
        DefaultsLocation = http://somehost/somepath/defaultsProject.cfg
        #Default values for dirac-install
        LocalInstallation
        {
            #This section can contain the same as the LocalInstallation section in each project's defaults.cfg
        }
    }
    #And repeat for each installation or project
    OtherInstallation
    {
        ....
    }
    #Alias with another names
    ThisIsAnAlias = InstallationName
}
```

All the values in the defined defaults file file take precedence over the global ones. This file is useful for DIRAC maintainers to keep track of all the projects installable via native dirac-install.

**Common pitfalls**

Installation will find a given `releases.cfg` by looking up the project name. All modules defined inside a `releases.cfg` have to start with the same name as the project. For instance, if the project is `MyVO`, all modules inside have to start with `MyVO`. `MyVOWeb`, `MyVOSomething` and `MyVO` are all valid module names inside a `MyVO releases.cfg`

**3.1.5 Making DIRAC releases**

This section is describing the procedure to follow by release managers when preparing new DIRAC releases. The procedure consists of several steps:
DIRAC Documentation, Release integration

• Merge Pull Requests
• Propagate patches to downstream release
• Make release notes
• Tag release branches with release version tags
• Update the state of release and integration branches in the central repository
• Update DIRAC software project description
• Build and upload release tar files
The release steps are described in this chapter.
Starting
For simplicity and reproducibility, it’s probably a good idea to start from a fresh copy in a clean directory. This means
that, you may want to start by moving to a temporary directory and issue the following:
> mkdir $(date +”20%y%m%d”) && cd $(date +”20%y%m%d”)
which will create a clean directory with today’s date. We then clone the DIRAC repository and rename the created
“origin” remote in “release”:
> git clone git@github.com:DIRACGrid/DIRAC.git > git remote rename origin release
Merging Pull Requests
The new code and patch contribution are made in the form of Github Pull Request. The PR are provided by the
developers and are publicly available on the Web. The PR‘s should be first reviewed by the release managers as well
as by other developers to possibly spot evident problems ( relevance of the new features, conventions, typos, etc ).
After the review the PR can be merged using the Github tools. After that the remote release branch is in the state ready
to be tagged with the new version.
Propagating patches
In the DIRAC Development Model several release branches can coexist in production. This means that patches applied
to older branches must be propagated to the newer release branches. This is done in the local Git repository of the
release manager. Let’s take an example of a patch created against release branch rel-v6r10 while the new release
branch rel-v6r11 is already in production. This can be accomplished by the following sequence of commands:
> git fetch release

This will bring all the changes from the central repository including all the release branches.:
> git checkout -b rel-v6r10 release/rel-v6r10
> vim release.notes

We create local branch from the the remote one containing the patch. Release notes must be updated to create a new
section for the new patch release describing the new changes. Now we can make a local branch corresponding to a
downstream branch and merge the commits from the patches:
> git checkout -b rel-v6r11 release/rel-v6r11
> git merge --no-ff rel-v6r10

Note that if the release branches already exist in the repository, they can be rebased on the remote counterparts instead
of recreating them::

3.1. Development Model

309


DIRAC Documentation, Release integration

> git fetch release
> git checkout rel-v6r10
> git rebase --no-ff release/rel-v6r10

This will bring the patches into the local release branch, you can now update the release notes and proceed with tagging and uploading. All the patches must be also propagated to the integration branch:

> git checkout -b integration release/integration
> git merge --no-ff rel-v6r11

Release notes

Release notes are contained in the release.notes file. Each release version has a dedicated section in this file, for example:

[v6r10p1]

  *Core
  BUGFIX: typo in the dirac-install script

  *WMS
  CHANGE: JobAgent - handle multi-core worker nodes

The section title as taken into the square brackets. Change notes are collected per subsystem denoted by a name starting with *. Each change record starts with one of the follow header words: FIX:, BUGFIX:, CHANGE:, NEW: for fixes, bug fixes, small changes and new features correspondingly.

Release notes for the given branch should be made in this branch.

Tagging and uploading release branches

Once the local release and integration branches have got all the necessary changes they can be tagged with the new version tags:

> git checkout rel-v6r10
> git tag v6r10p11
> git checkout rel-v6r11
> git tag v6r11p1

Note that if even the patch was made to an upstream release branch, the subsequent release branch must also receive a new patch release tag. Multiple patches can be add in one release operation. If the downstream release branch has got its own patches, those should be described in its release notes under the v6r11p1 section.

Once the tags are done, the updated branches and new tags must be pushed to the central repository:

> git push --tags release rel-v6r10
> git push --tags release rel-v6r11

Note that we have not yet pushed the integration branch. We have to update first the releases.cfg file with the description of dependencies on the new versions of the DIRAC modules (see DIRAC Projects).

The integration branch is also receiving new features to go into the next release. Therefore, it is used to tag prerelease versions that can be then installed with standard tools on test DIRAC servers, for example:

> git checkout integration
> git tag v7r0-pre12
After the `releases.cfg` file is updated in the `integration` branch and prerelease tags are made, the branch can be pushed in the usual way

```bash
> git push --tags release integration
```

### How to make a distribution

Once the release branches are tagged and pushed, the new release versions are properly described in the `release.cfg` file in the `integration` branch and also pushed to the central repository, the tar archives containing the new codes can be created. Just execute `dirac-distribution` command with the appropriate flags. For instance:

```bash
dirac-distribution -r v6r10p11 -l DIRAC
```

You can also pass the `releases.cfg` to use via command line using the `-C` switch. `dirac-distribution` will generate a set of tarballs, release and md5 files. Please copy those to your installation source so `dirac-install` can find them.

The command will compile tar files as well as release notes in html and pdf formats. In the end of its execution, the `dirac-distribution` will print out a command that can be used to upload generated release files to a predefined repository (see [DIRAC Projects](https://dirac-project.org)).

### 3.2 Development Environment

This chapter describes the way to set up the environment for developing DIRAC software.

#### 3.2.1 Setting up a development installation

We’ll do a step-by-step set up of what we call a developer installation. A developer installation is a closed installation: an installation that can even be used while being disconnected from the Internet.

**What is this for?**

Here we describe the suggested method for developing DIRAC server and client code, and direct extensions of it.

**What is this NOT for?**

- This method is NOT for the WebAppDIRAC code development.
- This method is NOT for the Pilot code development, although few things may apply.
- This method will NOT work out of box if you need 3rd party python libraries that are not on PyPi.
- This method will NOT work out of box if you need to contact non-DIRAC services

**Notes before continuing**

*OS:* any *nix should be fine (maybe even windows is fine but I would not know how). Examples below are given for Linux (and specifically, the writer used Ubuntu)

*shell:* examples below are given in bash, and are proven to work also in zsh. Any csh like tcsh should not pose a problem.
repository and as already explained, DIRAC’s GIT repositories are hosted on GitHub, for which you need to have an account before continuing.

OS, more notes: a DIRAC server can be installed, as of today, only on SLC6/CC7 OS. The reason being some binaries are proved to work only there, and this includes several DMS (Data Management) libraries. If you have to do many DMS development, you should consider using SLC6 or CC7.

Stuff you need to have installed

python: make sure python 2.7.x is installed and set as default (beware: latest Ubuntu use python 3.5 as default).

python-pip: the tool for installing python packages hosted on PyPi.

git: DIRAC’s version control system of choice is git, so install it.

basic packages: you will need at least gcc, python-devel (python all-dev), openssl-devel (libssl-dev), mysql-client, libmysqlclient-dev, libfreetype6-dev, libncurses5-dev, libjpeg-dev. The names above are OS-dependant, distribution dependant, and version dependant, so you’ll need to figure it out by yourself how to install them.

editor: get your favourite one. Examples include IDE like Eclipse or PyCharm, or whatever you prefer (vim, sublime, atom...) - anyway you’ll need some plugins!

Setting up your development installation

The following steps will try to guide you on setting up a development installation for DIRAC

Checking out the source

0. Go to a clean directory, e.g. $HOME/pyDevs/.

From now on we will call that directory $DEVROOT

1. export DEVROOT=$PWD && export WORKSPACE=$PWD

(persist this in the way you prefer)

2. Check out DIRAC source code. DIRAC source is hosted on github.com. Fork it, then:

git clone https://github.com/fstagni/DIRAC.git

This will create a $DEVROOT/DIRAC for you and the git repository will be cloned in.

3. This will create a remote pointer (in git terms) in the local git repository called origin that points to your source repository on GitHub. In that repository you will publish your code to be released. But all the releases will be done from the https://github.com/DIRACGrid/DIRAC repository. You need to define a remote for that repository to be able to pull newly released changes into your working repo. We will name that repository release:

```bash
    cd DIRAC
    git remote add release https://github.com/DIRACGrid/DIRAC.git
    git fetch release
```

Repository structure

Just looking at the root directory:

```bash
    ls -al $DEVROOT/DIRAC/
```

will tell you a lot about the DIRAC code structure. Note that:
• there is a tests/ directory
• there is a docs/ directory
• there are several *System/ directories
• there is an __init__.py file
• there are some base files (README, LICENCE, etc.) and some dotfiles

Unsurprisingly:
• “tests” contains tests - and specifically, it contains all the non-unit tests
• “docs” contains... documentation (including this very same page!)
• all the *System/ directories contain the (python) code of the DIRAC systems

Adding an extension

You can add an extension of DIRAC, of course. The repository structure may be the same of the DIRAC, or something slightly different. The only important thing is what you are going to put in the $PYTHONPATH.

Installing the dependencies

First first, be sure setuptools is at the latest version:

[sudo] pip install --upgrade setuptools

We’ll use virtualenv and virtualenvwrapper for working in a separate virtual python environment, and for creating and deleting such environments:

[sudo] pip install virtualenv
[sudo] pip install virtualenvwrapper
export WORKON_HOME=~/.Envs
mkdir -p $WORKON_HOME
source /usr/local/bin/virtualenvwrapper.sh

Now, let’s create the virtual environment, and populate it:

mkvirtualenv DIRACGrid
pip install -r $DEVROOT/DIRAC/requirements.txt

This will create a virtual python environment in which we can install all python packages that DIRAC uses (this may take a while, and you might need to manually install some package from your distribution).

Some useful commands:

"pip install --upgrade" will upgrade the packages
"deactivate" will exit from a virtualenv
"workon DIRACGrid" will get you back in DIRACGrid virtualenv

Adding to the PYTHONPATH

Don’t forget to:

export PYTHONPATH=$PYTHONPATH:$DEVROOT

3.2. Development Environment
And repeat for the extension development root.

And now you should be able to do:

```
ipython
In [1]: import DIRAC
In [2]: import GSI
```

If the above fails, check the log of the pip installations you just done.

**Deploy DIRAC scripts**

By running:

```
$DEVROOT/DIRAC/Core/scripts/dirac-deploy-scripts.py
```

It is a good idea to add the scripts directory to your $PATH.

**Configure DIRAC**

We’ll configure DIRAC to work in isolation. At this point, the key becomes understanding how the DIRAC
Configuration Service (CS) works. I’ll explain here briefly. The CS is a layered structure: whenever you access a CS
information (e.g. using a “gConfig” object, see later), DIRAC will first check into your local “dirac.cfg” file (it can be
in your home as .dirac.cfg, or in etc/ directory, see the link above). If this will not be found, it will look for such info
in the CS servers available.

When you develop locally, you don’t need to access any CS server: instead, you need to have total control. So, you
need to work a bit on the local dirac.cfg file. There is not much else needed, just create your own etc/dirac.cfg. The
example that follows might not be easy to understand at a first sight, but it will become easy soon. The syntax is
extremely simple, yet verbose: simply, only brackets and equalities are used.

If you want to create an isolated installation just create a $DEVROOT/etc/dirac.cfg file with (create the etc directory
first):

```
DIRAC
  { 
    Setup = DeveloperSetup 
    Setups 
    { 
      DeveloperSetup 
      { 
        Framework = DevInstance 
        Test = DevInstance 
      } 
    } 
  } 
```

Systems 
{ 
  Framework 
  { 
    DevInstance 
    { 
      URLs 
      { 
      } 
      Services 
      { 
      } 
    } 
  } 
}
Certificates

DIRAC understands certificates in pem format. That means that certificate set will consist of two files. Files ending in cert.pem can be world readable but just user writable since it contains the certificate and public key. Files ending in key.pem should be only user readable since they contain the private key. You will need two different sets certificates and the CA certificate that signed the sets. The following commands should do the trick for you, by creating a fake CA, a fake user certificate, and a fake host certificate:

```
cd $DEVROOT/DIRAC
git checkout release/integration
source tests/Jenkins/utilities.sh
generateCertificates
generateUserCredentials
mkdir -p ~/.globus/
```
Now we need to register those certificates in DIRAC. To do you you must modify $DEVROOT/etc/dirc.cfg file and set the correct certificate DNs for you and your development box. For instance, to register the host replace "your/box/dn/goes/here" (/Registry/Hosts/mydevbox/DN option) with the result of:

```
openssl x509 -noout -subject -in etc/grid-security/hostcert.pem | sed 's:^subject= ::g'
```

You’re ready for DIRAC development!

### 3.2.2 Configuring Eclipse

A possible way to develop in DIRAC is using Eclipse. We know Emacs is a great tool, and someone can’t just leave without it. And that also vim is great. Other excellent options include pycharm, sublime, and atom. Anyway, this is a quick guide for Eclipse in case you want to use it, but it is not mandatory to use it.

As it has been true for the previous paragraph, this guide will NOT explain the basic usage of eclipse and its plugins: go on the net and search. In a nutshell, based on experience we can say that:

- Eclipse is a good tool. It will simplify your life.
- There are many Eclipse versions out there: just take the classic one, and take the latest package.
- If you have a very old machine, Eclipse will eat your memory. If you are on Linux, use the Oracle JVM.
- You’ll have to update eclipse and its plugins manually (e.g. “Help” -> “Check for updates”).

Two extensions are required for developing DIRAC with Eclipse. To install them go to Help->Install new software- >top right button “Add...” -> Insert name and URL and then select the software to install in the list.

- pyDev: Use [http://pydev.org/updates](http://pydev.org/updates) as the URL to install from. For more info go Pydev updates page.

- EGit: Git team provider for eclipse. The latest of Eclipse come with this extension ready to be installed. If this is not the case, use [http://download.eclipse.org/egit/updates](http://download.eclipse.org/egit/updates) as the URL.

For more info go [Eclipse site](http://www.eclipse.org).

Now you need to configure the pyDev plugin. Go to Window->Preferences (Eclipse->preferences if you’re in a MacOSX box). In the preferences pane go to Pydev->Editor, select 2 as the tab length and click “Replace tabs with spaces when typing”. In Pydev->Editor->Code Style->Code formatter check all the boxes.

For EGit you simply need to configure your name and mail. Go to the preferences pane and then go to Team->Git- >Configuration and add two entries: user.name with your name and user.email with your email.

That’s it! Eclipse is configured now :)

### 3.2.3 Creating a development installation in Eclipse

All that remains is to import these directories as projects in Eclipse. To import DIRAC:

1. File -> Import...
2. Git -> Projects from Git and click Next.
3. In the “Import Projects from Git” click Add.
4. In the “Add Git Repositories”, click Browse and select the DIRAC source code folder you cloned into before. Then click Search and the .git directory in the DIRAC source code directory should appear. Select it and click OK.
5. In the “Import Projects from Git” pane the DIRAC folder should now appear. Select it and click Next.

Create as a general project

6. Select “Import as General Project” and click Next.

7. In the “Import Project from Git” write the project name of your Workspace and then click Finish.

Create as a pydev project (may not work in all versions of eclipse/pydev)

6. Select “Use the New Project wizard” and click Finish.

7. In the New Project wizard choose Pydev -> Pydev project and click Next.

8. Choose the necessary settings for the project, in particular:
   • Project name, e.g. DIRAC_dev
   • The project working directory
   • Use Python 2.6 grammar
   • Choose the python interpreter (you might need to set it up in a separate form)
   • Uncheck creation of src directory

9. Click Finish.

If you want to add DIRACWeb to eclipse repeat the same steps with the Web source directory. For additional extensions, add them as projects to Eclipse. You’ll have to look on how to do it depending on your team provider. For instance, if you are using subversion for your extension:

1. Go to the SVN Repository Exploring exploring perspective
2. In the SVN repositories panel, right click -> New -> Repository Location and fill in the details for your repository
3. Once the repository appears in the SVN repositories panel, browse it until you find the extension directory.
4. Once you find the extension directory -> right click -> Find/Check out as...
5. Select Check out as project with the name specified and fill in the extension name (name ending with DIRAC). For instance LHCbDIRAC
6. Click next
7. Uncheck Use default workspace location and browse to the directory where DIRAC is installed. If DIRAC is in /some/path/DIRAC, select /some/path
8. Click finish

That’s it! You have a nice development installation set up :)

**Setting up a working set for the DIRAC workspace**

Eclipse can manage several projects and developers may need have more than one development installation. It is useful to set up a view per installation. To define different views for each installation we will use Eclipse’s working sets. A working set is nothing more than a group of projects. By defining a working set Eclipse can hide the rest of the projects so only the projects in the current working set are shown. To define a working set:

1. Click on the small arrow on the package explorer and then on Select working set...
2. Click on New... and then select Resource and then click Next>

3. Give it a meaningful name and select all the projects you want to include in the working set and click Finish
4. Now the new working set will appear. If you want to activate it just select it and click OK
Now, to change the active working set or to disable them:

1. Click again on the small arrow on the package explorer and then on Select working set... as before.
2. Select the working set you want to activate or select No working sets to deactivate them.

### 3.3 Architecture overview

Most of the computing resources needed by the LHC HEP experiments as well as for some other communities are provided by Computing Grids. The Grids provide a uniform access to the computing and storage resources which simplifies a lot their usage. The Grid middleware stack offers also the means to manage the workload and data for the users. However, the variety of requirements of different Grid User Communities is very large and it is difficult to meet everybody’s needs with just one set of the middleware components. Therefore, many of the Grid User Communities, and most notably the LHC experiments, have started to develop their own sets of tools which are evolving towards complete Grid middleware solutions. Examples are numerous, ranging from subsystem solutions (PANDA workload management system or PHEDEX data management system) or close to complete Grid solutions (AliEn system). DIRAC project is providing a complete Grid solution for both workload and data management tasks on the Grid.
Although developed for the LHCb experiment, it is designed to be a generic system with LHCb specific features well isolated as plugin modules. It allows to construct medium sized grids of up to several tens of thousands processors by uniting PC farms with most widely used cluster software systems as well as individual PCs within its integrated Workload Management System. DIRAC also provides means for managing tasks on Grid resources taking over the workload management functions. The DIRAC Data Management components provide access to standard grid storage systems based on the SRM standard interface or ordinary (S)FTP, HTTP file servers. The File Catalog options include the LCG File Catalog (LFC) as well as a native DIRAC File Catalog. The modular organization of the DIRAC components allows selecting a subset of the functionality suitable for particular applications or easily adding the missing functionality. All these features provide a Grid solution for a medium size community of users.

The DIRAC architecture consists of numerous cooperating Distributed Services and Light Agents built within the same DISET framework following the Grid security standards.

DIRAC introduced the now widely used concept of Pilot Agents. This allows efficient Workload Management Systems (WMS) to be built. The workload of the community is optimized in the central Task Queue. The WMS is carefully designed to be resilient to failures in the ever changing Grid environment.

The DIRAC project includes a versatile Data Management System (DMS) which is optimized for reliable data transfers. The DMS automates the routine data distribution tasks.

The DIRAC Transformation Management System is built on top of the Workload and Data Management services. This provides automated data driven submission of processing jobs with workflows of arbitrary complexity.

The DIRAC Project has all the necessary components to build Workload and Data management systems of varying complexity. It offers a complete and powerful Grid solution for other user grid communities.

### 3.3.1 DIRAC design principles

- DIRAC is conceived as a light grid system.
- Following the paradigm of a Services Oriented Architecture (SOA), DIRAC is lightweight, robust and scalable. This was inspired by the OGSA/OGSI “grid services” concept and the LCG/ARDA RTAG architecture blueprint.
- It should support a rapid development cycle to accommodate ever-evolving grid opportunities.
- It should be easy to deploy on various platforms and updates in order to bring in bug fixes and new functionalities should be transparent or even automatic.
- It is designed to be highly adaptable to the use of heterogeneous computing resources available to the LHCb Collaboration.
- It must be simple to install, configure and operation of various services. This makes the threshold low for new sites to be incorporated into the system.
- The system should automate most of the tasks, which allows all the DIRAC resources to be easily managed by a single Production Manager.
- Redundancy
  - The information which is vital to the successful system operation is duplicated at several services to ensure that at least one copy will be available to client request. This is done for the DIRAC Configuration Service and for the File Catalog each of which has several mirrors kept synchronized with the master instance.
- All the important operations for which success is mandatory for the functioning of the system without losses are executed in a failover recovery framework which allows retrying them in case of failures. All the information necessary for the operation execution is encapsulated in an XML object called request which is stored in one of the geographically distributed request databases.
  - For the data management operations, for example for initial data file uploading to some grid storage, in case of failure the files are stored temporarily in some spare storage element with a failover request to move the data to the final destination when it becomes available.
• **System state information**
  
  – Keeping the static and dynamic information separately reduces the risk of compromising the static information due to system overloading.

  – In DIRAC the static configuration data is made available to all the clients via the Configuration Service (CS) which has multiple reservations. Moreover, this information can be cached on the client side for relatively short periods without risk of client misbehaviour.

  – The dynamic information is in most cases looked for at its source. This is why, for example, the DIRAC Workload Management System is following the “pull” paradigm where the computing resources availability is examined by a network of agents running in close connection to the sites.

• **Requirements to sites**
  
  – The main responsibility of the sites is to provide resources for the common use in a grid. The resources are controlled by the site managers and made available through middleware services (Computing and Storage Elements).

  – DIRAC puts very low requirements on the sites asking for no special support for the LHCb VO. The data production activity requires no special support from the site managers apart from ensuring availability of the standard services. There is also no special requirement on VO job optimization and accounting.

  – All this allows for the exploitation of numerous sites providing resources to the LHCb VO by a small central team of production managers.

### 3.3.2 DIRAC Architecture

DIRAC follows the paradigm of a Services Oriented Architecture (SOA).

**Figure 1. DIRAC Architecture Overview**

The **DIRAC components can be grouped in the following 4 categories:**

- Resources
- Services
- Agents
- Interfaces

**Resources**

DIRAC covers all the possible resources available to the LHCb experiment, if necessary, new types of the computing resources can be easily added:

- Individual PCs
- Computing farms with various batch systems: PBS/Torque, LSF, Sun Grid Engine, Condor, BQS and Microsoft Compute Cluster.

- Computing Elements in the EGEE grid which are based on the GRAM interface.

DIRAC does not provide a complex Storage Element service capable of managing multiple disk pools or tertiary storage systems. Storage Element can be:

- Disk storage managed by a POSIX compliant file system.
- Storage Elements wit the SRM standard interface: gridftp, (s)ftp, http, and some others.
Sometimes the same physical storage is available through several different protocols. This can be expressed in the storage configuration description and the DIRAC data access tools will be able to use any of the possible protocols in an optimal way. This also adds redundancy ensuring higher storage availability in case of intermittent failures.

**Services**

- The DIRAC system is built around a set of loosely coupled services which keep the system state and help to carry out workload and data management tasks. The services are passive components which are only reacting to the requests of their clients possibly soliciting other services in order to accomplish the requests.
- All services and their clients are built in the DISET framework which provides secure access and flexible authorization rules. Each service has typically a MySQL database backend to store the state information. The services as permanent processes are deployed centrally at CERN and on a number of hosts (VO-boxes) at several sites.
- The number of sites where services are installed is limited to those with well-controlled environment where an adequate support can be guaranteed. The services are deployed using system start-up scripts and watchdog processes which ensure automatic service restart at boot time and in case of service interruptions or crashes. Standard host certificates typically issued by national Grid Certification Authorities are used for the service/client authentication.
- The services accept incoming connections from various clients. These can be user interfaces, agents or running jobs. But since services are passive components, they have to be complemented by special applications to animate the system.

**Agents**

Agents are light and easy to deploy software components which run as independent processes to fulfill one or several system functions.

- All the agents are built in the same framework which organizes the main execution loop and provides a uniform way for deployment, configuration, control and logging of the agent activity.
- Agents run in different environments. Those that are part of the DIRAC subsystems, for example Workload Management or Data Distribution, are usually deployed close to the corresponding services. They watch for changes in the service states and react accordingly by initiating actions like job submission or result retrieval.
- Agents can run on a gatekeeper node of a site controlled by the DIRAC Workload Management System. In this case, they are part of the DIRAC WMS ensuring the pull job scheduling paradigm. Agents can also run as part of a job executed on a Worker Node as so called “Pilot Agents”.

**Interfaces**

- The DIRAC main programming language is Python and programming interfaces (APIs) are provided in this language.
- For the users of the DIRAC system the functionality is available through a command line interface.
- DIRAC also provides Web interfaces for users and system managers to monitor the system behaviour and to control the ongoing tasks. The Web interfaces are based on the DIRAC Web Portal framework which ensures secure access to the system service using X509 certificates loaded into the user browsers.

**3.3.3 DIRAC Framework**

The Dirac framework for building secure SOA based systems provides generic components not specific to LHCb which can be applied in the contexts of other VOs as well. The framework is written in the Python language and
includes the following components (Figure 2):

- DISET (DIRAC Secure Transport) secure communication protocol
- Web Portal framework
- Configuration System
- Logging System
- Monitoring System

Figure 2. DIRAC Framework Components

Web portal framework

The Web portal framework allows the building of Web interfaces to DIRAC services. It provides Authentication based on user grid credentials and user groups which can be selected during the interactive session. The framework uses the DISET portal functionality to redirect client requests to corresponding services and to collect responses. It provides the means to organize the contents of the DIRAC Web sites using the Pylons contents management system.

All the monitoring and control tools of a DIRAC system are exported through the Web portal which makes them uniform for users working in different environment and on different platforms.

Configuration Service

The Configuration Service is built in the DISET framework to provide static configuration parameters to all the distributed DIRAC components. This is the backbone of the whole system and necessitates excellent reliability. Therefore, it is organized as a single master service where all the parameter updates are done and multiple read-only slave services which are distributed geographically, on VO-boxes at Tier-1 LCG sites in the case of LHCb. All the servers are queried by clients in a load balancing way. This arrangement ensures configuration data consistency together with very good scalability properties.

Logging and Monitoring Services

- All the DIRAC components use the same logging facility which can be configured with one or more back-ends including standard output, log files or external service.
- The amount of the logging information is determined by a configurable level specification.
- Use of the logger permit report to the Logging Service where all the distributed components are encountering system failures.
- This service accumulates information for the analysis of the behaviour of the whole distributed system including third party services provided by the sites and central grid services.
- The quick error report analysis allows spotting and even fixing the problems before they hit the user.
- The Monitoring Service collects activity reports from all the DIRAC services and some agents. It presents the monitoring data in a variety of ways, e.g. historical plots, summary reports, etc. Together with the Logging Service, it provides a complete view of the health of the system for the managers.
3.4 Coding Conventions

Rules and conventions are necessary to insure a minimal coherence and consistency of the DIRAC software. Compliance with the rules and conventions is mainly based on the good will of all the contributors, who are working for the success of the overall project.

3.4.1 Code Organization

DIRAC code is organized in packages corresponding to Systems. Systems packages are split into the following standard subpackages:

- **Service** contains Service Handler modules together with possible auxiliary modules
- **Agent** contains Agent modules together with possible auxiliary modules
- **DB** contains Database definitions and front-end classes
- **scripts** contains System commands codes

Some System packages might also have additional

- **test** Any unit tests and other testing codes
- **Web** Web portal codes following the same structure as described in Developing Web Portal Pages.

Packages are sets of Python modules and eventually compilable source code together with the instructions to use, build and test it. Source code files are maintained in the SVN code repository.

**R1** Each package has a unique name, that should be written such that each word starts with an initial capital letter ("CamelCase" convention). Example: DataManagementSystem.

3.4.2 Module Coding Conventions

**R2** The first line of every file should contain `$HeadURL: $` macro, which SVN translates into the Author name and the Date of the most recent commit operation. This macro is typically placed in the first comment line of the module. Ignored with git.

**R3** Each module should define the following variables in its global scope:

```python
__RCSID__ = "$Id$"
```

this is the SVN macro substituted by the module revision number.

```python
__docformat__ = "restructuredtext en"
```

this is a variable specifying the mark-up language used for the module inline documentation (doc strings). See Documenting your developments for more details on the inline code documentation.

**R4** The first executable string in each module is a doc string describing the module functionality and giving instructions for its usage. The string is using ReStructuredText mark-up language.

**Importing modules**

**R5** Standard python modules are imported using:

```python
import <ModuleName>
```

Public modules from other packages are imported using:
import DIRAC.<Package[.SubPackage]>.<ModuleName>

**Naming conventions**

Proper naming the code elements is very important for the code clarity especially in a project with multiple developers. As a general rule, names should be meaningful but not too long.

**R6** Names are usually made of several words, written together without underscore, each first letter of a word being uppercased (CamelCase convention). The case of the first letter is specified by other rules. Only alphanumeric characters are allowed.

**R7** Names are case sensitive, but names that differ only by the case should not be used.

**R8** Avoid single characters and meaningless names like “jjj”, except for local loops or array indices.

**R9** Class names must be nouns, or noun phrases. The first letter is capital.

**R10** Class data attribute names must be nouns, or noun phrases. The first letter is lower case. The last word should represent the type of the variable value if it is not clear from the context otherwise. *Examples*: `fileList`, `nameString`, `pilotAgentDict`.

**R11** Function names and Class method names must be verbs or verb phrases, the first letter in lower case. *Examples*: `getDataMember`, `executeThisPieceOfCode`.

**R12** Class data member accessor methods are named after the attribute name with a “set” or “get” prefix.

**R13** Class data attributes must be considered as private and must never be accessed from outside the class. Accessor methods should be provided if necessary.

**R14** Private methods of a module or class must start by double underscore to explicitly prevent its use from other modules.

**Python files**

**R15** Python files should contain a definition of a single class, they may contain auxiliary (private) classes if needed. The name of the file should be the same as the name of the main class defined in the file.

**R16** A constructor must always initialize all attributes which may be used in the class.

**Methods and arguments**

**R17** Methods must not change their arguments. Use assignment to an internal variable if the argument value should be modified.

**R18** Methods should consistently return a `Result` (S_OK or S_ERROR) structure. A single return value is only allowed for simple methods that can not fail after the code is debugged.

**R19** Returned `Result` structures must always be tested for possible failures.

**R20** Exception mechanism should be used only to trap “unusual” problems. Use `Result` structures instead to report failure details.

### 3.4.3 Coding style

It is important to try to get a similar look, for an easier maintenance, as most of the code writers will eventually be replaced during the lifetime of the project.
General lay-out

R21 The length of any line should be preferably limited to 80 characters to allow debugging on any terminal.
R22 Each block is indented by two spaces.
R23 When declaring methods with multiple arguments, consider putting one argument per line. This allows inline comments and helps to stay within the 80 column limit.

Comments and doc strings

Comments should be abundant, and must follow the rules of automatic documentation by Epydoc tool using ReStructuredText mark-up.
R24 Each class and method definition should start with the doc strings. See Documenting your developments for more details.
R25 Use blank lines to separate blocks of statements but not blank commented lines.

Readability and maintainability

R26 Use spaces to separate operator from its operands.
R27 Method invocations should have arguments separated, at least by one space. In case there are long or many arguments, put them each on a different line.
R28 When doing lookup in dictionaries, don’t use dict.has_key(x) - it is deprecated and much slower than x in dict. Also, in python 3.0 this isn't valid.

3.5 Developing DIRAC components

What starts here is a guide to develop DIRAC components. This guide is done in the form of a tutorial, that should be followed if you are a new DIRAC developer. This guide will not teach you how to develop for a specific DIRAC system, rather will show you examples, and propose some exercises.

3.5.1 Check your installation

If you are here, we suppose you have read the documentation that came before. Specifically:

- you should know about our Development Model
- you should have your Development Environment already set up, specifically you should already have a developer installation.

Within this part we’ll check the basics, and we’ll do few exercises.

Is my installation correctly done?

We will now do few, very simple checks. The first can be done by using the python interactive shell. For these examples I will actually use iPython, which is a highly recommended shell.
In [1]: from DIRAC.Core.Base.Script import parseCommandLine

In [2]: parseCommandLine()
Out[2]: True

Was this good? If it wasn’t, then you should probably hit the “previous” button of this guide.

So, what’s that about? These 2 lines will initialize DIRAC. They are used in several places, especially for the scripts: each and every script in DIRAC start with those 2 lines above.

Let’s do one more check:

In [14]: from DIRAC import gConfig

In [15]: gConfig.getValue('/DIRAC/Setup')
Out[15]: 'DeveloperSetup'

Was this good? If it wasn’t, again, then you should probably hit the “previous” button of this guide.

Do not think about you just typed right now. It will become more clear later.

The real basic stuff

Let’s start with the logger

In [3]: from DIRAC import gLogger

In [4]: gLogger.notice('Hello world')
Hello world
Out[4]: True

What’s that? It is a singleton object for logging in DIRAC. Needless to say, you’ll use it a lot.

In [5]: gLogger.info('Hello world')
Out[5]: True

Why “Hello world” was not printed? Because the logging level is too high:

In [6]: gLogger.getLevel()
Out[6]: 'NOTICE'

But we can increase it simply doing, for example:

In [7]: gLogger.setLevel('VERBOSE')
Out[7]: True

In [8]: gLogger.info('Hello world')
Hello world
Out[8]: True

In DIRAC, you should not use print. Use the gLogger instead.

Let’s continue, and we have a look at the return codes:

In [11]: from DIRAC import S_OK, S_ERROR

These 2 are the basic return codes that you should use. How do they work?

In [12]: S_OK('All is good')
Out[12]: {'OK': True, 'Value': 'All is good'}
Quite clear, isn’t it? Often, you’ll end up doing a lot of code like that:

```
result = aDIRACMethod()
if not result['OK']:
    gLogger.error('aDIRACMethod-Fail', "Call to aDIRACMethod() failed with message %s" %result['Message'])
else:
    returnedValue = result['Value']
```

### Playing with the Configuration Service

If you are here, it means that your developer installation contains a `dirac.cfg` file, that should stay in the `$DIRACDEVS/etc` directory. We’ll play a bit with it now.

You have already done this:

```
In [14]: from DIRAC import gConfig
In [15]: gConfig.getValue('/DIRAC/Setup')
Out[15]: 'DeveloperSetup'
```

Where does ‘DeveloperSetup’ come from? Open that dirac.cfg and search for it. Got it? it’s in:

```
DIRAC{
...
  Setup = DeveloperSetup
...
}
```

Easy, huh? Try to get something else now, still using `gConfig.getValue()`.

So, `gConfig` is another singleton: it is the guy you need to call for basic interactions with the Configuration Service. If you are here, we assume you already know about the CS servers and layers. More information can be found in the Administration guide. We remind that, for a developer installation, we will work in ISOLATION, so with only the local `dirac.cfg`

Mostly, `gConfig` exposes get type of methods:

```
In [2]: gConfig.get
gConfig.getOption  gConfig.getOptionsDict  gConfig.getServersList
gConfig.getOptions  gConfig.getSections  gConfig.getValue
```

for example, try:

```
In [2]: gConfig.getOptionsDict('/DIRAC')
```

In the next section we will modify a bit the `dirac.cfg` file. Before doing that, have a look at it. It’s important what’s in there, but for the developer installation it is also important what it is NOT there. We said we will work in isolation. So, it’s important that this file does not contain any URL to server infrastructure (at least, not at this level: later, when you will feel more comfortable, you can add some).

A very important option of the `cfg` file is “DIRAC/Configuration/Server”: this option can contain the URL(s) of the running Configuration Server. But, as said, for doing development, this option should stay empty.
Getting a Proxy

We assume that you have already your public and private certificates key in $HOME/.globus. Then, do the following:

```
dirac-proxy-init
```

if you got something like:

```
> dirac-proxy-init
Traceback (most recent call last):
  File "/home/dirac/diracInstallation/scripts/dirac-proxy-init", line 22, in <module>
    for entry in os.listdir( baseLibPath ):
OSError: [Errno 2] No such file or directory: '/home/dirac/diracInstallation/Linux_x86_64_glibc-2.12/
```

just create the directory by hand.

Now, if try again you will probably get something like:

```
> dirac-proxy-init
Generating proxy...
Enter Certificate password:
DN /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=fstagni/CN=693025/CN=Federico Stagni is not registered
```

This is because DIRAC still doesn’t know you exist. You should add yourself to the CS. For example, I had add the following section:

```
Registry
{
    Users
    {
        fstagni
        {
            DN = /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=fstagni/CN=693025/CN=Federico Stagni
            CA = /DC=ch/DC=cern/CN=CERN Trusted Certification Authority
            Email = federico.stagni@cern.ch
        }
    }
}
```

All the info you want and much more in:

```
openssl x509 -in usercert.pem -text
```

Now, it’s time to issue again:

```
toffo@pclhcb181:~/.globus$ dirac-proxy-init
Generating proxy...
Enter Certificate password:
User fstagni has no groups defined
```

So, let’s add the groups within the /Registry section:

```
Groups
{
    devGroup
    {
        Users = fstagni
    }
}
```

You can keep playing with it (e.g. adding some properties), but for the moment this is enough.
3.5.2 Your first DIRAC code

We will now code some very simple exercises, based on what we have seen in the previous section. Before going through the exercise, you should verify in which GIT branch you are, so go to the directory where you cloned DIRAC and issue:

```shell
> git branch
```

this will show all your local branches. Now, remember that you have to base your development on a remote branch. This is clearly explained in Contributing code, so be careful on what you choose: checkout a new branch from a remote one before proceeding.

**Exercise 1:**

Code a python module in DIRAC.Core.Utilities.checkCAOfUser where there is only the following function:

```python
def checkCAOfUser( user, CA ):
    """ user, and CA are string
    """
```

This function should:

- Get from the CS the registered Certification Authority for the user
- if the CA is the expected one return S_OK, else return S_ERROR

To code this exercise, albeit very simple, we will use TDD (Test Driven Development), and we will use the unittest and mock python packages, as explained in Testing (VO)DIRAC. What we will code here will be a real unit test, in the sense that we will test only this function, in isolation. In general, it is always an excellent idea to code a unit test for every development you do. We will put the unit test in DIRAC.Core.Utilities.test. The unit test has been fully coded already:

```python
# imports
import unittest, mock, importlib
# sut
from DIRAC.Core.Utilities.checkCAOfUser import checkCAOfUser

class TestcheckCAOfUser( unittest.TestCase ):
    def setUp( self ):
        self.gConfigMock = mock.Mock()
        self.checkCAOfUser = importlib.import_module( 'DIRAC.Core.Utilities.checkCAOfUser' )
        self.checkCAOfUser.gConfig = self.gConfigMock

    def tearDown( self ):
        pass

class TestcheckCAOfUserSuccess( TestcheckCAOfUser ):
    def test_success( self ):
        self.gConfigMock.getValue.return_value = 'attendedValue'
        res = checkCAOfUser( 'aUser', 'attendedValue' )
        self.assert_( res['OK'] )

    def test_failure( self ):
        self.gConfigMock.getValue.return_value = 'unAttendedValue'
        res = checkCAOfUser( 'aUser', 'attendedValue' )
        self.assertFalse( res['OK'] )
```

3.5. Developing DIRAC components
Now, try to run it. In case you are using Eclipse, it’s time to try to run this test within Eclipse itself (run as: Python unit-test): it shows a graphical interface that you can find convenient. If you won’t manage to run, it’s probably because there is a missing configuration of the PYTHONPATH within Eclipse.

Then, code checkCAOfUser and run the test again.

Exercise 2:

As a continuation of the previous exercise, code a python script that will:

- call DIRAC.Core.Utilities.checkCAOfUser.checkCAOfUser
- log with info or error mode depending on the result

Remember to start the script with:

```python
#!/usr/bin/env python
""" Some doc: what does this script should do?
"""
from DIRAC.Core.Base import Script
Script.parseCommandLine()
```

Then run it.

### 3.5.3 Developing Services

**Service Handler**

All the DIRAC Services are built in the same framework where developers should provide a Service Handler by inheriting the base RequestHandler class. An instance of the Service Handler is created each time the service receives a client query. Therefore, the handler data members are only valid for one query. If the service state should be preserved, this should be done using global variables or a database back-end.

Creating a Service Handler is best illustrated by the example below which is presenting a fully functional although a simple service:

```python
""" Hello Service is an example of how to build services in the DIRAC framework
""
__RCSID__ = "$Id: "$
import types
from DIRAC.CORE.DIRECT.RequestHandler import RequestHandler
from DIRAC import gLogger, S_OK, S_ERROR
from DIRAC.Core.Utilities import Time

class HelloHandler( RequestHandler ):
    @classmethod
    def initializeHandler( cls, serviceInfo ):
        """ Handler initialization
        """
Let us walk through this code to see which elements should be provided.

The first lines show the documentation string describing the service purpose and behavior. It is followed by the "__RCSID__" global module variable which is assigned the value of the "$Id: $" Git keyword.

After that come the import statements. Several import statements will be clear from the subsequent code.

Then comes the definition of the HelloHandler class. The Service name is Hello. The "initializeHandler" method is called once when the Service is created. Here one can put creation and initialization of the variables for the service class if necessary. Note that the "initializeHandler" has a '@classmethod' decorator. That's because the code initializes the class instead of the instance of it.

Then comes the "initialize" method. This is used to initialize each instance of the requests. Every request will trigger a creation of one instance of HelloHandler. This method will be called after all the internal initialization is done.

Regarding service methods accessible to clients: The name of each method which will be accessible to the clients has export_prefix. Note that the clients will call the method without this prefix. Otherwise, it is an ordinary class method which takes the arguments provided by the client and returns the result to the client. The result must always be returned as an S_OK or S_ERROR structure.

A useful method is "srv_getCSOption(csPath, defaultValue)" which allows to extract options from the Service section in the Configuration Service directly without having to use the "gConfig" object.

For each "exported" method the service can define an auth_<method_name> class variable being a list. This will restrict which clients can call this method. Only clients belonging to groups that have the properties defined in the list will be able to call this method. all is a special keyword that allows anyone to call this method. authenticated is also a special keyword that allows anyone with a valid certificate to call this method.

For each service interface method it is necessary to define types_<method_name> class variable of the List type. Each element of the List is one or a list of possible types of the method arguments in the same order as defined in the method definition. The types are imported from the "types" standard python module.

**Default Service Configuration parameters**

The Hello Handler is written. Now, we’ll need to put the new service in the dirac CS in order to see it running. Since we are running in an isolated installation, the net effect is that the service will have to be added to the local “dirac.cfg” file.

To do this, we should first have a “/Systems” section in it. The “/Systems” section keeps references to the real code, e.g. if you are developing for the “WorkloadManagementSystem” you should have a “/Systems/WorkloadManagement” section. If there are services that have to run in the WMS, you should place them under “/Systems/WorkloadManagement/Services”.

```python
cls.defaultWhom = "World"
return S_OK()

def initialize(self):
    """ Response initialization
    """
    self.requestDefaultWhom = self.srv_getCSOption( "DefaultWhom", HelloHandler.defaultWhom )

    auth_sayHello = [ 'all' ]
    types_sayHello = [ types.StringTypes ]

def export_sayHello( self, whom ):
    """ Say hello to somebody
    """
    if not whom:
        whom = self.requestDefaultWhom
    return S_OK( "Hello " + whom )
```
For what concerns our example, we should place it to the Service directory of one of the DIRAC System directories, for example we can use FrameworkSystem. The default Service Configuration parameters should be added to the corresponding System ConfigTemplate.cfg file. In our case the Service section in the ConfigTemplate.cfg will look like the following:

```
Services
{
    Hello
    {
        Port = 3424
        DefaultWhom = Universe
    }
}
```

Note that you should choose the port number on which the service will be listening which is not conflicting with other services. This is the default value which can be changed later in the Configuration Service. The Port parameter should be specified for all the services. The ‘DefaultWhom’ is this service specific option.

Now, you can try to run the service. To do that, simply:

```
dirac-service Framework/Hello -ddd
```

The `-ddd` is for running in DEBUG mode. At first, this will not work. Useful info will be printed out, and you’ll have to work on your dirac.cfg to make it run. Once you are done, you are ready to go.

If everything goes well, you should see something like:

```
```

The URL displayed should be added to the local dirac.cfg in the URLs section.

### Installing the Service

We are running in isolation. So, unless you run also a ConfigurationServer on your machine, you won’t be able to do the following, and you can safely skip this part.

The Service is ready it should be installed. The DIRAC Server installation is described in [[here]]. If you are adding the Service to an already existing installation it is sufficient to execute the following in this DIRAC instance:

```
> dirac-install-service Framework Hello
```

This command will do several things:

- It will create the Hello Service directory in the standard place and will set it up under the “runit” control - the standard DIRAC way of running permanent processes.
- The Hello Service section will be added to the Configuration System. So, its address and parameters will be available to clients.

The Service can be also installed using the SystemAdministrator CLI interface (provided that you are running Framework/SystemAdministrator service on your machine):

```
> install service Framework Hello
```

The SystemAdministrator interface can also be used to remotely control the Service, start or stop it, uninstall, get the Service status, etc. and can be invoked in the standard way via a DIRAC client installation:

```
> dirac-admin-sysadmin-cli --host=myDIRACServer
```

As said in the previous section, in any case, if you are developing a service, you might test it without installing it, by simply running:
Calling the Service from a Client

Once the Service is running it can be accessed from the clients in the way illustrated by the following code snippet:

```python
from DIRAC.Core.DISET.RPCClient import RPCClient

simpleMessageService = RPCClient('Framework/Hello')
result = simpleMessageService.sayHello( 'you' )
if not result['OK']:
    print "Error while calling the service:", result['Message']
else:
    print result[ 'Value' ]
```

Note that the service is always returning the result in the form of S_OK/S_ERROR structure.

When should a service be developed?

Write a service every time you need to expose some information, that is usually stored in a database.

There are anyway cases for which it is not strictly needed to write a service, specifically when all the following are true:
- when you never need to expose the data written in the DB (i.e. the DB is, for the DIRAC point of view, Read-Only)
- when the components writing in it have local access.

The advise is anyway to always write the service, because: - if later on you’ll need it, you won’t need to change anything but the service itself - db-independent logic should stay out of the database class itself.

3.5.4 Testing a service while developing it

As described in Testing (VO)DIRAC a way to test a service is to run an integration test, that can run when the service is actually running. It is also possible to write a proper unit test, but this is not the usually recommended way. Reasons are:

- It’s not trivial to write a unit test for a service: reason being, the DIRAC framework can’t be easily mocked.
- The code inside a service is (should be) simple, no logic should be embedded in there: so, what you want to test, is its integration.

Exercise 1:

Write an integration test for HelloHandler. This test should use python unittest, and should assume that the Hello service is running. The test stub follows:

```python
# imports
import unittest
# sut
from DIRAC.Core.DISET.RPCClient import RPCClient

class TestHelloHandler( unittest.TestCase ):
    def setUp( self ):
        self.helloService = RPCClient('Framework/Hello')
    def tearDown( self ):
```

3.5. Developing DIRAC components 335
As said, examples can be found in the TestDIRAC repository in GitHub.

### 3.5.5 Developing Databases

Before starting developing databases, you have to make sure that MySQL is installed, as well as python-mysql, as explained in *Setting up a development installation*.

#### Develop the database

To develop a new database structure it requires to design a database schema and develop the python database class that will interact with the database itself. A simple example of the python class of a database follows:

```python
from DIRAC import gConfig, gLogger, S_OK, S_ERROR
from DIRAC.Core.Base.DB import DB

class AtomDB( DB ):
    
    def __init__( self ):
        DB.__init__( self, 'AtomDB', 'Test/AtomDB', 10 )
        retVal = self.__initializeDB()
        if not retVal[ 'OK' ]:
            raise Exception( "Can't create tables: $s" % retVal[ 'Message' ] )

    def __initializeDB( self ):
        
        Create the tables
        
        retVal = self.__query( "show tables" )
        if not retVal[ 'OK' ]:
            return retVal
        
        tablesInDB = [ t[0] for t in retVal[ 'Value' ] ]
tablesD = {}

        if 'atom_mytable' not in tablesInDB:
            tablesD[ 'atom_mytable' ] = { 'Fields' : { 'Id': 'INTEGER NOT NULL AUTO_INCREMENT', 'Stuff' : 'VARCHAR(64) NOT NULL' }, 'PrimaryKey' : [ 'Id' ]
```

---

As said, examples can be found in the TestDIRAC repository in GitHub.

### 3.5.5 Developing Databases

Before starting developing databases, you have to make sure that MySQL is installed, as well as python-mysql, as explained in *Setting up a development installation*.

#### Develop the database

To develop a new database structure it requires to design a database schema and develop the python database class that will interact with the database itself. A simple example of the python class of a database follows:

```python
from DIRAC import gConfig, gLogger, S_OK, S_ERROR
from DIRAC.Core.Base.DB import DB

class AtomDB( DB ):
    
    def __init__( self ):
        DB.__init__( self, 'AtomDB', 'Test/AtomDB', 10 )
        retVal = self.__initializeDB()
        if not retVal[ 'OK' ]:
            raise Exception( "Can't create tables: $s" % retVal[ 'Message' ] )

    def __initializeDB( self ):
        
        Create the tables
        
        retVal = self.__query( "show tables" )
        if not retVal[ 'OK' ]:
            return retVal
        
        tablesInDB = [ t[0] for t in retVal[ 'Value' ] ]
tablesD = {}

        if 'atom_mytable' not in tablesInDB:
            tablesD[ 'atom_mytable' ] = { 'Fields' : { 'Id': 'INTEGER NOT NULL AUTO_INCREMENT', 'Stuff' : 'VARCHAR(64) NOT NULL' }, 'PrimaryKey' : [ 'Id' ]
```

---

As said, examples can be found in the TestDIRAC repository in GitHub.

### 3.5.5 Developing Databases

Before starting developing databases, you have to make sure that MySQL is installed, as well as python-mysql, as explained in *Setting up a development installation*.

#### Develop the database

To develop a new database structure it requires to design a database schema and develop the python database class that will interact with the database itself. A simple example of the python class of a database follows:

```python
from DIRAC import gConfig, gLogger, S_OK, S_ERROR
from DIRAC.Core.Base.DB import DB

class AtomDB( DB ):
    
    def __init__( self ):
        DB.__init__( self, 'AtomDB', 'Test/AtomDB', 10 )
        retVal = self.__initializeDB()
        if not retVal[ 'OK' ]:
            raise Exception( "Can't create tables: $s" % retVal[ 'Message' ] )

    def __initializeDB( self ):
        
        Create the tables
        
        retVal = self.__query( "show tables" )
        if not retVal[ 'OK' ]:
            return retVal
        
        tablesInDB = [ t[0] for t in retVal[ 'Value' ] ]
tablesD = {}

        if 'atom_mytable' not in tablesInDB:
            tablesD[ 'atom_mytable' ] = { 'Fields' : { 'Id': 'INTEGER NOT NULL AUTO_INCREMENT', 'Stuff' : 'VARCHAR(64) NOT NULL' }, 'PrimaryKey' : [ 'Id' ]
```
Let's break down the example. The first two lines are simple includes required. Then the class definition. The name of the class should be the same name as the file where it is. So AtomDB should be in AtomDB.py. The class should inherit from the DB class. The DB class includes all the methods necessary to access, query, modify... the database.

The first line in the __init__ method should be the initialization of the parent (DB) class. That initialization requires 3 parameters:

1. Logging name of the database. This name will be used in all the logging messages generated by this class.
2. Full name of the database. With System/Name. So it can know where in the CS look for the initialization parameters. In this case it would be /Systems/Test/<instance name>/Databases/AtomDB.
3. Maximum number of requests to queue. 10 is a good value for this setting.

After the initialization of the DB parent class we call our own __initializeDB method. This method (following __init__ in the example) first retrieves all the tables already in the database. Then for each table that has not yet been created then it creates a definition of the table and creates all the missing tables. Each table definition includes all the fields with their value type, primary keys, extra indexes... By default all tables will be created using the InnoDB engine.

The addStuff method simply inserts into the created table the argument value.

Configure the database access

The last step is to configure the database credentials for DIRAC to be able to connect. In our previous example the CS path was /Systems/Test/<instance name>/Databases/AtomDB. That section should contain:

```python
Systems
{
    Test
    {
        <instance name>
        {
            Databases
            {
                AtomDB
                {
                    Host = localhost
                    User = yourusername
                    Password = yourpasswd
                    DBName = yourdbname
                }
            }
        }
    }
}
```

Keep in mind that <instance name> is the name of the instance defined under /DIRAC/Setup/<your setup>/Test and <your setup> is defined under /DIRAC/Setup.

Once that is defined you're ready to go
Trying the database from the command line

You can try to access the database by doing:

```python
from DIRAC.TestSystem.DB.AtomDB import AtomDB

try:
    atomdb = AtomDB()
except Exception:
    print "Oops. Something went wrong..."
    raise

result = atomdb.addStuff( 'something' )
if not result['OK']:
    print "Error while inserting into db:", result['Message']  # Here, in DIRAC, you better use the gLogger
else:
    print result[ 'Value' ]  # Here, in DIRAC, you better use the gLogger
```

3.5.6 Testing a DB while developing it

For testing a DB code, we suggest to follow similar paths of what is explained in *Testing a service while developing it*, so to run an integration test. In any case, to test the DB class you’ll need... the DB! And, on top of that, in DIRAC, it makes very little sense to have DB class functionalities not exposed by a service, so you might even want to test the service and DB together.

**Exercise 1:**

Write an integration test for AtomDB using python unittest. Then, write a service for AtomDB and its integration test.

3.5.7 Developing Agents

**What is an agent?**

Agents are active software components which run as independent processes to fulfil one or several system functions. They are the engine that make DIRAC beat. Agents are processes that perform actions periodically. Each cycle agents typically contact a service or look into a DB to check for pending actions, execute the required ones and report back the results. All agents are built in the same framework which organizes the main execution loop and provides a uniform way for deployment, configuration, control and logging of the agent activity.

Agents run in different environments. Those belonging to a DIRAC system, for example Workload Management or Data Distribution, are usually deployed close to the corresponding services. They watch for changes in the system state and react accordingly by initiating actions like job submission or result retrieval.

**Simplest Agent**

An agent essentially loops over and over executing the same function every \( X \) seconds. It has essentially two methods, *initialize* and *execute*. When the agent is started it will execute the *initialize* method. Typically this *initialize* method will define (amongst other stuff) how frequently the *execute* method will be run. Then the *execute* method is run. Once it finishes the agent will wait until the required seconds have passed and run the *execute* method again. This will loop over and over until the agent is killed or the specified amount of loops have passed.

Creating an Agent is best illustrated by the example below which is presenting a fully functional although simplest possible agent:
Let us walk through this code to see which elements should be provided.

First comes the documentation string describing the service purpose and behavior. It is followed by the ""__RCSID__"" global module variable which we have already seen in the services part.

Several import statements will be clear from the subsequent code.

The Agent name is SimplestAgent. The initialize method is called once when the Agent is created. Here one can put creation and initialization of the global variables if necessary. Please note that the __init__ method cannot be used when developing an Agent. It is used to initialize the module before it can be used.

Now comes the definition of the execute method. This method is executed every time Agent runs. Place your code inside this method. Other methods can be defined in the same file and used via execute method. The result must always be returned as an S_OK or S_ERROR structure for the execute method. The previous example will execute the same example code in the Services section from within the agent.

3.5. Developing DIRAC components
Default Agent Configuration parameters

The Agent is written. It should be placed to the Agent directory of one of the DIRAC System directories in the code repository, for example FrameworkSystem. The default Service Configuration parameters should be added to the corresponding System ConfigTemplate.cfg file. In our case the Service section in the ConfigTemplate.cfg will look like the following:

```plaintext
Agents
{
    SimplestAgent
    {
        LogLevel = INFO
        LogBackends = stdout
        PollingTime = 60
        Message = still working...
    }
}
```

Polling time define execution time scheduling. The Message is this agent specific option.

Installing the Agent

Once the Agent is ready it should be installed. As for the service part, we won’t do this part unless we want to mimic a full installation. Also, this part won’t work if we won’t have a ConfigurationServer running, which is often the case of a developer installation. For our development installation we can modify our local `dirac.cfg` in a very similar fashion to what we have done for the service part in the previous section, and run the agent using the dirac-agent command.

The DIRAC Server installation is described in documentation. If you are adding the Agent to an already existing installation it is sufficient to execute the following in this DIRAC instance:

```plaintext
> dirac-install-agent Framework SimplestAgent
```

This command will do several things:

- It will create the SimpleAgent Agent directory in the standard place and will set it up under the “runit” control - the standard DIRAC way of running permanent processes.
- The SimplestAgent Agent section will be added to the Configuration System.

The Agent can be also installed using the SystemAdministrator CLI interface:

```plaintext
> install agent Framework SimplestAgent
```

The SystemAdministrator interface can also be used to remotely control the Agent, start or stop it, uninstall, get the Agent status, etc.

Checking the Agent output from log messages

If case you are running a SystemAdministrator service, you’ll be able to login to the machine using (as adminturator) `dirac-admin-sysadmin-cli` and show the log of SimplestAgent using:

```plaintext
> show log Framework SimplestAgent
```

An info message will appear in log:

```plaintext
Framework/SimplestAgent INFO: message: still working...
```

Note that the service is always returning the result in the form of S_OK/S_ERROR structure.
3.5.8 Testing an agent while developing it

An agent can be tested in 2 ways: either with a unit test, or with an integration test. One does not exclude the other.

Agents can be very complex. So, deciding how you approach test is very much dependent on what’s the code inside the agent itself.

First, tackling the integration test: in TestDIRAC there’s no integration test involving agents. That’s because an integration test for an agent simply means “start it, and look in how it goes”. There’s not much else that can be done, maybe the only thing would be to test that “execute()” returns S_OK()

So, what can be wrote down are integration tests

```python
import unittest, importlib
from mock import MagicMock, patch

class MyAgentTestCase( unittest.TestCase ):
    def setUp( self ):
        self.mockAM = MagicMock()
        self.agent = importlib.import_module( 'LHCbDIRAC.TransformationSystem.Agent.MCSimulationTestingAgent' )
        self.agent.AgentModule = self.mockAM
        self.agent = MCSimulationTestingAgent()
        self.agent.log = gLogger
        self.agent.log.setLevel( 'DEBUG' )

    def tearDown( self ):
        pass

    def test_myTest( self ):
        bla

if __name__ == '__main__':
    suite = unittest.defaultTestLoader.loadTestsFromTestCase( MyAgentTestCase )
    testResult = unittest.TextTestResult(verbosity = 2).run(suite)
```

3.5.9 Developing Executors

The Executor framework is designed around two components. The Executor Mind knows how to retrieve, store and dispatch tasks. And Executors are the working processes that know what to do depending on the task type. Each Executor is an independent process that connects to the Mind and waits for tasks to be sent to them by the . The mechanism used to connect the Executors to the is described in section . A diagram of both components can been seen in the diagram.

The Mind is a DIRAC service. It is the only component of the Executor framework that needs write-access to the database. It loads tasks from the database and writes the results back. The Mind can periodically query a database to find new tasks, but it can also receive new tasks from other components. Executors don’t keep or store the result of any task. If an Executor dies without having finished a task, the Mind will simply send the task to another Executor.

When the Mind receives a task that has been properly processed by an Executor, the result will have to be stored in the database. But before storing it in the database the Mind needs to check that the task has not been modified by anyone else while the executor was processing it. To do so, the Mind has to store a task state in memory and check that this task state has not been modified before committing the result back to the database. The task state will be different for each type of task and has to be defined in each case.

When an Executor process starts it will connect to the Mind and send a list of task types it can process. The acts as task scheduler and dispatcher. When the Mind has a task to be processed it will look for an idle Executor that can
process that task type. If there is no idle Executor or no can process that task type, the Mind will internally queue the task in memory. As soon a an Executor connects or becomes idle, the Mind will pop a task from one of the queues that the can process and send the task to it. If the Executor manages to process the task, the Mind will store back the result of the task and then it will try to fill the again with a new task. If the Executor disconnects while processing a task, the Mind will assume that the has crashed and will reschedule the task to prevent any data loss.

Tasks may need to go through several different steps before being completely processed. This can easily be accomplished by having one task type for each step the task has to go through. Each Executor can then publish what task types it knows how to process. For each step the task has to go through, the Mind will send the task to an Executor that can process that type of task, receive and store the result, change the task to the next type and then send the task to the next Executor. The Mind will repeat this mechanism until the task has gone through all the types.

This architecture allows to add and remove Executors at any time. If the removed Executor was being processing a task, the Mind will send the task to another Executor. If the task throughput is not enough Executors can be started and the Mind will send them tasks to process. Although Executors can be added and removed at any time, the Mind is still a single point of failure. If the Mind stops working the whole system will stop working.

**Implementing an Executor module**

Implementing an executor module is quite straightforward. It just needs 4 methods to be implemented. Here’s an example:

```python
import threading
from DIRAC import S_OK, S_ERROR, gLogger
from DIRAC.Core.Utilities import DEncode
from DIRAC.Core.Base.ExecutorModule import ExecutorModule

class PingPongExecutor( ExecutorModule ):
    @classmethod
    def initialize( cls ):
        """
        Executors need to know to which mind they have to connect.
        """
        cls.ex_setMind( "Test/PingPongMind" )
        return S_OK()

    def processTask( self, taskid, taskData ):
        """
        This is the function that actually does the work. It receives the task, does the processing and sends the modified task data back.
        """
        taskData[ 'bouncesLeft' ] -= 1
        return S_OK( taskData )

    def deserializeTask( self, taskStub ):
        """
        Tasks are received as a stream of bytes. They have to be converted from that into a usable object.
        """
        return S_OK( DEncode.decode( taskStub )[0] )

    def serializeTask( self, taskData ):
        """
        Before sending the task back to the mind it has to be serialized again.
        """
        return S_OK( DEncode.encode( taskData ) )
```

3.5. Developing DIRAC components 343
All *Executor* modules need to know to which mind they have to connect. In the *initialize* method we define the mind to which the module will connect. This method can also have any other initialization required by the *Executor*.

Function *processTask* does the task processing. It receives the task to be processed already deserialized. Once the work it’s done it can to return the modified task or just and empty *S_OK*.

The last two methods provide the knowledge on how to serialize and deserialize tasks when receiving and sending them to the *Mind*.

**Running an Executor**

*Executor* modules are run by the *dirac-executor* script. This allows to run more than one module by the same process. Just invoke *dirac-executor* passing as parameter all the required modules. It will group all the modules by *Mind* and create just one connection to the each requested *Mind*. *Minds* will know how to handle *Executors* running more than one module.

**Implementing a Mind**

The *Mind* is a bit more complex. It has to:

- Dispatch tasks to executors that can handle them. A *Mind* can have more than one type of *Executor* module connected. So it has to decide which module type will handle the task. For instance there may be two *Executor* modules connected, the task has to be processed by module 1 and then by module 2. So the mind has to decide to send the task first to module 1, and once it comes back then send it to module 2.
- It has to either get notified or check some resource to start executing a task. Once the task has been processed it has to store back the result to the database or to wherever the result has to go.

A simple example follows:

```python
import types
import time
import random
from DIRAC import S_OK, S_ERROR, gLogger
from DIRAC.Core.Utilities import DEncode
from DIRAC.Core.Base.ExecutorMindHandler import ExecutorMindHandler

random.seed()

class PingPongMindHandler( ExecutorMindHandler ):

    MSG_DEFINITIONS = { 'StartReaction' : { 'numBounces' : ( types.IntType, types.LongType ) } }

    auth_msg_StartReaction = [ 'all' ]
    def msg_StartReaction( self, msgObj ):
        bouncesLeft = msgObj.numBounces
        taskid = time.time() + random.random()
        taskData = { 'bouncesLeft' : bouncesLeft }
        return self.executeTask( time.time() + random.random(), taskData )

    auth_startPingOfDeath = [ 'all' ]
    types_startPingOfDeath = [ types.IntType ]
    def export_startPingOfDeath( self, numBounces ):
        taskData = { 'bouncesLeft' : numBounces }
        gLogger.info( "START TASK = %s" % taskData )
        return self.executeTask( int( time.time() + random.random() ), taskData )
```

344 Chapter 3. Developer Guide
@classmethod
def exec_executorConnected( cls, trid, eTypes ):
    
    This function will be called any time an executor reactor connects
    eTypes is a list of executor modules the reactor runs
    
gLogger.info( "EXECUTOR CONNECTED OF TYPE %s" % eTypes )
    return S_OK()

@classmethod
def exec_executorDisconnected( cls, trid ):
    
    This function will be called any time an executor disconnects
    
    return S_OK()

@classmethod
def exec_dispatch( cls, taskid, taskData, pathExecuted ):
    
    Before a task can be executed, the mind has to know which executor module can process it
    
gLogger.info( "IN DISPATCH %s" % taskData )
    if taskData['bouncesLeft'] > 0:
        gLogger.info( "SEND TO PLACE" )
        return S_OK( "Test/PingPongExecutor" )
    return S_OK()

@classmethod
def exec_prepareToSend( cls, taskId, taskData, trid ):
    
    return S_OK()

@classmethod
def exec_serializeTask( cls, taskData ):
    gLogger.info( "SERIALIZE %s" % taskData )
    return S_OK( DEncode.encode( taskData ) )

@classmethod
def exec_deserializeTask( cls, taskStub ):
    gLogger.info( "DESERIALIZE %s" % taskStub )
    return S_OK( DEncode.decode( taskStub )[0] )

@classmethod
def exec_taskProcessed( cls, taskid, taskData, eType ):
    
    This function will be called when a task has been processed and by which executor module
    
gLogger.info( "PROCESSED %s" % taskData )
    taskData['bouncesLeft'] -= 1
    return cls.executeTask( taskid, taskData )

@classmethod
def exec_taskError( cls, taskid, taskData, errorMsg ):
    print "OOOOOO THERE WAS AN ERROR!!", errorMsg
    return S_OK()
@classmethod
def exec_taskFreeze( cls, taskid, taskData, eType ):
    
    A task can be frozen either because there are no executors connected that can handle it 
    or because an executor has requested it.
    
    print "OOOOOO THERE WAS A TASK FROZEN"
    return S_OK()

As shown in the example, Minds are DIRAC services so they can use any capability available. In the example we define a message called ‘StartReaction’. Each time the Mind receives that message it will add a task to be processed. For this example, a task is just a dictionary with one key having one number as value. This number will define how many times the task will go to an Executor to be processed. Each time an Executor processes a task we will just reduce the number of bounces left.

The Mind also has two methods to react when an Executor connects or disconnects. Keep in mind that each Executor can have more than one module as explained in section Running an Executor. The connect callback will give the Mind the list of modules the Executor has.

The exec_dispatch method is quite important. It decides which Executor module has to process the task. Returning an empty S_OK means that no module has to process this task and thus that this task can now be forgotten. In the example exec_dispatch will just look at the number of bounces our task has done. If there are still bounces to do it will just say that the Framework/PingPong Executor has to process the task and no module if there are no bounces left to do.

Methods exec_serialize and exec_deserialize have to provide a mechanism for packing and unpacking tasks from byte arrays. Executors have the same mechanism in methods serialize and deserialize. In fact, it is highly recommended that Executors and their Minds share this methods.

Method exec_prepareToSend allows the Mind to prepare before sending a task. It is not required to overwrite this method. It’s there in case some Mind needs it.

All that’s left are callbacks for when tasks come back from Executors:

* **exec_taskDone** will be called if the task has been processed without error. In this method the Mind can save the new state into a database, notify a resource...

* **exec_taskError** will be called if the Executor has found any error while processing the task. After this method the Mind will forget about the task.

* **exec_taskFreeze** will be called if the Executor requests to freeze the task for some time. For instance an Executor can process a task and decide that it has to be retried later. It can just freeze the task for a certain amount of time. The Mind will keep this task for at least that amount of time. It can keep it for more time if there aren’t free Executors to handle it.

### 3.5.10 Developing Commands

Commands are one of the main interface tools for the users. Commands are also called scripts in DIRAC lingo.

#### Where to place scripts

All scripts should live in the scripts directory of their parent system. For instance, the command:

```
dirac-wms-job-submit
```

will live in:
The command script name is the same as the command name itself with the .py suffix appended. When DIRAC client software is installed, all scripts will be placed in the installation scripts directory and stripped of the .py extension. This is done by the dirac-deploy-scripts command that you should have already done when you installed. This way users can see all the scripts in a single place and it makes easy to include all the scripts in the system PATH variable.

**Coding commands**

All the commands should be coded following a common recipe and having several mandatory parts. The instructions below must be applied as close as possible although some variation are allowed according to developer’s habits.

1. All scripts must start with a Shebang line like the following:

```bash
#!/usr/bin/env python
```

which will set the interpreter directive to the python on the environment.

2. The next is the documentation line which is describing the command. This same documentation line will be used also the command help information available with the -h command switch.

3. Users need to specify parameters to scripts to define what they want to do. To do so, they pass arguments when calling the script. The first thing any script has to do is define what options and arguments the script accepts. Once the valid arguments are defined, the script can parse the command line. An example follows which is a typical command description part

```python
#!/usr/bin/env python

""" Ping a list of services and show the result
""

__RCSID__ = "$Id$"

import sys
from DIRAC import S_OK, S_ERROR, gLogger, exit
from DIRAC.Core.Base import Script

# Define a simple class to hold the script parameters
class Params:

    def __init__( self ):
        self.raw = False
        self.pingsToDo = 1

    def setRawResult( self, value ):
        self.raw = True
        return S_OK()

    def setNumOfPingsToDo( self, value ):
        try:
            self.pingsToDo = max( 1, int( value ) )
        except ValueError:
            return S_ERROR( "Number of pings to do has to be a number" )
        return S_OK()

# Instantiate the params class
cliParams = Params()
```
Let’s follow the example step by step. First, we import the required modules from DIRAC. \texttt{S\_OK} and \texttt{S\_ERROR} are the default way DIRAC modules return values or errors. The \texttt{Script} module is the initialization and command line parser that scripts use to initialize themselves. No other DIRAC module should be imported here.

Once the required modules are imported, a \texttt{Params} class is defined. This class holds the values for all the command switches together with all their default values. When the class is instantiated, the parameters get the default values in the constructor function. It also has a set of functions that will be called for each switch that is specified in the command line. We’ll come back to that later.

Then the list of valid switches and what to do in case they are called is defined using \texttt{registerSwitch()} method of the \texttt{Scripts} module. Each switch definition has 4 parameters:

1. Short switch form. It has to be one letter. Optionally it can have ‘:’ after the letter. If the switch has ‘:’ it requires one parameter with the switch. A valid combination for the previous example would be ‘-r-p 2’. That means show raw results and make 2 pings.
2. Long switch form. ‘=’ is the equivalent of ‘:’ for the short form. The same combination of command switches in a long form will look like ‘--showRaw --numPings 2’.
3. Definition of the switch. This text will appear in the script help.
4. Function to call if the user uses the switch in order to process the switch value

There are several reserved switches that DIRAC uses by default and cannot be overwritten by the script. Those are:

- \texttt{-h} and \texttt{--help} show the script help
- \texttt{-d} and \texttt{--debug} enables debug level for the script. Note that the forms \texttt{-dd} and \texttt{-ddd} are accepted resulting in increasingly higher verbosity level
- \texttt{-s} and \texttt{--section} changes the default section in the configuration for the script
- \texttt{-o} and \texttt{--option} set the value of an option in the configuration
- \texttt{-c} and \texttt{--cert} use certificates to connect to services

All the command line arguments that are not corresponding to the explicitly defined switches are returned by the \texttt{getPositonalArguments()} function.

After defining the switches, the \texttt{parseCommandLine()} function has to be called. This method not only parses the command line options but also initializes DIRAC collecting all the configuration data. \textbf{It is absolutely important to}
call this function before importing any other DIRAC module. The callbacks defined for the switches will be called when parsing the command line if necessary. Even if the switch is not supposed to receive a parameter, the callback has to receive a value. Switches without callbacks defined can be obtained with getUnprocessedSwitches() function.

4. Once the command line has been parsed and DIRAC is properly initialized, the rest of the required DIRAC modules can be imported and the script logic can take place:

```python
# Import the required DIRAC modules
from DIRAC.Interfaces.API.DIRAC import DIRAC
from DIRAC import gLogger

# Do stuff... depending on cliParams.raw, cliParams.pingsToDo and servicesList

def executeCommandLogic():
    # Do stuff
    gLogger.notice('This is the result of the command')

if __name__ == '__main__':
    executeCommandLogic()
```

Having understood the logic of the script, there are few good practices that must be followed:

- Use `DIRAC.exit( exitCode )` instead of `sys.exit( exitCode )`
- Encapsulate the command code into functions / classes so that it can be easily tested
- Usage of `gLogger` instead of `print` is mandatory. The information in the normal command execution must be printed out in the NOTICE logging level.
- Use the `if __name__ == "__main__"` close for the actual command execution to avoid running the script when it is imported.

### Example command

Applying all the above recommendations, the command implementation can look like this yet another example:

```python
#!/usr/bin/env python

""
  dirac-my-great-script

  This script prints out how great is it, shows raw queries and sets the
  number of pings.

  Usage:
  dirac-my-great-script [option|cfgfile] <Arguments>
  Arguments:
    <service1> [<service2> ...]

  ""

from DIRAC import S_OK, S_ERROR, gLogger, exit as DIRACExit
from DIRAC.Core.Base import Script

__RCSID__ = '$Id$'

cliParams = None
switchDict = None

class Params:
    ""
    Class holding the parameters raw and pingsToDo, and callbacks for their
```
respective switches.

```python
def __init__( self ):
    self.raw = False
    self.pingsToDo = 1

def setRawResult( self, value ):
    self.raw = True
    return S_OK()

def setNumOfPingsToDo( self, value ):
    try:
        self.pingsToDo = max( 1, int( value ) )
    except ValueError:
        return S_ERROR( "Number of pings to do has to be a number" )
    return S_OK()

def registerSwitches():
    Registers all switches that can be used while calling the script from the
command line interface.

    #Some of the switches have associated a callback, defined on Params class.
    cliParams = Params()

    switches = [
        ( '', 'text=', 'Text to be printed' ),
        ( 'u', 'upper', 'Print text on upper case' ),
        ( 'r', 'showRaw', 'Show raw result from the query', cliParams.setRawResult ),
        ( 'p:', 'numPings=', 'Number of pings to do (by default 1)', cliParams.setNumOfPingsToDo )
    ]

    # Register switches
    for switch in switches:
        Script.registerSwitch( *switch )

    #Define a help message
    Script.setUsageMessage( __doc__ )

def parseSwitches():
    Parse switches and positional arguments given to the script

    #Parse the command line and initialize DIRAC
    Script.parseCommandLine( ignoreErrors = False )

    #Get the list of services
    servicesList = Script.getPositionalArgs()

    gLogger.info( 'This is the servicesList %s:' % servicesList )

    # Gets the rest of the
    switches = dict( Script.getUnprocessedSwitches() )

    gLogger.debug( "The switches used are:")
map( gLogger.debug, switches.iteritems() )

switches[ 'servicesList' ] = servicesList

return switches

def main():
    
    # let's do something
    if not len( switchDict[ 'servicesList' ] ) :
        gLogger.error( 'No services defined' )
        DIRACExit( 1 )
        gLogger.notice( 'We are done' )

    if __name__ == "__main__":

        # Script initialization
        registerSwitches()
        switchDict = parseSwitches()
        #Import the required DIRAC modules
        from DIRAC.Interfaces.API.Dirac import Dirac

        # Run the script
        main()

        # Bye
        DIRACExit( 0 )

3.5.11 DIRAC Utilities

Here are described some useful utilities that can be used for coding DIRAC components

DIRAC CS Helpers

CS Helpers are useful utilities to interact with the Configuration Service. They can be found in:

DIRAC.ConfigurationSystem.Client.Helpers

Helper for accessing /Operations

/Operations section is VO and setup aware. That means that configuration for different VO/setup will have a different CS path:

- For multi-VO installations /Operations/<vo>/<setup> should be used.

- For single-VO installations /Operations/<setup> should be used.

In any case, there is the possibility to define a default configuration, that is valid for all the setups. The Defaults keyword can be used instead of the setup. For instance /Operations/myvo/Defaults.

Parameters defined for a specific setup take precedence over parameters defined for the Defaults setup. Take a look at Operations - Section for further info.

3.5. Developing DIRAC components
DIRAC Documentation, Release integration

To ease accessing the /Operations section a helper has been created. This helper receives the VO and the Setup at instantiation and will calculate the Operations path automatically. Once instanced it’s used as the gConfig object. An example would be:

```python

ops = Operations( vo = 'dirac', setup = 'Production' )
#This would check the following paths and return the first one that is defined
# 1.- /Operations/dirac/Production/JobScheduling/CheckJobLimits
# 2.- /Operations/dirac/Defaults/JobScheduling/CheckJobLimits
# 3.- Return True

print ops.getValue( "JobScheduling/CheckJobLimits", True )
```

It’s not necessary to define the VO if a group is known. The helper can extract the VO from the group. It’s also possible to skip the setup parameter and let it discover itself. For instance:

```python

ops = Operations( group = 'dirac_user' )
```

**Helper for accessing /Resources**

**Utilities for parallel programming**

**ThreadPool**

ThreadPool creates a pool of worker threads to process a queue of tasks much like the producers/consumers paradigm. Users just need to fill the queue with tasks to be executed and worker threads will execute them.

To start working with the ThreadPool first it has to be instanced:

```python
threadPool = ThreadPool( minThreads, maxThreads, maxQueuedRequests )
```

- minThreads - at all times no less than <minThreads> workers will be alive
- maxThreads - at all times no more than <maxThreads> workers will be alive
- maxQueuedRequests - No more than <maxQueuedRequests> can be waiting to be executed. If another request is added to the ThreadPool, the thread will lock until another request is taken out of the queue.

The ThreadPool will automatically increase and decrease the pool of workers as needed.

To add requests to the queue:

```python
threadPool.generateJobAndQueueIt( <functionToExecute>,
    args = ( arg1, arg2, ... ),
    oCallback = <resultCallbackFunction> )
```

or:

```python
request = ThreadedJob( <functionToExecute>,
    args = ( arg1, arg2, ... )
    oCallback = <resultCallbackFunction> )
threadPool.queueJob( request )
```

The result callback and the parameters are optional arguments. Once the requests have been added to the pool. They will be executed as soon as possible. Worker threads automatically return the return value of the requests. To run the result callback functions execute:
threadPool.processRequests()

This method will process the existing return values of the requests. Even if the requests do not return anything this method (or any process result method) has to be called to clean the result queues.

To wait until all the requests are finished and process their result call:

threadPool.processAllRequests()

This function will block until all requests are finished and their result values have been processed.

It is also possible to set the threadPool in auto processing results mode. It’ll process the results as soon as the requests have finished. To enable this mode call:

threadPool.daemonize()

ProcessPool

author Krzysztof Daniel Ciba <Krzysztof.Ciba@NOSPAMgmail.com>
date Tue, 8th Jul 2012
version second and final

The ProcessPool creates a pool of worker sub-processes to handle a queue of tasks much like the producers/consumers paradigm. Users just need to fill the queue with tasks to be executed and worker tasks will execute them.

To construct ProcessPool one first should call its constructor:

\[
\text{pool} = \text{ProcessPool}\left(\text{minSize}, \text{maxSize}, \text{maxQueuedRequests}, \text{strictLimits=True}, \text{poolCallback=None}, \text{poolExceptionCallback=None}\right)
\]

where parameters are:

- **param int minSize** at least \(<\text{minSize}>\) workers will be alive all the time
- **param int maxSize** no more than \(<\text{maxSize}>\) workers will be alive all the time
- **param int maxQueuedRequests** size for request waiting in a queue to be executed
- **param bool strictLimits** flag to kill/terminate idle workers above the limits
- **param callable poolCallback** pool owned results callback
- **param callable poolExceptionCallback** pool owned exception callback

In case another request is added to the full queue, the execution will lock until another request is taken out. The ProcessPool will automatically increase and decrease the pool of workers as needed, of course not exceeding above limits.

To add a task to the queue one should execute:

\[
\text{pool.createAndQueueTask}\left(\text{funcDef}, \text{args = (arg1, arg2, ... )}, \text{kwargs = \{"kwarg1" : value1, "kwarg2" : value2 \}}, \text{taskID = taskID}, \text{callback = callbackDef}, \text{exceptionCallback = exceptionCallbackDef}, \text{usePoolCallbacks = False}, \text{timeOut = 0}, \text{blocking = True}\right)
\]

or alternatively by using ProcessTask instance:
task = ProcessTask( funcDef,
    args = ( arg1, arg2, ... ),
    kwargs = { "kwarg1" : value1, .. },
    callback = callbackDef,
    exceptionCallback = exceptionCallbackDef,
    usePoolCallbacks = False,
    timeOut = 0,
    blocking = True )

pool.queueTask( task )

where parameters are:

- **param callable funcDef**  callable py object definition (function, lambda, class with __call__
  slot defined)
- **param list args**  argument list
- **param dict kwargs**  keyword arguments dictionary
- **param callable callback**  callback function definition (default None)
- **param callable exceptionCallback**  exception callback function definition (default None)
- **param bool usePoolCallbacks**  execute pool callbacks, if defined (default False)
- **param int timeOut**  time limit for execution in seconds (default 0 means no limit)
- **param bool blocking**  flag to block queue until task is en-queued

The *callback*, *exceptionCallback*, *usePoolCallbacks*, *timeOut* and *blocking* parameters are all optional. Once task has been added to the pool, it will be executed as soon as possible. Worker sub-processes automatically return the result of the task. To obtain those results one has to execute:

```python
pool.processRequests()
```

This method will process the existing return values of the task, even if the task does not return anything. This method has to be called to clean the result queues. To wait until all the requests are finished and process their result call:

```python
pool.processAllRequests()
```

This function will block until all requests are finished and their result values have been processed.

It is also possible to set the ProcessPool in daemon mode, in which all results are automatically processed as soon they are available, just after finalization of task execution. To enable this mode one has to call:

```python
pool.daemonize()
```

To monitor if ProcessPool is able to execute a new task one should use ProcessPool.hasFreeSlots() and ProcessPool.isFull(), but boolean values returned could be misleading, especially if en-queued tasks are big.

**Callback functions**  There are two types of callbacks that can be executed for each tasks: exception callback function and results callback function. The first one is executed when unhandled exception has been raised during task processing, and hence no task results are available, otherwise the execution of second callback type is performed. The callback functions can be defined on two different levels:

- directly in ProcessTask, in that case those have to be shelvable/picklable, so they should be defined as global functions with the signature:

  ```python
callback( task, taskResult )
  ```

  where *task* is a ProcessPool.ProcessTask reference and *taskResult* is whatever task callable is returning for results callback and:
exceptionCallback(task, exc_info)

where `exc_info` is a S_ERROR dictionary extended with “Exception”: { “Value”: exceptionName, “Exc_info”: exceptionInfo }

- in the ProcessPool itself, in that case there is no limitation on the function type: it could be a global function or a member function of a class, signatures are the same as before.

The first types of callbacks could be used in case various callable objects are put into the ProcessPool, so you probably want to handle them differently depending on their definitions, while the second types are for executing same type of callables in sub-processes and hence you are expecting the same type of results everywhere.

If both types of callbacks are defined, they will be executed in the following order: task callbacks first, pool callbacks afterwards.

**Timed execution**  One can also put a time limit for execution for a single task, this is done by setting `timeOut` argument in ProcessTask constructor to some integer value above 0. To use this functionality one has to make sure that underlying code is not trapping SIGALRM, which is used internally to break execution after `timeOut` seconds.

**Finalization procedure**  The finalization procedure is not different from Unix shutting down of a system, first ProcessPool puts a special bullet tasks to pending queue, used to break WorkingProcess.run main loop, then SIGTERM is sent to all still alive sub-processes. If some of them are not responding to termination signal, ProcessPool waits a grace period (timeout) before killing of all children by sending SIGKILL.

To use this procedure one has to execute:

```
pool.finalize( timeout = 10 )
```

where `timeout` is a time period in seconds between terminating and killing of sub-processes. The ProcessPool instance can be cleanly destroyed once this method is called.

**WorkingProcess life cycle**  The ProcessPool is creating workers on demand, checking if their is not exceeding required limits. The pool worker life cycle is managed by WorkingProcess itself.
Once created worker is spawning a watchdog thread checking on every 5 seconds PPID of worker. If parent process executing `ProcessPool` instance is dead for some reason (as so the PPID is 1, as orphaned process is adopted by init process), watchdog is sending SIGTERM and SIGKILL signals to the worker main thread in interval of 30 seconds, preventing too long adoption and closing worker life cycle to save system resources.

Just after spawning of a watchdog, the main worker thread starts also to query input task queue. After ten fruitless
attempts (when task queue is empty), it is committing suicide emptying the `ProcessPool` worker’s slot.

When input task queue is not empty and `ProcessTask` is successfully read, `WorkingProcess` is spawning a new thread in which task processing is executed. This task thread is then joined and results are put to the results queue if they are available and ready. If task thread is stuck and task timeout is defined, `WorkingProcess` is stopping task thread forcefully returning `S_ERROR`('Timed out') to the `ProcessPool` results queue.

**Handling errors within DIRAC**

The choice was made not to use exception within DIRAC. The return types are however standardized.

**S_ERROR**

This object is now to be phased out by the `DError` object.

The `S_ERROR` object is basically a dictionary with the ‘OK’ key to `False`, and a key ‘Message’ which contains the actual error message.

```python
from DIRAC import S_ERROR

res = S_ERROR("What a useful error message")

print res
# {'Message': 'What a useful error message', 'OK': False}
```

There are two problems with this approach:

- It is difficult for the caller to react based on the error that happened
- The actual error cause is often lost because replaced with a more generic error message that can be parsed

```python
def func1():
    # Error happening here, with an interesting technical message
    return S_ERROR('No such file or directory')

# returns a similar, but only similar error message
def func2():
    # Error happening here, with an interesting technical message
    return S_ERROR('File not found')

def main():
    ret = callAFunctoin{}

    if not res['OK']:
        if 'No such file' in res['Message']:
            # Handle the error properly
            # Unfortunately not for func2, even though it is the same logic
```

A similar logic is happening when doing the bulk treatment. Traditionally, we have for bulk treatment an `S_OK` returned, which contains as value two dictionaries called ‘Successful’ and ‘Failed’. The ‘Failed’ dictionary contains for each item an error message.

```python
def doSomething(listOfItems):
    successful = {}
    failed = {}

    for item in listOfItems:
```
# execute an operation
res = complicatedStuff(item)

if res['OK']:
    successful[item] = res['Value']
else:
    print "Oh, there was a problem: %s" % res['Message']
    failed[item] = "Could not perform doSomething"

return S_OK('Successful' : successful, 'Failed': failed)

DError

In order to address the problems raised earlier, the DError object has been created. It contains an error code, as well as a technical message. The human readable generic error message is inherent to the error code, in a similar way to what os.strerror is doing.

from DIRAC.Core.Utilities import DError
import errno

def func1():
    # Error happening here, with an interesting technical message
    return DError(errno.ENOENT, 'the interesting technical message')

The interface of this object is fully compatible with S_ERROR

res = DError(errno.ENOENT, 'the interesting technical message')

print res
# No such file or directory ( 2 : the interesting technical message)

print res['OK']
# False

print res['Message']
# No such file or directory ( 2 : the interesting technical message)

# Extra info of the DError object

print res.errno
# 2

print res.errmsg
# the interesting technical message

Another very interesting feature of the DError object is that it keeps the call stack when created, and the stack is displayed in case the object is displayed using gLogger.debug

The Derror object replaces S_ERROR, but should also be used in the Failed dictionary for bulk treatments.

**Handling the error** Since obviously we could not change all the S_ERROR at once, the DError object has been made fully compatible with the old system. This means you could still do something like
res = func1()
if not res['OK']:
    if 'No such file' in res['Message']:
        # Handle the error properly

There is however a much cleaner method which consists in comparing the error returned with an error number, such as ENOENT. Since we have to be compatible with the old system, a utility method has been written `cmpError`.

```python
from DIRAC.Core.Utilities import DErrno
import errno
res = func1()
if not res['OK']:
    if DErrno.cmpError(res, errno.ENOENT):
        # Handle the error properly
```

An important aspect and general rule is to NOT replace the object, unless you have good reasons

```python
# Do that!
def func2():
    res = func1()
    if not res['OK']:
        # I cannot handle it, so I return it AS SUCH
        return res

# DO NOT DO THAT
def func2():
    res = func1()
    if not res['OK']:
        return S_ERROR("func2 failed with %s" % res['Message'])
```

**Error code**  The best practice is to use the errors at your disposal in the standard python module errno. If, for a reason or another, no error there would match your need, there are already “DIRAC standard” errors defined in `DErrno` (Core/Utilities/DErrno.py)

In case the error you would need does not exist yet as a number, there are 5 things you need to do:

- Think whether it really does not match any existing error number
- Declare the global variable corresponding to your error in DErrno.py
- Update the dErrorCode dictionary in DErrno.py
- Update the dStrError dictionary in DErrno.py
- Think again whether you really need that

Refer to the python file for more detailed explanations on these two dictionary. Note that there is a range of number defined for each system (see DErrno.py)

There is a third dictionary that can be filled, which is called `compatErrorString`. This one is used for error comparison. To illustrate its purpose suppose the following existing code:

```python
def func1():
    [...]  
    return S_ERROR("File does not exist")

def main():
    res = func1()
```
You happen to modify \texttt{func1} and decide to return the appropriate \texttt{DError} object, but do not change the \texttt{main} function:

```python
def func1():
    [...]    
    return DError(errno.ENOENT, 'technical message')

def main():
    res = func1()
    if not res['OK']:
        if res['Message'] == "File does not exist":
            # Handle the error properly
```

The test done in the main function will not be satisfied anymore. The cleanest way is obviously to update the test, but if ever this would not be possible, for a reason or another, you could add an entry in the \texttt{compatErrorString} which would state that "File does not exist" is compatible with \texttt{errno.E}NO\texttt{ENT}.

### Extension specific Error codes

In order to add extension specific error, you need to create in your extension the file Core/Utilities/DErrno.py, which will contain:

- \texttt{extra_dErrName}: keys are the error name, values the number of it
- \texttt{extra_dErrorCode}: same as \texttt{dErrName}. keys are the error code, values the name (we don’t simply revert the previous dict in case we do not have a one to one mapping)
- \texttt{extra_dStrError}: same as \texttt{dStrError}, Keys are the error code, values the error description
- \texttt{extra_compatErrorString}: same as \texttt{compatErrorString}. The compatible error strings are added to the existing one, and not replacing them.

Example of extension file:

```python
extra_dErrName = { 'ELHCBSPE' : 3001 }
extra_dErrorCode = { 3001 : 'ELHCBSPE'}
extra_dStrError = { 3001 : "This is a description text of the specific LHCb error" }
extra_compatErrorString = { 3001 : ["living easy, living free"],
                           DErrno.ERRX : ['An error message for ERRX that is specific to LHCb']} # This is added yet another compatible error message for an error defined in the DIRAC DErrno
```

### 3.5.12 Developing Web Portal Pages

### 3.5.13 Code quality

DIRAC code should be coded following the conventions explained in \textit{Coding Conventions}. There are automatic tools that can help you to follow general good code quality rules.

Specifically, \texttt{pylint}, a static code analyzer, can be used. Pylint can give you nice suggestions, and might force you to code in a “standard” way. In any case, to use pylint on DIRAC code we have to supply a configuration file, otherwise pylint will assume that we are coding with standard rules, which is not fully the case: just to say, our choice was to use 2 spaces instead of 4, which is non-standard.

A \texttt{pylint} config file for DIRAC can be found here
Exercise:

Start a new branch from an existing remote one (call it, for example, codeQualityFixes). Run pylint (e.g. via pylint-gui) using the DIRAC.pylint.rc file on a file or 2. Then, commit your changes to your branch. Push this branch to origin, and then ask for a Pull Request using the DIRACGrid github page. Remember to choose the correct remote branch on which your branch should be merged. Remember to add a line or 2 of documentation for your PR.

3.6 Documenting your developments

Where should you document your developments? Well, at several places, indeed, depending on the documentation we are talking about:

3.6.1 Code documentation

This is quite easy indeed. It’s excellent practice to add docstring to your python code. The good part of it is that tools like pyDev can automatically read it. Also your python shell can (try help()), and so does iPython (just use ? for example). Python stores every docstring in the special attribute __doc__.

Pylint will, by default, complain for every method/class/function left without a docstring.

3.6.2 Release documentation

Releases documentation can be found in 2 places: release notes, and github wiki:

- release notes are automatically created from titles of the Pull Requests. So, pay attention to what you put in there.
- The github wiki can contain a section, for each DIRACGrid repository, highlighting update operations, for example the DIRAC releases notes are linked from the DIRAC wiki main page.

3.6.3 Full development documentation

As said at the beginning of this guide, this documentation is in github. It is very easy to contribute to it, and you are welcome to do that. You don’t even have to clone the repository: github lets you edit it online. This documentation is written in RST and it is compiled using sphinx. From time to time, the documentation is built and added to the official documentation on the DiracGrid website.

3.7 Testing (VO)DIRAC

3.7.1 Who should read this document

- All (VO)DIRAC developers should read, at least, the sections about unit tests and integration tests
- All software testers should read fully this document
- All (VO)DIRAC developers coordinators should read fully this document

NB: if you are a developer coordinator, you better be most and foremost, a development instructor, and a software tester.
3.7.2 Why this document should be interesting for you

- Because you want your code to work as expected
- Because preventing disasters is better than fixing them afterwards
- Because it’s your duty, as developer, to verify that a new version of DIRAC fits your VO needs.

3.7.3 What we mean by testing

Every large enough software project needs to be carefully tested, monitored and evaluated to assure that minimum standards of quality are being attained by the development process. A primary purpose of that is to detect software and configuration failures so that defects may be discovered and corrected before making official release and to check if software meets requirements and works as expected. Testing itself could also speed up the development process rapidly tracing problems introduced with the new code.

DIRAC is not different from that scenario, with the exception that service-oriented architecture paradigm, which is one of the basic concepts of the project, making the quality assurance and testing process the real challenge. However as DIRAC becomes more and more popular and now is being used by several different communities, the main question is not: to test or not to test?, but rather: how to test in an efficient way?

The topic of software testing is very complicated by its own nature, but depending on the testing method employed, the testing process itself can be implemented at any time in the development phase and ideally should cover many different levels of the system:

- **unit tests**, in which the responsible person for one source file is proving that his code is written in a right way,
- **integration tests** that should cover whole group of modules combined together to accomplish one well defined task,
- **regression tests** that seek for errors in existing functionality after patches, functionality enhancements and or configuration changes have been made to the software,
- **certification tests (or system tests)**, which are run against the integrated and compiled system, treating it as a black box and trying to evaluate the system’s compliance with its specified requirements.

If your unit tests are not passing, you should not think yet to start the integration tests. Similarly, if your unit tests show some broken software, you should not bother running any system test.

3.7.4 Unit tests

In DIRAC the unit tests should be prepared for the developer herself, integration tests could be developed in groups of code responsible persons, for regression tests the responsible person should be a complete subsystem (i.e. WMS, DMS, SMS etc..) manager, while certification tests should be prepared and performed by release managers.

3.7.5 Tools and methodology

In DIRAC unit tests should be prepared for the developer herself. As the main implementation language is Python, the developers should use its default tool for unit testing, which is already a part of any Python distributions: the unittest module.

This module provides a rich set of tools for constructing and running tests, supporting some very important concepts, like:

- **fixtures**: initialisation needed for setting up a group of tests together with appropriate clean-up after the execution
- **cases**: the smallest unit of testing for one use case scenario
• *suites*: collection of test cases for aggregation of test that should be executed together

• *runners*: classes for executing tests, checking all the spotted asserts and providing output results to the user.

The developers are encouraged to make themselves familiar with unittest module documentation, which could be found [here](#). It is suggested to read at least documentation for `TestCase`, `TestSuite` and `TestLoader` classes and follow the examples over there.

One of the requirements for writing a suitable test is an isolation from depended-on code and the same time from production environment. This could be obtained by objects mocking technique, where all fragile components used in a particular test suite are replaced by their false and dummy equivalents - test doubles. For that it is recommended to use `mock` module, which should be accessible in DIRAC externals for server installation. Hence it is clear that knowledge of `mock` module API is essential.

Unit tests are typically created by the developer who will also write the code that is being tested. The tests may therefore share the same blind spots with the code: for example, a developer does not realize that certain input parameters must be checked, most likely neither the test nor the code will verify these input parameters. If the developer misinterprets the requirements specification for the module being developed, both the tests and the code will be wrong. Hence if the developer is going to prepare her own unit tests, she should pay attention and take extra care to implement proper testing suite, checking for every spot of possible failure (i.e. interactions with other components) and not trusting that someone else’s code is always returning proper type and/or values.

Testing the code, and so proper code developing cycle, can be done in four well defined steps:

Step 1. **Preparation**

The first step on such occasions is to find all possible use cases scenarios. The code is the correct place for constructing input and output data stubs, mock objects that the production code is using from the outside world and initial state of object being tested. Spending more time on this preparation phase will help to understand all possible branches, paths and points of possible failures inside the code and accelerate the second step, which is the test suite implementation.

Amongst all scenarios one is very special - so special, that it even has got its own name: the main success scenario. This is the path in execution process, in which it is assumed that all components are working fine so the system is producing results correct to the last bit. The developer should focus on this scenario first, as all the others are most probably branching from it if some error condition would appear.

Step 2. **Implementation**

Once the set of use cases is well defined, the developer should prepare and implement test case for each of use cases which should define:

• initial and final states of the system being tested,

• runtime configuration,

• set of input values, associated objects and their internal states,

• correct behaviour,

• set of output results.

Each test case should be instrumented with a special method: `setUp`, which is preparing the testing environment. This is the correct place for constructing input and output data stubs, mock objects that the production code is using from the outside world and initial state of object being tested. It is a good practice to implement also second special method: `tearDown`, which is doing a clean up after the tests execution, destroying all objects created inside `setUp` function.

A test case should try to cover as much as possible the API of software under test and the developer is free to decide how many tests and asserts she would be implementing and executing, but of course there should be at least one test method inside each of test cases and at least one assert in every test method. The developer should also keep in her mind that being greedy is not a good practice: her test cases should check only her own code and nothing else.

---

1 Or even better software requirements document, if any of such exists. Otherwise this is a great opportunity to prepare one.
Step 3. **Test execution**

Every developer is encouraged to execute her test suites by herself. Execution code of test suite should be put into unit test module in a various ways. Of course once the test results are obtained, it is the high time for fixing all places in the tested code, in which tests have failed.

Step 4. **Refactoring**

Once the code is tested and all tests are passed, the developer can start thinking about evolution of the code. This includes performance issues, cleaning up the code from repetitions, new features, patching, removing obsolete or not used methods. So from this point the whole developing cycle can start again and again and again...

### 3.7.6 Test doubles

Unit tests should run in *isolation*. Which means that they should run without having DIRAC fully installed, because, remember, they should just test the code logic. If, to run a unit test in DIRAC, you need a dirac.cfg file to be present, you are failing your goal.

To isolate the code being tested from depended-on components it is convenient and sometimes necessary to use *test doubles*: simplified objects or procedures, that behaves and looks like the their real-intended counterparts, but are actually simplified versions that reduce the complexity and facilitate testing. Those fake objects meet the interface requirements of, and stand in for, more complex real ones, allowing programmers to write and unit-test functionality in one area without actually calling complex underlying or collaborating classes. The isolation itself help developers to focus their tests on the behaviour of their classes without worrying about its dependencies, but also may be required under many different circumstance, i.e.:

- if depended-on component may return values or throw exceptions that affect the behaviour of code being tested, but it is impossible or difficult for such cases to occur,
- if results or states from depended-on component are unpredictable (like date, weather conditions, absence of certain records in database etc..),
- if communication with internal states of depended-on component is impossible,
- if call to depended-on component has unacceptable side effects,
- if interactions with depended-on component is resource consuming operation (i.e. database connections and queries),
- if depended-on component is not available or even not existing in the test environment (i.e. the component’s implementation hasn’t stared yet, but its API is well defined).

It is clear that in such cases the developer should try to instrument the test suite with a set doubles, which come is several flavours:

**Dummy** A *dummy object* is an object that is used when method being tested has required object of some type as a parameter, but apart of that neither test suite nor code being tested care about it.

**Stub** A *test stub* is a piece of code that doesn’t actually do anything other than declare itself and the parameters it accepts and returns something that is usually the values expected in one of the scenarios for the caller. This is probably the most popular double used in a test-driven development.

**Mock** A *mock object* is a piece of code, that is used to verify the correct behaviour of code that undergo tests, paying more attention on how it was called and executed inside the test suite. Typically it also includes the functionality of a test stub in that it must return values to the test suite, but the difference is it should also validate if actions that cannot be observed through the public API of code being tested are performed in a correct order.

---

2 To better understand this term, think about a movie industry: if a scene movie makers are going to film is potentially dangerous and unsafe for the leading actor, his place is taken over by a stunt double.
In a dynamically typed language like Python every test double is easy to create as there is no need to simulate the full API of depended-on components and the developer can freely choose only those that are used in her own code.

3.7.7 Example

Nota bene: the example that follows suppose that the reader has already a basic familiarity with some DIRAC constructs. If this is not the case, we suggest the reader to first read Developing DIRAC components.

Let’s assume we are coding a client to the CheeseShopSystem inside DIRAC. The depended-on components are CheeseShopSystem.Service.CheeseShopOwner with CheeseShopSystem.DB.CheeseShopDB database behind it. Our CheeseShopSystem.Client.CheeseShopClient could only ask the owner for a specific cheese or try to buy it

3. We know the answers for all question that have been asked already, there was no cheese at all in original script, but here for teaching purposes we can just pretend for a while that the owner is really checking the shop’s depot and even more, the Cheddar is present. The code for CheeseShopOwner:

```python
from types import *
from DIRAC import S_OK, S_ERROR, gLogger, gConfig
from DIRAC.Core.DISET.RequestHandler import RequestHandler
from DIRAC.CheeseShopSystem.DB.CheeseShopDB import CheeseShopDB

# global instance of a cheese shop database
cheeseShopDB = False

# initialize it first
def initializeCheeseShopOwner( serviceInfo ):
    global cheeseShopDB
    cheeseShopDB = CheeseShopDB()
    return S_OK()

class CheeseShopOwner( RequestHandler ):

    types_isThere = [ StringType ]
    def export_isThere( self, cheese ):
        return cheeseShopDB.isThere( cheese )

    types_buyCheese = [ StringType, FloatType ]
    def export_buyCheese( self, cheese, quantity ):
        return cheeseShopDB.buyCheese( cheese, quantity )

    # ... and so on, so on and so on, i.e:
    types_insertCheese = [ StringType, FloatType, FloatType ]
    def export_insertCheese( self, cheeseName, price, quantity ):
        return cheeseShopDB.insertCheese( cheeseName, price, quantity )
```

And here for CheeseShopClient class:

```python
from DIRAC import S_OK, S_ERROR, gLogger, gConfig
from DIRAC.Core.Base.Client import Client

class Cheese( object ):

    def __init__( self, name ):
        self.name = name

class SpanishInquisitionError( Exception ):
    pass
```

3 And eventually is killing him with a gun. At least in a TV show.

3.7. Testing (VO)DIRAC
This maybe oversimplified code example already has several hot spots of failure for chess buying task: first of all, your input parameters could be wrong (i.e. you want to buy rather parrot, not cheese); the shop owner could be out; they haven’t got cheese you are asking for in the store; or maybe it is there, but not enough for your order; or you haven’t got enough money to pay and at least the transaction itself could be interrupted for some reason (connection lost, database operation failure etc.).

We have skipped CheeseShopDB class implementation on purpose: our CheeseShopClient directly depends on CheeseShopOwner and we shouldn’t care on any deeper dependencies.

Now for our test suite we will assume that there is a 20 lbs of Cheddar priced 9.95 pounds, hence the test case for success is i.e. asking for 1 lb of Cheddar (the main success scenario) having at least 9.95 pounds in a wallet:

- **input:** Cheese("Cheddar"), 1.0 lb, 9.95 pounds in your pocket
- **expected output:** S_OK = { "OK" : True, "Value" : 0.0 }

Other scenarios are:

1. Wrong order ⁴:

---

⁴ You may ask: *isn’t it silly?* No, in fact it isn’t. Validation of input parameters is one of the most important tasks during testing.
• Want to buy Norwegian blue parrot:
  – input: `Parrot("Norwegian Blue")`
  – expected output: an exception `SpanishInquisitionError("It’s stone dead!")` thrown in a client

• Asking for wrong quantity:
  – input: `Cheese("Cheddar"), quantity = "not a number"` or `quantity = 0`
  – expected output: an exception `SpanishInquisitionError("It’s stone dead!")` thrown in a client

3. The shop is closed:
• input: `Cheese("Cheddar")`
• expected output: `S_ERROR = { "OK" : False, "Message" : "Shop is closed!" }`

4. Asking for any other cheese:
• input: `Cheese("Greek feta"), 1.0 lb`
• expected output: `S_ERROR = { "OK" : False, "Message" : "Ah, not as such!" }`

5. Asking for too much of Cheddar:
• input: `Cheese("Cheddar"), 21.0 lb`
• expected output: `S_ERROR = { "OK" : False, "Message" : "Not enough Cheddar, sorry!" }`

6. No money on you to pay the bill:
• input: `Cheese("Cheddar"), 1.0 lb, 8.0 pounds in your pocket`
• expected output: `S_ERROR = { "OK" : False, "Message" : "Not enough money in your pocket, get lost!" }`

7. Some other unexpected problems in underlying components, which by the way we are not going to be test or explore here. *You just can’t test everything, keep track on testing your code!*

The test suite code itself follows:

```python
import unittest
from mock import Mock
from DIRAC import S_OK, S_ERROR
from DIRAC.CheeseShopSystem.Client.CheeseShopClient import Cheese, CheeseShopClient
from DIRAC.CheeseShopSystem.Service.CheeseShopOwner import CheeseShopOwner

class CheeseClientMainSuccessScenario( unittest.TestCase ):

    def setUp( self ):
        # stub, as we are going to use it’s name but nothing else
        self.cheese = Cheese( "Cheddar" )
        # money, dummy
        self.money = 9.95
        # amount, dummy
        self.amount = 1.0
        # real object to use
        self.shopOwner = CheeseShopOwner( "CheeseShop/CheeseShopOwner" )
        # but with mocking of isThere
        self.shopOwner.isThere = Mock( return_value = S_OK( { "Price" : 9.95, "Quantity" : 20.0 } ) )
```

3.7. Testing (VO)DIRAC
# and buyCheese methods
self.shopOwner.buyCheese = Mock()

def tearDown( self ):
    del self.shopOwner
    del self.money
    del self.amount
    del self.cheese

def test_buy( self ):
    client = CheeseShopClient( money = self.money, shopOwner = self.shopOwner )
    # check if test object has been created
    self.assertEqual( isinstance( client, CheeseShopClient), True )
    # and works as expected
    self.assertEqual( client.buy( self.cheese, self.amount ), { "OK" : True, "Value" : 0.0 } )
    # and now for mocked objects
    # asking for cheese
    self.shopOwner.isThere.assert_called_once_with( self.cheese.name )
    # and buying it
    self.shopOwner.buyCheese.assert_called_once_with( self.cheese.name, self.amount )

if __name__ == "__main__":
    unittest.main()
    #testSuite = unittest.TestSuite( [ "CheeseClientMainSuccessScenario" ] )

3.7.8 Conventions

All test modules should follow those conventions:

**T1** Test environment should be shielded from the production one and the same time should mimic it as far as possible.

**T2** All possible interactions with someone else’s code or system components should be dummy and artificial. This could be obtained by proper use of stubs, mock objects and proper set of input data.

**T3** Tests defined in one unit test module should cover one module (in DIRAC case one class) and nothing else.

**T4** The test file name convention should follow the rule: test word concatenated with module name, i.e. in case of CheeseClient module, which implementation is kept CheeseClient.py disk file, the unit test file should be named testCheeseClient.py

**T5** Each TestCase derived class should be named after module name and scenario it is going to test and Scenario world, i.e.: CheeseClientMainSuccessScenario, CheeseClientWrongInputScenario and so on.

**T6** Each unit test module should hold at least one TestCase derived class, ideally a set of test cases or test suites.

**T7** The test modules should be kept as close as possible to the modules they are testing, preferably in a test subdirectory on DIRAC subsystem package directory, i.e: all tests modules for WMS should be kept in DIRAC/WMS/tests directory.

3.7.9 Integration and System tests

Integration and system tests should not be defined at the same level of the unit tests. The reason is that, in order to properly run such tests, an environment might need to be defined.

Integration and system tests do not just run a single module’s code. Instead, they evaluate that the connection between several modules, or the defined environment, is correctly coded.
3.7.10 The TestDIRAC repository

The GIT repository https://github.com/DIRACGrid/TestDIRAC contains some integration and system tests. These tests are not only used for the certification process. Some of them, in fact, might be extremely useful for the developers.

3.7.11 Integration tests

Integration is a quite vague term. Within DIRAC, we define as integration test every test that does not fall in the unit test category, but that does not need external systems to complete. Usually, for example, you won’t be able to run an integration test if you have not added something in the CS. This is still vague, so better look at an example

This test submits few very simple jobs. Where? Locally. The API DIRAC.Interfaces.API.Job.Job contains a runLocal() method. Admittently, this method is here almost only for testing purposes.

Submitting a job locally means instructing DIRAC to consider your machine as a worker node. To run this test, you’ll have to add few lines to your local dirac.cfg:

```
LocalSite
{
    Site = DIRAC.mySite.local
    CPUScalingFactor = 0.0
    #SharedArea = /cvmfs/lhcb.cern.ch/lib
    #LocalArea =/home/some/local/LocalArea
    GridCE = my.CE.local
    CEQueue = myQueue
    Architecture = x86_64-slc5
    #CPUTimeLeft = 200000
    CPUNormalizationFactor = 10.0
}
```

These kind of tests can be extremely useful if you use the Job API and the DIRAC workflow to make your jobs.

Another example of integration tests are tests of the chain:

```
Client -> Service -> DB
```

They supposes that the DB is present, and that the service is running. Indeed, usually in DIRAC you need to access a DB, write and read from it. So, you develop a DB class holding such basic interaction. Then, you develop a Service (Handler) that will look into it. Lastly, a Client will hold the logic, and will use the Service to connect to the DB. Just to say, an example of such a chain is:

```
TransformationClient -> TransformationManagerHandler -> TransformationDB
```

And this is tested in https://github.com/DIRACGrid/TestDIRAC/blob/master/System/TransformationSystem/TestClientTransformation.py

The test code itself contains something as simple as a series of put/delete, but running such test can solve you few headaches before committing your code.

Tipically, other requirements might be needed for the integration tests to run. For example, one requirement might be that the DB should be empty.

Integration tests, as unit tests, are coded by the developers. Suppose you modified the code of a DB for which its integration test already exist: it is a good idea to run the test, and verify its result.

Within section Developing DIRAC components we will develop one of these tests as an exercise.
3.7.12 Validation and System tests

Validation and System tests are black-box tests. As such, coding them should not require knowledge of the inner design of the code or logic. At the same time, to run them you’ll require a DIRAC server installation. Examples of a system test might be: send jobs on the Grid, and expecting them to be completed after hours. Or, replicate a file or two.

Validation and system tests are usually coded by software testers. The TestDIRAC repository contains a minimal set of test jobs, but since most of the test jobs that you can run are VO specific, we suggest you to expand the list.

3.7.13 Continuous Integration software

There are several tools, on the free market, for so-called Continuous Integration, or simply CI. The most used right now is probably Jenkins, which is also our recommendation. If you have looked in the TestDIRAC repository (and if you haven’t yet, you should, now!) you will see also a Jenkins folder.

What can Jenkins do for you? Several things, in fact:

- it can run all the unit tests
- it can run Pylint (of which we didn’t talk about yet, but, that you should use, and for which it exists a nice documentation that you should probably read) (ah, use this file as configuration file.)
- (not so surprisingly) it can run all the integration tests
- (with some tuning) it can run some of the system tests

For example, the TestDIRAC.Jenkins.dirac_ci.sh adds some nice stuff, like:

- a function to install DIRAC (yes, fully), configure it, install all the databases, install all the services, and run them!
- a function that runs the Pilot, so that a Jenkins node will look exactly like a Grid WN. Just, it will not start running the JobAgent

What can you do with those above? You can run the Integration tests you read above!

How do I do that?

- you need a MySQL DB somewhere. Empty. To be used only for testing purposes.
- you need to configure the Jenkins jobs. What follows is an example of Jenkins job for system tests:

```
#!/bin/bash

export DIRACBRANCH=v6r13
export PRERELEASE=True
export DEBUG=True
export DB_USER=Dirac
export DB_PASSWORD=password
export DB_ROOTUSER=admin
export DB_ROOTPWD=password
export DB_HOST=db-test.example.org
export DB_PORT=5501

git clone git://github.com/DIRACGrid/TestDIRAC.git
source TestDIRAC/Jenkins/dirac_ci.sh

clone
```
fullInstallDIRAC

```bash
fullInstallDIRAC

echo "**** INSTALLATION DONE ****

echo "**** STARTING INTEGRATION TESTS ****"

echo "**** FRAMEWORK TESTS (partially skipped) ****"

python $WORKSPACE/TestDIRAC/Integration/Framework/testInstalledComponentsDB.py -dd

# python $WORKSPACE/TestDIRAC/Integration/Framework/testComponentInstallUninstall.py -dd

echo "**** RMS TESTS ****

python $WORKSPACE/TestDIRAC/Integration/RequestManagementSystem/TestClientReq.py -dd

echo "**** WMS TESTS ****

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestJobDB.py -dd

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestJobLoggingDB.py -dd

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestTaskQueueDB.py -dd

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestClientWMS.py $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/sb.cfg -dd

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestSandboxStoreClient.py $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/sb.cfg -dd

python $WORKSPACE/TestDIRAC/Integration/WorkloadManagementSystem/TestJobWrapper.py -dd

```

echo "**** DMS TESTS ****"

```bash
# Run the DFC test as user without admin privileges

echo "Getting a non privileged user"

dirac-proxy-init -C $WORKSPACE/user/client.pem -K $WORKSPACE/user/client.key $DEBUG

python $WORKSPACE/TestDIRAC/Integration/DataManagementSystem/TestClientDFC.py -dd

# Run it with the admin privileges

```bash
echo "getting the prod role again"

dirac-proxy-init -g prod -C $WORKSPACE/user/client.pem -K $WORKSPACE/user/client.key $DEBUG

python $WORKSPACE/TestDIRAC/Integration/DataManagementSystem/TestClientDFC.py -dd

python $WORKSPACE/TestDIRAC/Integration/DataManagementSystem/TestClientFTS.py -dd

```bash

echo "**** TS TESTS ****

python $WORKSPACE/TestDIRAC/Integration/TransformationSystem/TestClientTransformation.py -dd


```bash
echo "**** TESTS OVER ****

echo "**** Now stopping/removing stuff ****"

clean

echo "*** DONE ****"
```

### 3.7.14 Footnotes

#### Table of contents

- Framework Overview
  - Static Component Monitoring

3.7. Testing (VO)DIRAC
3.8 Framework Overview

Information regarding use of the DIRAC Framework to build new components

3.8.1 DISET Stable connections

DISET is the communication, authorization and authentication framework of top of which DIRAC services are built. Traditionally DISET offered RPC and file transfer capabilities. Those communication mechanisms are not well suited for the Executor framework. RPC doesn’t allow the server to send data to the clients asynchronously, and each RPC query requires establishing a new connection and going through another SSL handshake. On average the SSL process is the most resource consuming part of the request.

The Executor framework relies on a new DISET capability. Support for stable connections and asynchronous requests has been added. Any component can open a connection and reuse it to send and receive requests though it. Services can send information to clients without having to wait for the clients to ask for them as shown in the stable connections figure.

Although once connected services can send data asynchronously to clients, services are still servers and require clients to start the connection to them. No service can start the connection towards the client. Once the service has received the connection the asynchronous data transfer can take place.

Server side usage

Any DIRAC service can make use of the stable connection mechanism. It’s usage is quite similar to the usual RPC mechanism but with extended capabilities. Here we have an example of a service using the stable connections mechanism:

```
1 import types
2 from DIRAC import S_OK, S_ERROR
3 from DIRAC.Core.DISET.RequestHandler import RequestHandler
```
The first thing the server requires is a definition of the messages that it can use. In the example, lines 7 and 8 define two messages: Ping and Pong messages. Each message has one attribute called id that can only be either an integer or a long. Lines 10-22 define the connection callback conn_connected. Whenever the client receives a new client connection this function will be called. This function receives three parameters:

trid Transport identifier. Each client connection will have a unique id. If a client reconnects it will have a different trid each time.

identity Client identifier. Each client will have a unique id. This id will be maintained across reconnects.

kwargs Dictionary containing keyword arguments sent by client when connecting.

If this function doesn’t return S_OK the client connection will be rejected.

If a client drops the connection, method conn_drop will be called with the trid of the disconnected client to allow the
handler to clean up its state regarding that client if necessary.

Lines 32-46 define callback for Ping message. All message callbacks will receive only one parameter. The parameter will be an object containing the message data. As seen in line 37, the message object will have defined the attributes previously defined with the values the client is sending. Accessing them is as easy as just accessing normal attributes. On line 38 the Pong message is created and then assigned a value in to the id attribute on line 43. Finally the message is sent back to the client using srv_msgSend with the client trid as first parameter and the Pong message as second one. To just reply to a client there’s a shortcut function srv_msgReply. If any message callback doesn’t return S_OK the client will be disconnected.

In the example there’s no callback for the Pong message because not all services may have to react to all messages. Some messages will only make sense to be sent to clients not received from them. If the Service receives the Pong message, it will send an error back to the client and disconnect it.

**Client side usage**

Clients do not have to define which messages they can use. The Message client will automatically discover those based on the service to which they are connecting. Here’s an example on how a client could look like:

```python
import sys
import time
from DIRAC import S_OK, S_ERROR
from DIRAC.Core.Base import Script
from DIRAC.Core.DISET.MessageClient import MessageClient

Script.parseCommandLine()

def sendPingMsg( msgClient, pingid = 0 ):
    
    """Send Ping message to the server""

    result = msgClient.createMessage( "Ping" )
    if not result[ 'OK' ]:
        return result
    msgObj = result[ 'Value' ]
    msgObj.id = pingid
    return msgClient.sendMessage( msgObj )

def pongCB( msgObj ):
    
    """Callback for the Pong message.
    Just send a Ping message incrementing in 1 the id""

    pongid = msgObj.id
    print "RECEIVED PONG %d" % pongid
    return sendPingMsg( msgObj.msgClient, pongid + 1 )

def disconnectedCB( msgClient ):
    """
    Reconnect :)
    """

    retryCount = 0
    while retryCount:
        result = msgClient.connect()
        if result[ 'OK' ]:
            return result
        time.sleep( 1 )
        retryCount -= 1
```
Let's start with line 39 onwards. The client app is instancing a `MessageClient` pointing to the desired service. After that it registers all the callbacks it needs. One for receiving `Pong` messages and one for reacting to disconnects. After that it just connects to the server and sends the first `Ping` message. Lastly it will just wait 10 seconds before exiting.

Function `sendPingMsg` in line 5 onwards just creates a `Ping` message and sends it to the server via the supplied `msgClient`.

The `pongCB` function will be executed for each `Pong` message received. Messages received on the client callbacks have a special attribute `msgClient` with the client that has received the message. If this attribute is accessed in services it will just return `None`.

Function `disconnectedCB` will be invoked if the client is disconnected from the service. In the example it will just try to reconnect for some time and then exit if it doesn’t manage to do so.

### 3.8.2 Static Component Monitoring

The Component Monitoring system takes care of logging information about the components that have been installed and uninstalled in different hosts, like the date or author of the change. The following figure illustrates how different components from this system communicate with each other:
All of the static information is stored in a MySQL database, InstalledComponentsDB. This database contains 3 tables, as illustrated below:

* InstallTools is replaced by ComponentInstaller in v6r15
The InstalledComponentsDB.py file in the Framework system defines all the tables and their relationships using SQLAlchemy, as well as functions to access and modify the values in the database. The following code shows the definition of the ‘Component’ class:

```python
class Component(Base):
    __tablename__ = 'Components'
    __table_args__ = {
        'mysql_engine': 'InnoDB',
        'mysql_charset': 'utf8'
    }

    componentID = Column('ComponentID', Integer, primary_key = True)
    system = Column('System', String(32), nullable = False)
```

3.8. Framework Overview
As can be seen, it is fairly easy to define a new class/table. The only thing that might seem off is the self-installationList field, as it has not been ‘declared’ before. This field acts as a back reference for the InstalledComponent table (it is a list of all the installations the component is associated to, i.e., a list of InstalledComponent objects). This reference is completed in the InstalledComponent class definition with the addition of the ‘installationComponent’ field:

```
class InstalledComponent(Base):
    """
    This class defines the schema of the InstalledComponents table in the
    InstalledComponentsDB database
    """
    __tablename__ = 'InstalledComponents'
    __table_args__ = {
        'mysql_engine': 'InnoDB',
        'mysql_charset': 'utf8'
    }
    componentID = Column('ComponentID',
        Integer,
        ForeignKey('Components.ComponentID'),
        primary_key = True)
    hostID = Column('HostID',
        Integer,
        ForeignKey('Hosts.HostID'),
        primary_key = True)
    instance = Column('Instance',
        String(32),
        primary_key = True)
    installationTime = Column('InstallationTime',
        DateTime,
        primary_key = True)
    unInstallationTime = Column('UnInstallationTime',
        DateTime)
    installedBy = Column('InstalledBy', String(32))
    unInstalledBy = Column('UnInstalledBy', String(32))
    installationComponent = relationship('Component',
        backref = 'installationList')
```

Although we are using MySQL here, it is possible to switch to other SQLAlchemy-supported databases by changing the URI of the database in the initialization methods to point to the one being used instead.

For instance, from:

```python
self.engine = create_engine( 'mysql://%s:%s@%s:%s/%s' %
    ( self.user, self.password, self.host, self.port, self.db ),
    pool_recycle = 3600, echo_pool = True
)
```

to:

```python
```
The ComponentMonitoring service acts as an interface between the client side and the functionalities provided by InstalledComponentsDB (accessing and modifying the database). Clients to this service are created to modify the database or access its data. The MonitoringUtilities module provides the functionality needed to store or delete monitoring entries from the database:

```python
from DIRAC.FrameworkSystem.Utilities import MonitoringUtilities

# Add an entry to the DB for the SysAdmin service
result = MonitoringUtilities.monitorInstallation( 'service', 'Framework', 'SystemAdministrator' )
if not result[ 'OK' ]:
    print 'Something went wrong'
```

### 3.9 REST Interface

DIRAC has been extended to provide the previously described language agnostic API. This new API follows the REST style over HTML using JSON as the serialization format. OAuth2 is used as the credentials delegation mechanism to the applications. All three technologies are widely used and have bindings already made for most of today’s modern languages. By providing this new API DIRAC can now be interfaced to any component written in most of today’s modern languages.

The REST interface endpoint is an HTTPS server provided in the RESTDIRAC module. This HTTPS server requires Tornado. If you don’t have it installed just do:

```bash
pip install -U "tornado>=2.4"
```

All requests to the REST API are HTTP requests. For more info about REST take a look here. From here on a basic understanding of the HTTP protocol is assumed.

#### 3.9.1 OAuth2 authentication

Whenever an application wants to use the API, DIRAC needs to know on behalf of which user the application is making the request. Users have to grant privileges to the application so DIRAC knows what to do with the request. Apps have to follow a OAuth2 flow to get a token that has user assigned privileges. There are two different flows to get a token depending on the app having access to the user certificate. Both flows are one or more HTTP queries to the REST server.

- If the app has access to the user certificate it has to make a GET request to /oauth2/token using the user certificate as the client certificate. That request has to include as GET parameters:
  - `grant_type` set to `client_credentials`
  - `group` set to the dirac group the token is being request for.
    * To retrieve a list of valid groups for a certificate, make a GET request to /oauth2/groups using the certificate.
  - `setup` set to the dirac setup the token is being request for.
    * To retrieve a list of valid setups for a certificate, make a GET request to /oauth2/setups using the certificate.

- If the app does not have access to the user certificate (for instance a web portal) it has to:
  1. Redirect the user to /oauth2/auth passing as GET parameters:
– `response_type` set to `code`. This is a mandatory parameter.
– `client_id` set to the identifier given to you when the app was registered in DIRAC. This is a mandatory parameter.
– `redirect_uri` set to the URL where the user will be redirected after the request has been authorized. Optional.
– `state` set to any value set by the app to maintain state between the request and the callback.

2. Once the user has authorized the request, it will be redirected to the `redirect_uri` defined either in the request or in the app registration in DIRAC. The user request will carry the following parameters:
– `code` set to a temporal token
– `state` set the the original value

3. Exchange the `code` token for the final one. Make a `GET` request to `/oauth2/token` with:
– `grant_type` set to `authorization_code`. Mandatory.
– `code` set to the temporal token received by the client.
– `redirect_uri` set to the original `redirect_uri` if it was defined in step 1
– `client_id` set to the identifier. Same as in step 1.

4. Receive access token :)

From now on. All requests to the REST API have to bear the access token either as:

• `GET access_token` parameter
• `Authorization` header with form “tokendata Bearer”

For more info check out the OAuth2 draft.

### 3.9.2 REST API Resources

Once the app has a valid access token, it can use the REST API. All data sent or received will be serialized in JSON.

#### Job management

**GET /jobs** Retrieve a list of jobs matching the requirements. Parameters:

• `allOwners`: Show jobs from all owners instead of just the current user. By default is set to `false`.
• `maxJobs`: Maximum number of jobs to retrieve. By default is set to `100`.
• `startJob`: Starting job for the query. By default is set to `0`.
• Any job attribute can also be defined as a restriction in a HTTP list form. For instance:

  ```
  Site=DIRAC.Site.com&Site=DIRAC.Site2.com&Status=Waiting
  ```

**GET /jobs/<jid>** Retrieve info about job with id=*jid*  
**GET /jobs/<jid>/manifest** Retrieve the job manifest  
**GET /jobs/<jid>/inputsandbox** Retrieve the job input sandbox  
**GET /jobs/<jid>/outputsandbox** Retrieve the job output sandbox
POST /jobs  Submit a job. The API expects a manifest to be sent as a JSON object. Files can also be sent as a multipart request. If files are sent, they will be added to the input sandbox and the manifest will be modified accordingly. An example of manifest can be:

```json
{
    Executable: "/bin/echo",
    Arguments: "Hello World",
    Sites: [ "DIRAC.Site.com", "DIRAC.Site2.com" ]
}
```

DELETE /jobs/<jid>  Kill a job. The user has to have privileges over a job.

**File catalogue**

All directories that have to be set in a URL have to be encoded in url safe base 64 (RFC 4648 Spec where ‘+’ is encoded as ‘-‘ and ‘/’ is encoded as ‘_‘). There are several implementations for different languages already.

An example in python of the url safe base 64 encoding would be:

```python
>>> import base64
>>> base64.urlsafe_b64encode( "/" )
'Lw=='
```

Most of the search queries accept a metadata condition. This condition has to be coded as a GET query string of key value pairs. Each key can be a metadata field and its value has to have the form ‘operation|value’. The operation depends on the type of metadata field. For integers valid operations are ‘<’, ‘>’, ‘=’, ‘<=’, ‘>=’ and the value has to be a number. For string fields the operation has to be ‘in’ and the value has to be a comma separated list of possible values. An example would be:

```text
someNumberField=>|4.2&someStrangeName=in|name1,name2
```

**GET /filecatalogue/metadata**  Retrieve all metadata keys with their type and possible values that are compatible with the metadata restriction. *Accepts metadata condition*

**GET /filecatalogue/directory/<directory>**  Retrieve contents of the specified directory. Set parameter `verbose` to true to get extended information.

**GET /filecatalogue/directory/<directory>/metadata**  Retrieve metadata values for this directory compatible with the metadata condition. *Accepts metadata condition*

**GET /filecatalogue/directory/<directory>/search**  Search from this directory subdirectories that match the requested metadata search. Each directory will also have the amount of files it contains and their total size. *Accepts metadata condition*

**GET /filecatalogue/file/<file>/attributes**  Get the file information

**GET /filecatalogue/file/<file>/metadata**  Get the file metadata

### 3.10 WebAppDIRAC

The new DIRAC web framework provides many facilities to develop and test web applications. This framework loads each application in a separate window, and these windows can be arranged at the desktop area by means of resizing, moving and pinning. In this tutorial we are going to explain the ways of developing and testing new applications.

Before you start this tutorial, it is desirable that you have some experience with programming in Python, JavaScript, HTML, CSS scripting, client-server communication (such as AJAX and web sockets) and sufficient knowledge in object-oriented programming. If you are not familiar with some of the web technologies, or there has been a while
since you used those technologies, please visit the W3CSchool web site (http://www.w3schools.com/). There, you can find tutorials that you can use to learn or to refresh your knowledge for web-programming.

### 3.10.1 Setup Eclipse

In this section we introduce the tools which can be used for developing web applications based on ExtJS. You can use different editors to write javascript code such as PICO, VI, gEdit, etc. I encourage you to do not use text editors, instead use an Integrated Development Environment (IDE). You can found various IDEs in the market. We propose to use Eclipse as it provides lot of plugins, which can help for debugging or coding javascript. In addition it is free.

- **Using IDEs**
- **Install Eclipse**
- **Install ExtJS**
- **Eclipse and ExtJS**

#### Using IDEs

Text editors are used to write text, but they are not used to write efficient code. We would like to highlight some disadvantages of the text editors:

- code quality: It is not easy to have same code style.
- missing the auto-complete feature
- it is not easy to manage the code

Advantages of the IDEs:

- code quality: Each developer can use the same template
- auto-complete feature: When you type a class name and after press a dot the IDE show the possible methods as well as a short description of the method
- easy to manage the code
- it is easy to create tasks: When required to change some code in the comment we can add //TODO and text; This will appears a Tasks list
- easy to navigate between classes. etc.

#### Install Eclipse

- You can download from: [Eclipse IDE](http://www.eclipse.org)
- installation instructions can be found: [Eclipse wiki](http://wiki.eclipse.org)

#### Install ExtJS

- download from [Sencha page](http://www.sencha.com) and un-zip it. Note if you have installed WebAppDIRAC, you can found it under WebApp/static/extjs directory.
Eclipse and ExtJS

We used the DuckQuoc’s blog to set up our Eclipse. There is another page when you can read about how to setup Eclipse in Spket page.

We use Indigo Eclipse and Spket plugin for developing better javascript code.

Install Spket plugin:

1. Click Help -> Install New software... The following form Install form will appear:

![Install form](image)

Please give a name and use the following link: [http://www.agpad.com/update/](http://www.agpad.com/update/)

1. Click Ok -> select all components
2. Accept the term and conditions -> Finish
3. Wait till the package will be downloaded and installed in case of warning click OK.
4. Restart Eclipse (If it will not restart automatically)

Create Spket profile for ExtJs (Configuration panel):

- Click “Eclipse” -> “Preferences...” You will see the following configuration form:

![Configuration form](image)

- select “Spket JavaScript Profile” and click to the New button and then type ExtJs.

- Click “Add Library” select ExtJs
• Click “Add Folder” you have to add the path of the ExtJs folder (more details in [https://github.com/DIRACGrid/WebAppDIRAC/wiki/_preview#wiki-extjs](https://github.com/DIRACGrid/WebAppDIRAC/wiki/_preview#wiki-extjs) section).

**Make default JavaScript profile**

• In the same window (“Spket JavaScript Profile”) click on the Extjs profile and after make it default by clicking on the “Default” button.

• in the “Configuration panel” click on the “General”->”Editors”->”File Associations”

[Image](https://zmathe.web.cern.ch/zmathe/spketauto.png)

• Please select *.js and then select “Spket JavaScript Editor” and click on the “Default button”

• Restart Eclipse.

**Auto-complete feature**

After the restart you can create a javascript file and try type Ext. and Ctrl+Space

https://zmathe.web.cern.ch/zmathe/spketauto.png

**Code convention**

We use similar code convention to DIRAC. We have created a template used to format the code. You can download from [https://zmathe.web.cern.ch/zmathe/extjs-template.xml](https://zmathe.web.cern.ch/zmathe/extjs-template.xml). In order to use the template you have to import to your Spket profile:

1. Click “Eclipse” -> “Preferences…”
2. In the “Preferences” window select “Spket->Editors->JavaScript Editor->Formatter”
3. Click on the “Import button”
4. Apply
NOTE:

If you encounter some problem, please check your java jdk. We tested with jdk6 and jdk7. We did not discover any problem using those versions.

3.10.2 Install WebAppDIRAC

You have already prepared your eclipse. Now you can try to install DIRAC and the Web portal locally. The instruction is given for MAC OS users, but it is similar to Linux users as well. I use different directory for developing WebAppDIRAC than the directory where the portal is installed. You can link the directory where you develop the WebAppDIRAC to where the Web portal installed or you can copy the code from the development area to the installed area.

Install DIRAC & WebAppDIRAC

We propose to read the following documentation and after continue to install DIRAC https://github.com/DIRACGrid/DIRAC/wiki/GitSetup.

1. Create a directory where you will install DIRAC and WebAppDIRAC: mkdir portal; cd portal
2. git clone git://github.com/zmathe/DIRAC.git. (NOTE: This works when you forked the DIRAC repository) or execute: git clone https://github.com/DIRACGrid/DIRAC.git
3. git clone git://github.com/zmathe/WebAppDIRAC.git (NOTE: This works when you forked the WebAppDIRAC repository on github) or git clone https://github.com/DIRACGrid/WebAppDIRAC.git ./scripts/dirac-install -r v6r10p15 -X -t server or ./DIRAC/Core/scripts/dirac-install.py -r v6r10p15 -X -t server (You can use the current production version of DIRAC which can found http://diracgrid.org.) If DIRAC properly installed, you can continue to install WebAppDIRAC. NOTE: In case of timeout use: ./DIRAC/Core/scripts/dirac-install.py -r v6r10p15 -X -t server -T 600000
4. python DIRAC/Core/scripts/dirac-deploy-scripts.py
5. ./WebAppDIRAC/dirac-postInstall.py
6. source bashrc (we have to use the correct python in order to install tornado)
7. pip install tornado
8. mkdir etc
9. you need to create: vi etc/dirac.cfg file

For example:

```plaintext
DIRAC {
    Setup = LHCb-Development
    #Setup = LHCb-Production
    #Setup = LHCb-Certification
    Configuration {
        Servers = dips://lhcb-conf-dirac.cern.ch:9135/Configuration/Server
        Servers += dips://lhcbprod.pic.es:9135/Configuration/Server
    }
}
```

Note: It is an LHCb specific configuration. You have to use your Configuration servers

**Quick install**

- python dirac-install -t server $installCfg
- source $installDir/bashrc
- dirac-configure $installCfg $DEBUG
- dirac-setup-site $DEBUG

**Start the web framework**

1. You need the grid-certificates under etc directory. If you do not known about it, please ask the appropriate developer.
2. python WebAppDIRAC/scripts/dirac-webapp-run.py -ddd Use firefox/safari/chrome... and open the following url: https://localhost:8443/DIRAC

### 3.10.3 Developing new web application

The new DIRAC web framework provides many facilities to develop and test web applications. This framework loads each application:

- in a separate window, and these windows can be arranged at the desktop area by means of resizing, moving and pinning.
- in a separate tab and these tabs can be customized.

In this tutorial we are going to explain the ways of developing and testing new applications. Before you start this tutorial, it is desirable that you have some experience with programming in Python, JavaScript, HTML, CSS scripting, client-server communication (such as AJAX and web sockets) and sufficient knowledge in object-oriented programming. If you are not familiar with some of the web technologies, or there has been a while since you used those technologies, please visit the W3CSchool web site (http://www.w3schools.com/). There, you can find tutorials that you can use to learn or to refresh your knowledge for web-programming. As well we suggest to read *Setup Eclipse* section.

Each application consists of two parts:

- Client side (CS): Builds the user interface and communicates with the web server in order to get necessary data and show it appropriately.
• Server side (SS): Provides services to the client side run in browser.

The folder structure of the server side web installation is as follows:

• <Module name folder such as DIRAC, LHCbDIRAC, WebAppDIRAC>
  – WebApp
    * __init__.py
    * handler: contains all the server side implementations of the framework and all the applications.
      – __init__.py
    * static: contains all the static content that can be loaded by the client side such as JavaScript files, images and css
    · <Module name folder such as DIRAC, LHCbDIRAC, WebAppDIRAC>: contains the client side implementation of each application
    · template: contains all the templates used by the files in the handler folder

In order to explain how to develop an application, we will go step by step creating an example one. We will name it MyApp.

**Server side**

Each application server side logic is implemented in one Python file. The name of the file is formed by appending the word **Handler** to the name of the application. In the case of the application we want to build, the name of the Python file should be **MyAppHandler**. This file has to be located into the **handler** folder.

Be aware that if this file is not defined in the folder, the application is not going to appear in the main menu.

This file defines a Python class responsible for all server side functionality of MyApp. The class has to extend **WebHandler** class which is the base class for all server side applications handling clients requests. The starting definition of this class is as follows:

```python
from WebAppDIRAC.Lib.WebHandler import WebHandler

class MyAppHandler(WebHandler):
```

For each type of client request there must be an entry point i.e. a method that will be invoked when a clients requests arrive at the server. Let’s say that the URL of the requested method is MyApp/getData. Therefore the name of the class is **MyAppHandler** and the name of the method within the class will be **web_getData**. This means that if you want a method to be accessible in the application class you have to put the prefix **web_** to the name of the method:

```python
from WebAppDIRAC.Lib.WebHandler import WebHandler

class MyAppHandler(WebHandler):
  def web_getData(self):
    self.write({"data":[1,2,3,4]})
```

In order to send back response to the client, we can use the **write** method of the **WebHandler** class. This method whenever invoked, sends to the client the value given as a parameter. If the value is of type dictionary, then the dictionary is converted to JSON string before it is sent back to the client.
The server handles all requests one-by-one which means that the server does not handle the next request until the current one is finished. This mechanism becomes a bottleneck if one request lasts longer and increases the response time for each subsequent request waiting in the server queue until the previous one has finished. Thus the server provides a way to asynchronously handle clients’ requests and mitigate this obstacle. Read the following link and tutorial for further information https://github.com/DIRACGrid/WebAppDIRAC/wiki/Asynchronous-handling-mechanisms-of-clients%27-requests

Any other method that is not an entry point, can have any arbitrary name satisfying the rules of the Python programming language.

Usually the clients requests come with parameters that contain data. In order to access a parameter, you have to use the following expression:

```
self.request.arguments["parameter_name"]
```

or in a full example:

```python
def web_ping(self):
    pingValue = self.request.arguments["ping_val"]
    self.write({"pong_val": pingValue})
```

Every parameter value is enclosed by a list by default so the 0-index stands for taking the value out of the list.

## Client side

The CS side consists of files needed for rendering the UI and communicating with the server side. Technologies used are JavaScript with ExtJS4.x, HTML and CSS. The files of the CS are located into the `static/<Module name folder>` such as `DIRAC, LHCbDIRAC, WebAppDIRAC>` folder and are organized as follows:

- **MyApp**: this folder is named after the name of the application we want to build. It contains all the files regarding this application's side.
  - **build**: this folder contains the compiled version of the javascript files contained in the classes folder
  - **classes**: this folder contains the javascript file that defines the main ExtJS class representing the application on the client side. The name of the file must have the same name as the application we want to build. The file must be created no matter it contains some code or not.
    - *MyApp.js*: this mandatory file contains the main ExtJS class representing the application on the client side. The name of the file must have the same name as the application we want to build.
  - **css**: this folder contains all the css files specific to this application.
    - *MyApp.css*: this mandatory file contains the css style needed by some of the components of the application. The name of the file must have the same name as the application we want to build. The file must be created no matter it contains some code or not.
  - **images**: this folder contains all the specific images and icons needed by this application.

The most important part of all files and folders is the file that contains the main ExtJS class representing the application on the client side (in our case that is `MyApp.js`).

This file defines a ExtJS class responsible for all client side functionality of `MyApp`. This class extends `Ext.dirac.core.Module` class which is the base class for all applications. The starting definition of this class is as follows:
When extending the base class, there are some mandatory methods to be implemented within the derived class:

- **initComponent**: this method is called by the constructor of the application. In this method you can set up the title of the application, its width and height, its maximized state, starting position on the screen and the icon css class. Here it is suitable to set up the layout of the entire application. For further information regarding ExtJS component layouts refer to [http://docs.sencha.com/extjs/4.2.1/extjs-build/examples/layout-browser/layout-browser.html](http://docs.sencha.com/extjs/4.2.1/extjs-build/examples/layout-browser/layout-browser.html).

- **buildUI**: this method is used to build the user interface. Usually this is done by instantiating ExtJS widgets. These instances are added to the application in a way prescribed by the layout which is defined in the initComponent method. This method is called after all the CSS files regarding this application have been successfully loaded.

- **getStateData**: The DIRAC web framework provides a generic way to save and load states of an application. This method is not mandatory, and it can be overridden by a new implementation in the application class. Whenever the user saves an application state, this method is called in order to take the data defining the current state of the application. The data has to be a JavaScript object.

- **loadState(data)**: When we want to load a state, this method is being called. As an argument the framework provides the data that have been saved previously for that state.

The framework already defines handlers for some events related to the windows instances in which the applications are loaded. However there are cases when the developer would like to define some additional actions that have to be executed when those events appear.

In order to access the window object containing the instance of an application, you can use the method `getContainer()`.

For example, suppose we have an image shown inside an application. Suppose we want to resize the image whenever the window gets resized. So the code that we need in order to support this functionality is as follows (in the following code `this` refers to the application object):

```javascript
this.getContainer().__dirac_resize = function(oWindow, iWidth, iHeight, eOpts) {
    this.__oprResizeImageAccordingToWindow(image, oWindow);
}
```

### DIRAC reserved variables and constants

The DIRAC web framework provides a set of global variables and constants. These constants and variables can be accessed anywhere in the code.

- **GLOBAL.APP**: A reference to the main object representing the entire framework. The most important references provided by this reference are as follows:
  - **GLOBAL.APP.desktop**: A reference to the desktop object
  - **GLOBAL.APP.SM**: A reference to the state management object responsible for saving, loading, managing active state, creating and loading user interface forms related to the state management.
  - **GLOBAL.APP.CF**: A reference to the object providing common functions that can be used by applications.

- **GLOBAL.BASE_URL**: Base URL that has to be used when requesting a service from the server.

- **GLOBAL.EXTJS_VERSION**: The version of the ExtJS library
• **GLOBAL.MOUSE_X**: The X coordinate of the mouse cursor relative to the top left corner of the presentation area of the browser.

• **GLOBAL.MOUSE_Y**: The Y coordinate of the mouse cursor relative to the top left corner of the presentation area of the browser.

• **GLOBAL.IS_IE**: An indicator whether the browser embedding the system is Internet Explorer or not.

• **GLOBAL.USER_CREDENTIALS**: A reference to an object containing the user credentials.

• **GLOBAL.STATE_MANAGEMENT_ENABLED**: An indicator whether the state management is available or not.

### Useful web components

When building the client side, you can use some additional components that are not part of the standard ExtJS set of components. These components were especially designed for the framework and the applications and can be found in `<Module name folder such as DIRAC, LHCbDIRAC, WebAppDIRAC>/WebApp/static/core/js/utils`:

• **DiracBoxSelect**: This component looks like the standard combo-box component, but provides more functionality. Main features: supporting of multchecking, searching through the options, and making negation of the selection. You can see an example of this component within the left panel of the JobMonitor application.

• **DiracFileLoad**: Whenever you want to load an extra JavaScript file or CSS file, but also you want to define a callback upon successful loading of the file, this is the right component for doing this.

• **DiracToolButton**: This component represents a small squared button providing possibility to define menu. This button is suitable for buttons that should take small space in cases such as headers of others components. You can see an example of this component at the header of left panel of the JobMonitor.

### Making MyApp application

The application we named **MyApp** is going to present some simple functionality. It is going to contain two visual parts: one with textarea and two buttons, and another part showing grid with some data generated on the server. When first button gets clicked, the value of the textarea is sent to the server and brought back to the client. When the second button gets clicked an information for a service called by the server is shown in the textarea.

1. First we are going to create the SS side of the MyApp. Go to the `[root]/handler` and create a file named `MyAppHandler.py`. This file will define the class whose instances will serve the MyApp client. The class will provide two services:

   - **web_getData**: this method will provide random data for the grid
   - **web_echoValue**: this method will return the same value that was sent together with the user request
   - **web_getServiceInfo**: this method will return some information about some service called from the server side. The information returned by the service is sent back to the client and shown in a textarea.

The code:

```python
from WebAppDIRAC.Lib.WebHandler import WebHandler
from DIRAC.Core.DISET.RPCClient import RPCClient
import random

class MyAppHandler(WebHandler):
  """
  The main class inherits from WebHandler
  """
```

```python
```
AUTH_PROPS is constant containing (a list of) properties the client requesting a service has to have in order to use this class.

AUTH_PROPS = "authenticated"

Entry-point method for data returned to the grid

def web_getData(self):
    data = self.__generateRandomData()
    self.write({"result": data})

Entry-point method to echo a value sent by the client

def web_echoValue(self):
    value = self.request.arguments["value"][0]
    self.write({"value": value})

Entry-point method to get service information. This method presents how to asynchronously support the clients requests on the server side.

@asyncGen
def web_getServiceInfo(self):
    RPC = RPCClient("WorkloadManagement/JobMonitoring")
    result = yield self.threadTask(RPC.ping)
    self.finish({"info": str(result["Value"])})

Private method to generate random data. This method cannot be called directly by the client i.e. it is not an entry point

def __generateRandomData(self):
    data = []
    for n in range(50):
        data.append({"value": random.randrange(1,100)})
    return data

2. Now we have to create the folder structure for the CS. The main folder of the MyApp application have to be located in a namespace folder. Let name that namespace folder DIRAC and place it in the [root]/static/ folder.

- WebApp
- handler
- MyAppHandler.py (already created in step 1)
- static
  - DIRAC
    - MyApp
Next, the folder **MyApp** should be created in the DIRAC folder together with four new subfolders, as mentioned in the explanation before: build, classes, css, and images folder.

3. After we finished creating the folder structure, we have to create some mandatory files as explained before. In the [root]/static/DIRAC/MyApp/classes create the file MyApp.js file. Similarly, create the file MyApp.css in the [root]/static/DIRAC/MyApp/css folder.

4. Open the MyApp.js. Here we have to define the main class representing the client side of the application. First we are going to code the frame of the class:

   ```javascript
   Ext.define('DIRAC.MyApp.classes.MyApp', {
      extend: 'Ext.dirac.core.Module',
      requires: [],
      initComponent: function() {},
      buildUI: function() {}
   });
   ```

   As explained before, first we have to implement the **initComponent** and the **buildUI** methods:

   ```javascript
   initComponent : function() {

      var me = this;

      //setting the title of the application
      me.launcher.title = "My First Application";
      //setting the maximized state
      me.launcher.maximized = false;

      //since the maximized state is set to false, we have to set the width and height of the window
      me.launcher.width = 500;
      me.launcher.height = 500;

      //setting the starting position of window, loading the application
      me.launcher.x = 0;
      me.launcher.y = 0;

      //setting the main layout of this application. In this case that is the border layout
      Ext.apply(me, {
         layout : 'border',
         bodyBorder : false,
         defaults : {
            collapsible : true,
            split : true
         }
      });

      //at the end we call the initComponent of the parent ExtJS class
      me.callParent(arguments);
   },

   buildUI : function() {
   ```
var me = this;

/*
Creating the left panel.
Pay attention that the region config property is set up to west
which means that the panel will take the
left side of the available area.
*/
me.leftPanel = new Ext.create('Ext.panel.Panel', {
    title : 'Text area',
    region : 'west',
    width : 250,
    minWidth : 230,
    maxWidth : 350,
    bodyPadding : 5,
    autoScroll : true,
    layout : {
        type : 'vbox',
        align : 'stretch',
        pack : 'start'
    }
});

//creating the textarea
me.textArea = new Ext.create('Ext.form.field.TextArea', {
    fieldLabel : "Value",
    labelAlign : "top",
    flex : 1
});

//embedding the textarea into the left panel
me.leftPanel.add(me.textArea);

/*
Creating the docked menu with a button
to send the value from the textarea to the server
*/

//creating a button with a click handler
me.btnValue = new Ext.Button({
    text : 'Echo the value',
    margin : 1,
    handler : function() {
        Ext.Ajax.request({
            url : GLOBAL.BASE_URL + 'MyApp/echoValue',
            params : {
                value: me.textArea.getValue()
            },
            scope : me,
            success : function(response) {
                var me = this;
                var response = Ext.JSON.decode(response.responseText);
                alert("THE VALUE: " + response.value);
            }
        });
    }
});
// creating a button with a click handler
var me.btnRPC = new Ext.Button({
    text: 'Service info',
    margin: 1,
    handler: function() {
        Ext.Ajax.request({
            url: GLOBAL.BASE_URL + 'MyApp/getServiceInfo',
            params: {},
            scope: me,
            success: function(response) {
                var me = this;
                var response = Ext.JSON.decode(response.responseText);
                me.textArea.setValue(response.info);
            }
        });
    },
    scope: me
});

//creating the toolbar and embedding the button as an item
var oPanelToolbar = new Ext.toolbar.Toolbar({
    dock: 'bottom',
    layout: {
        pack: 'center'
    },
    items: [me.btnValue, me.btnRPC]
});

/*
Docking the toolbar at the bottom side of the left panel
*/
me.leftPanel.addDocked([oPanelToolbar]);

/*
Creating the store for the grid
This object stores the data.
*/
me.dataStore = new Ext.data.JsonStore({
    proxy: {
        type: 'ajax',
        url: GLOBAL.BASE_URL + 'MyApp/getData',
        reader: {
            type: 'json',
            root: 'result'
        }
    }
});
5. Throughout all the code, especially in the method buildUI, there are several components created in order to structure the user interface. Therefore, you have to append all the classes used within the `DIRAC.MyApp.classes.MyApp` requires definition. In our case the list of requires would look like:

```javascript
```

6. In order to have the application within the list of applications, you have to open the `web.cfg` file located into the root. There you have to add new registration line within the `Schema/Applications` section:

```plaintext
WebApp
{
    DevelopMode = True
    Schema
    {
        Applications
        {
            Job Monitor = DIRAC.JobMonitor
            Accounting = DIRAC.AccountingPlot
            Configuration Manager = DIRAC.ConfigurationManager
            File Catalog = DIRAC.FileCatalog
        }
    }
}
```
7. Now you can test the application. Before testing the application restart the server in order to enable the application within the main menu.

**Debugging an application**

In order to debug an application, debugging tools are needed to be used. In Firefox you can install and use the Firebug toolset which can be also used in Chrome but in a light version.

In Chrome you can use developer tools.

DIRAC web framework provides two modes of working regarding the CS. One is the development mode, which means that the JavaScripts are loaded as are, so that they can be easily debugged. The other mode is the production mode where JavaScripts are minimized and compiled before loaded. Those JavaScripts are lighter in memory but almost useless regarding the debugging process.

In order to set up the production mode, you have to set the `DevelopMode` parameter into the web.cfg file as shown as follows (by default this parameter is set to `True`):

```
WebApp {
    DevelopMode = False

    Schema {
        Applications {
            Job Monitor = DIRAC.JobMonitor
            Accounting = DIRAC.AccountingPlot
            Configuration Manager = DIRAC.ConfigurationManager
            File Catalog = DIRAC.FileCatalog
            Notepad = DIRAC.Notepad
            My First Application = DIRAC.MyApp
        }
        TestLink = link|http://google.com
    }
}
```

Before you can use the compiled version of the JavaScript files, you have to compile them first. For this reason you have to execute the python script `dirac-webapp-compile`. In order to run the script, you have to download and install a tool called Sencha Cmd (http://www.sencha.com/products/sencha-cmd/download). You can also refer to http://docs.sencha.com/extjs/4.2.1/#!/guide/command and read the System Setup section for detailed installation.

**Inheritance of applications**

The inheritance of an application is done in both SS and CS. In this case let suppose that we want to inherit the MyApp application. Let name this new application MyNewApp.

The procedure for creating a new application is the same one as explained in the previous section.
When creating the python file, the Python class, namely `DIRAC.MyNewApp.classes.MyNewApp`, has to inherit from `DIRAC.MyApp.classes.MyApp`. Be aware that before you can inherit, firstly you have to import the parent file. The code would look like as follows:

```python
from WebAppDIRAC.WebApp.handler.MyAppHandler import MyAppHandler
import random

class MyNewAppHandler(MyAppHandler):
    AUTH_PROPS = "authenticated"
```

When creating the main JavaScript file, in this case named `MyNewApp.js`, there are two parts that differ from the obvious development. First of all, the ExtJS class to be developed, namely `DIRAC.MyNewApp.classes.MyNewApp` has to extend `DIRAC.MyApp.classes.MyApp` instead of `Ext.dirac.core.Module`.

Next, when defining the buildUI method, first of all the parent buildUI has to be called before any other changes take place.

### User credentials and user properties

For some functionalities of the applications you have to distinguish between various kind of users. For example, in the configuration manager, the whole configuration can be browsed, but also it can be managed and edited. The management functionality shall be allowed only for the users that have the property of `CSAdministrator`.

On the client side, these properties of a user can be accessed via the `GLOBAL.USER_CREDENTIALS.properties` variable. On the server side the list of user properties is contained in `self.getSessionData().properties`. So in the case of configuration manager, at the client side we use the following code:

```javascript
if ("properties" in GLOBAL.USER_CREDENTIALS) && (Ext.Array.indexOf(GLOBAL.USER_CREDENTIALS.properties, "CSAdministrator") != -1) { ...
```

At the server side of configuration manager we did a method to check whether an user is a configuration manager or not:

```python
def __authorizeAction(self):
    data = SessionData().getData()
    isAuth = False
    if "properties" in data["user"]:  
        if "CSAdministrator" in data["user"]["properties"]: 
            isAuth = True
    return isAuth
```

Be aware that sometimes `properties` list is not part of the credentials object so it can be checked first for its existence before it can be used.

### Using predefined widgets

Create your first example

We already prepared a simple example using predefined widgets (You can found more information https://hostname/DIRAC/static/doc/index.html and you can have a look the code in github: https://github.com/DIRACGrid/WebAppDIRAC/tree/integration/WebApp/static/DIRAC).

NOTE: Please make sure that your application will compile. You have to use:

dirac-webapp-compile
# Documentation sources

<table>
<thead>
<tr>
<th>User Guide</th>
<th>Developer Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Everything users need to know</td>
<td>Adding new functionality to DIRAC</td>
</tr>
<tr>
<td>Administrator Guide</td>
<td>CodeReference/index</td>
</tr>
<tr>
<td>Administration of the DIRAC service</td>
<td>Code reference</td>
</tr>
</tbody>
</table>

This documentation is also available in PDF version
Indices and tables

- genindex
- search