DIRAC Documentation

Release v6r21_tuto_tsdms

DIRAC Project.

07:43 24/04/2019 Coordinated Universal Time

Contents

The DIRAC project is a complete Grid solution for one, or more than one community of users that need to exploit distributed, heterogeneous resources.

DIRAC forms a layer between a community and various compute resources to allow optimized, transparent and reliable usage. The types of resources that DIRAC can handle include:

- *Computing* Resources, including Grids, Clouds, and Batch systems
- *Storage* Resources
- *Catalog* Resources

Many communities use DIRAC, the oldest and most experienced being the LHCb collaboration. Other communities include, but are not limited to, Belle2, ILC, and CTA

DIRAC source code is open source (GPLv3), written largely in python 2.7, and hosted on github.

DIRAC provides code for:

- client installations
- server installations
- pilots installations

A more detailed description of the DIRAC system can be found at this location or in this presentation

CHAPTER 1

User Guide

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 few steps. You can do these steps as any user, there's no need to be root.

A DIRAC client installation (and a server too) is fully in user space, and in fact, it's all in one directory. Which means that on a same machine you can have several client (or server even) installed.

If you want to create a shared client installation, you can do it by simply giving (UNIX) access to the directory where the client is installed.

Install script

Download the *dirac-install* script from:

```
wget -np -O dirac-install https://github.com/DIRACGrid/DIRAC/raw/integration/Core/
↪scripts/dirac-install.py --no-check-certificate
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, and, the version of the "lcgBundle" (with "-g" option) that you want to use:

```
dirac-install -r v6r20p14 -g v14r2
```

The example above assumes that you need version v6r20p14, and that with it you are installing lcgBundle version v14r2.

An "lcgBundle" is simply a tarball containing a number of statically-compiled libraries that are used for interacting with grid environments (e.g. GFAL2, or ARC, or Condor). The libraries in a "lcgBundle" are not maintained within DIRAC, but DIRAC may use them. The produced lcgBundles can be found in this server.

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

```
drwxr-xr-x. 20 dirac dirac  4096 Jul 25 15:13 DIRAC
drwxr-xr-x.  6 dirac dirac  4096 Jul 21 16:27 Linux_x86_64_glibc-2.12
-rw-r--r--.  1 dirac dirac  2153 Jul 25 15:13 bashrc
-rw-r--r--.  1 dirac dirac  2234 Jul 25 15:13 cshrc
-rw-r--r--.  1 dirac dirac  42   Jul 25 15:13 defaults-DIRAC.cfg
-rwxr-xr-x.  1 dirac dirac  61754 Jul 25 15:11 dirac-install
drwxr-xr-x.  2 dirac dirac  12288 Jul 25 15:13 scripts
```

Instead of the *Linux_x86_64_glibc-2.12* 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* and *cshrc* script is there to easily set up the environment for your DIRAC installation, so assuming you are using bash:

```
source bashrc
```

Think of adding the above line to your login scripts.

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 *dirac-configure* command.

The dirac-configure command can take as input a cfg file whose content can be, for example, the following:

```
LocalInstallation
{
  ConfigurationServer = dips://lbcertifdirac6.cern.ch:9135/Configuration/Server
  Setup = Dirac-Certification
}
```

where the Setup option is specifying the DIRAC Setup name within which the client will be working. The ConfigurationServer option is used 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.

In any case, save a "install.cfg" file with the content desired.

At this point, in order to run the *dirac-configure* command, you need either a user proxy, or a host certificate. They are needed because dirac-configure will take care of updating the local configuration, but also because it will download the CAs used for connecting to DIRAC services (this option may be overridden if necessary).

Using a user proxy

If you want to use a user proxy, we assume that you already have a user certificate, so in this case create a directory *.globus* in your home directory and copy the certificate files (public and pruvate keys in pem format) to this directory:

```
$ mkdir ~/.globus
$ cp <<certificate files>> ~/.globus/
```

At this point you need a proxy, but you still have not configured DIRAC. So, you should do:

```
$ dirac-proxy-init
```

This will probably give you an error, but will still create a local proxy file anyway. You can see which file is your proxy certificate using the *dirac-proxy-info* command.

It id then possible to issue the *dirac-configure* command:

```
dirac-configure install.cfg
```

Using a host certificate

If you have a host certificate for the machine where the client is being installed, and if this host certificate DN is registered in the Configuration Server of the DIRAC server machine, then such host certificate can be used instead of the user proxy certificate, with the following:

```
dirac-configure --UseServerCertificate -o /DIRAC/Security/CertFile=<directory>/
↪hostcert.pem -o /DIRAC/Security/KeyFile=<directory>/hostkey.pem install.cfg
```

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

Users authenticate with DIRAC services, and therefore with the Grid services that DIRAC expose via "proxies", which you can regard as a product of personal certificates.

There are two major differences between certificates and proxies:

- certificates are signed by a CA, while proxies can be signed by a certificate and/or by another proxy
- proxies can have extra token embedded (like macaroon of Google)

There are two types of proxies in DIRAC. The *legacy* proxies, and the *RFC* proxies. The legacy proxies are really specific to the Grid, while the RFC follow an RFC standard (https://www.ietf.org/rfc/rfc3820.txt). Unless you are using a fairly old DIRAC version, the *RFC* proxies are the default type of proxies that will be created by the commands that follow.

Proxies are much less spread than certificates. It might come in a few years, because they are rumors than commercial clouds are interested in that kind of solution for short lived services. But as of now, it is not very spread. They are anyway a de-facto standard for grid services since many years now. Everything related to RFC proxies is already in standard *openssl*.

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:

```
dirac-proxy-init (or simply "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:

```
dirac-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:

```
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.

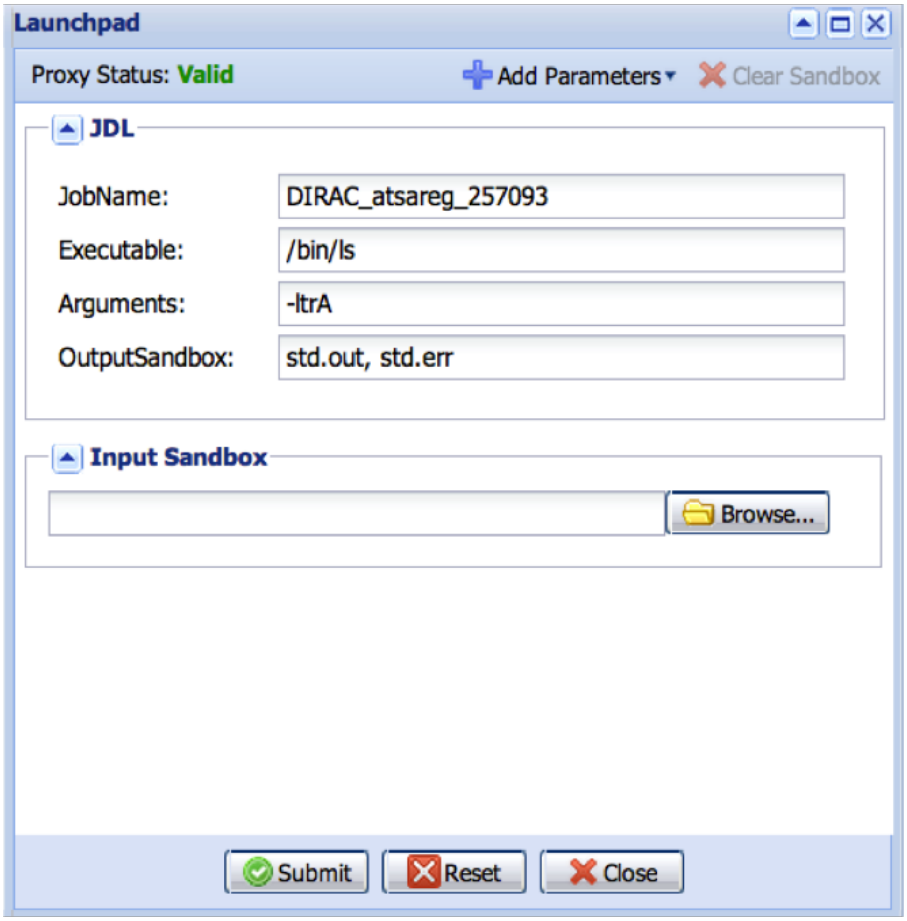
Once the job in its final Status (Done or Failed), you can retrieve the job outputs by:

```
>  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 you 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.:

```
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.submitJob(j)
print 'Submission Result: ',jobID
```

In this example, the job has tree 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:

```
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:

```
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		
Attribute Name	**Description**	**Example**
Executable	Name of the executable file	Executable = "/bin/ls";
Arguments	String of arguments for the job executable	Arguments = "-ltr";
StdError	Name of the file to get the standard error stream of the user application	StdError = "std.err";
StdOutput	Name of the file to get the standard output stream of the user application	StdOutput = "std.out";
InputSandbox	A list of input sandbox files	InputSandbox = {"jobScript.sh"};
OutputSandbox	A list of output sandbox files	OutputSandbox = {"std.err","std.out"};
Job Requirements These parameters are interpreted as job requirements		
Attribute Name	**Description**	**Example**
CPUTime	Max CPU time required by the job in HEP-SPEC06 seconds	CPUTime = 18000;
Site	Job destination site	Site = {"EGI.CPPM.fr"};
BannedSites	Sites where the job must not go	BannedSites = {"EGI.LAPP.fr","EGI.M3PEC.fr"};
Platform	Target Operating System	Platform = "Linux_x86_64_glibc-2.5";
Data Describing job data		
Attribute Name	**Description**	**Example**
InputData	Job input data files	InputData = {"/dirac/user/a/atsareg/data1", "/dirac/user/a/atsareg/data1"};
OutputData	Job output data files	OutputData = {"output1","output2"};
OutputPath	The output data path in the File Catalog	OutputPath = {"/myjobs/output"};
OutputSE	The output data Storage Element	OutputSE = {"DIRAC-USER"};
Parametric Jobs Bulk submission parameters		
Attribute Name	**Description**	**Example**
Parameters	Number of parameters or a list of values	Parameters = 10;
ParameterStart	Value of the first parameter	ParameterStart = 0.;
ParameterStep	Parameter increment	ParameterStep = 0.1; (default 0.)
ParameterFactor	Parameter multiplier	ParameterFactor = 1.1; (default 1.)

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 Storaget 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:

```
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

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

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

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

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:

Status	Comment
Waiting	Job is accepted for DIRAC WMS
Scheduled	Job is assigned to a Site
Running	Job has started running in the CE
Done	Job finished successfully
Deleted	Job deleted by the user
Killed	Job killed by the user

Minor Status

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

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

Application Status

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

Application Status	Comment
Failed Input Sandbox Download	Job failed to download Input Sandbox.
Unknown	Job failed by a unknown reason.

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
- Site Descending
- Status Ascending
- Status Descending
- Minor Status Ascending
- Minor Status Descending

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.

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

Minor Status

Complement Job Status.

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

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.

GetLogFile

GetPendingRequest

GetStagerReport

GetSandboxFile

Actions

Actions that the user can perform over their jobs are showed below:

Action	Comment
Reset	Restart the Job
Kill	Kill the Job selected
Delete	Delete the job

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 than 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

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.

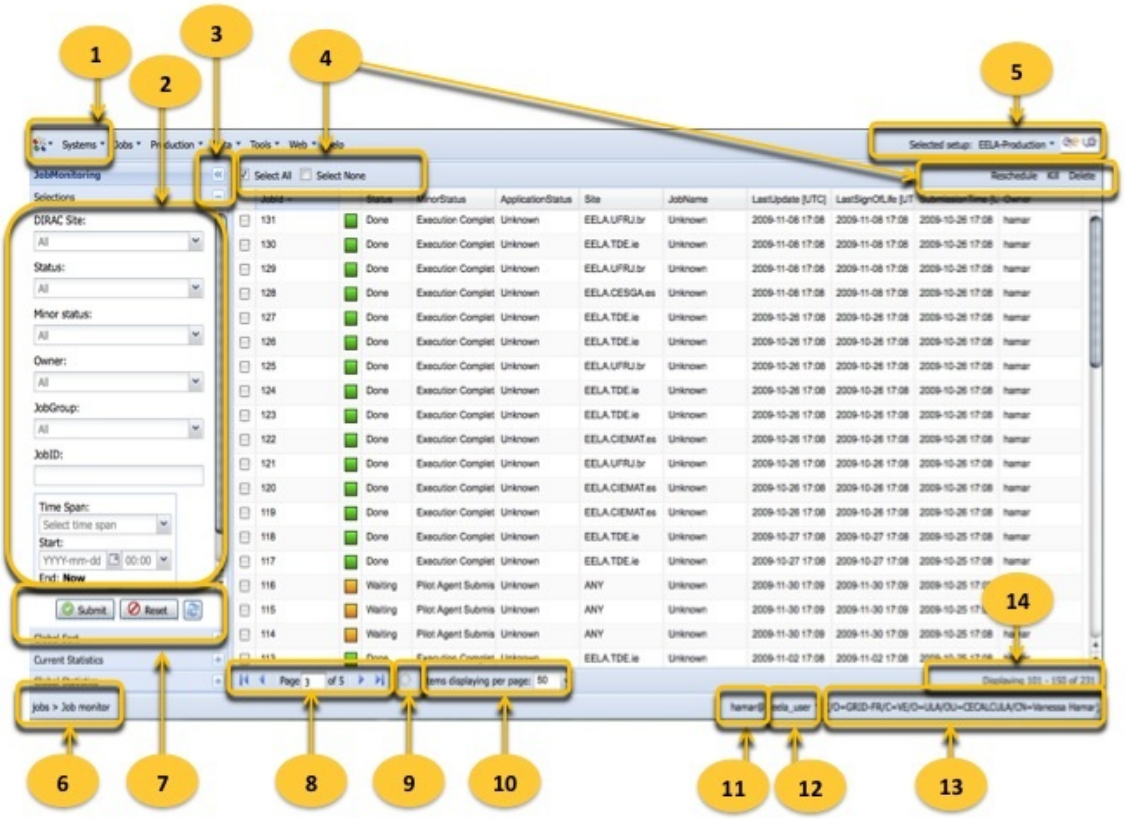


Fig. 1: DIRAC Web Portal

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.

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

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:

Status Comment

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

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 pilot 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.

Status	Comment
Bad	Site effectiveness less than 25% of pilot jobs executed successfully
Poor	Site effectiveness less than 60% of pilot jobs executed successfully
Fair	Site effectiveness less than 85% of pilot jobs executed successfully
Good	Site effectiveness more than 85% of pilot jobs executed successfully

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

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:

Status	Comments
New	New Production
Active	Active Production
Stopped	A production can be stopped by
Validating Input	Inputs of production are being checked
Validating Output	Outputs of productions are being checked
Waiting Integrity	**The system is waiting for integrity results??**
Remove Files	
Removed Files	
Completed	Production completely processed
Archived	**Output of production are archived into**
Cleaning	Production is being cleaned

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.

ID

DIRAC Production ID.

Status

Production Status.

Agent Type

How the agent was submit: Automatic or Manual

Type

Production Type, by example: MCSimulation.

Group

DIRAC group of the user than submit the production.

Name

Production name.

Files

Number of files required to run the production.

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 summited 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:

Action	Comment
Start	Start the production
Stop	Stop the production
Flush	Flush the production
Clean	Clean

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

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 whit the TargetDN over who the operation is performed.

Filters

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

The menu show options are:

Filter by action

Depending of the value of the log than was clicked will be created the filter.

Filter by issuer DN

Depending of the value of the log than was clicked will be created the filter.

Filter by target DN

Depending of the value of the log than was clicked will be created the filter.

Filter by target group

Depending of the value of the log than was clicked will be created the filter.

At the bottom field appears the following items:

Page Manager

Allow the user 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 than present option 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 clear 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

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:

Status	Comment
Active	
Done	
Failed	

Global Statistics

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

Status	Comment
Active	
Done	
Failed	

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

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.

Commiter

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

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.

Status

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:

Status

Status	Comment
Bad	Site effectiveness less than 25% of pilot jobs executed successfully
Poor	Site effectiveness less than 60% of pilot jobs executed successfully
Fair	Site effectiveness less than 85% of pilot jobs executed successfully
Good	Site effectiveness more than 85% of pilot jobs executed successfully

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

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.

1.3.2 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&***

  1. https://: protocol
  2. lhcb-portal-dirac.cern.ch/DIRAC/: host.
  3. s:LHCb-Production: DIRAC setup.
  4. g:lhcb_prmgr : role

5. view=desktop : 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 state. It can be 0 or 1.
8. The state is a desktop: url_state=1|AllPlots
9. The state is an application: url_state=0|LHCbDIRAC.LHCbJobMonitor.classes.LHCbJobMonitor:statename:0:0:14
   1,-1,-1,-1

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* link.

Tab theme

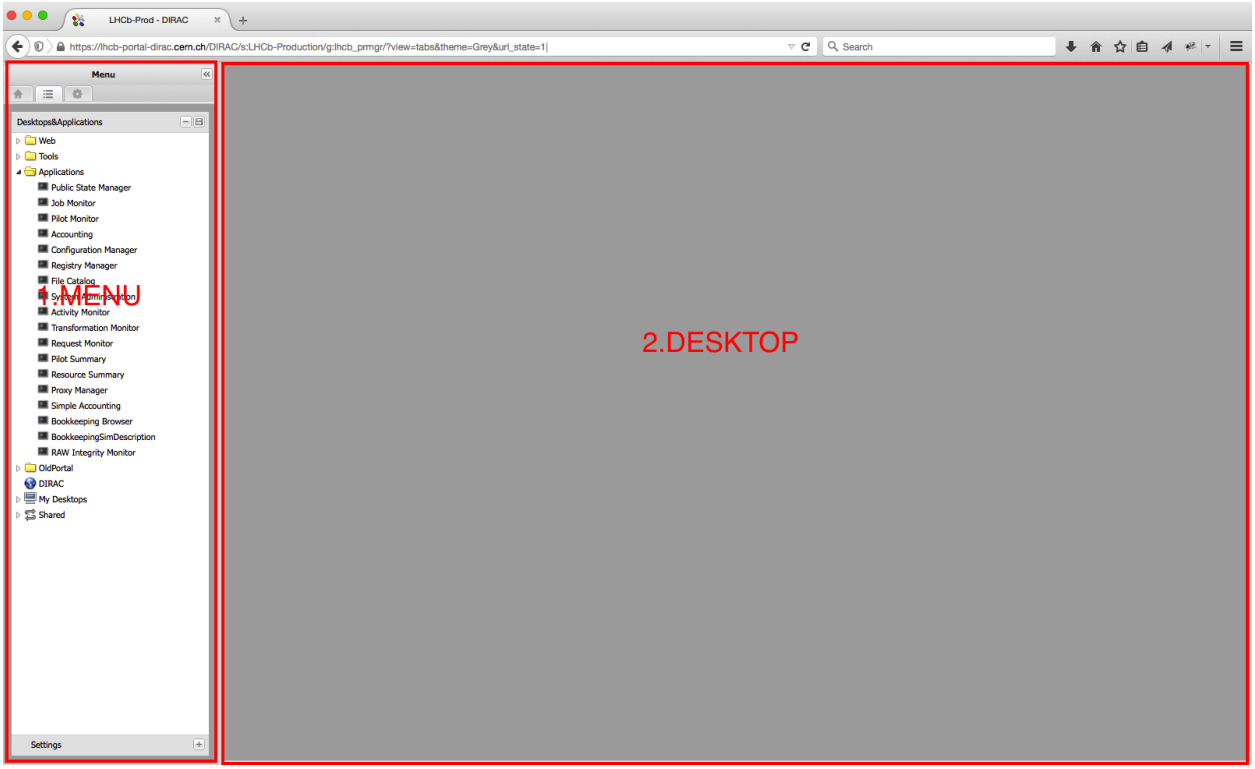
In this section a detailed description of the Tab theme is presented:

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

Main panel

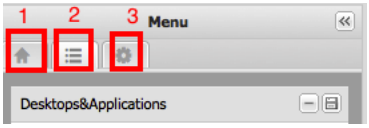
The main panel consists of two widget:

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.

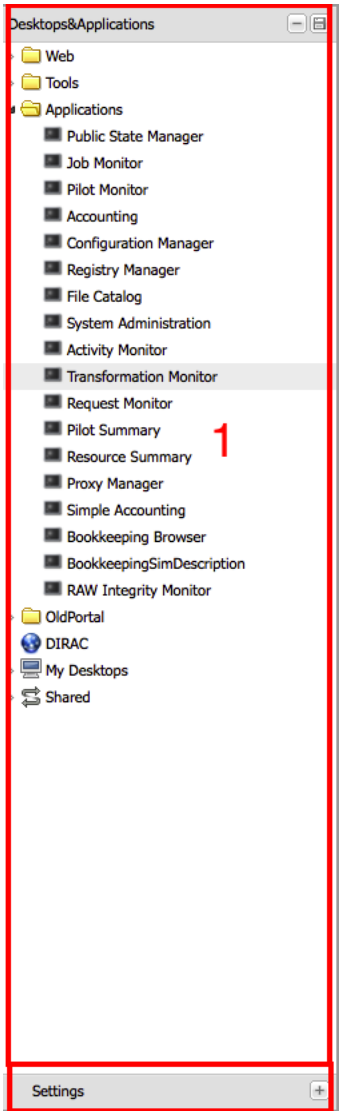
Desktop

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

Menu structure

Menu consists of two widgets:

1. Desktops&Applications
2. Settings



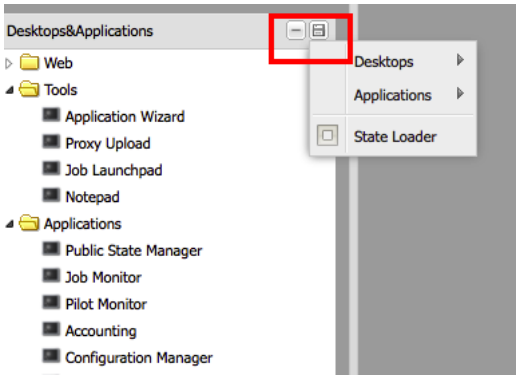
Desktop&Applications

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 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

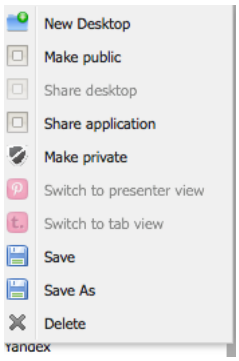
The Application menu item contains:

- 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.

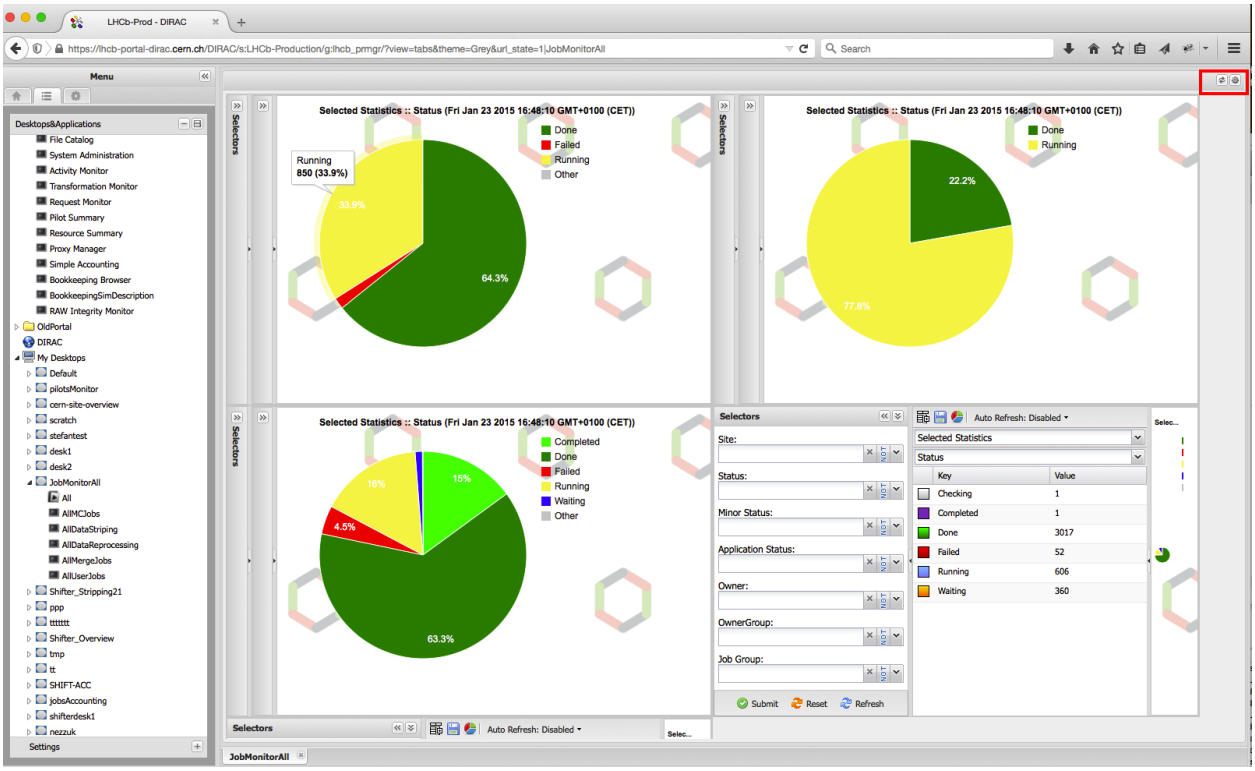


You have few additional menu items:

- 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.

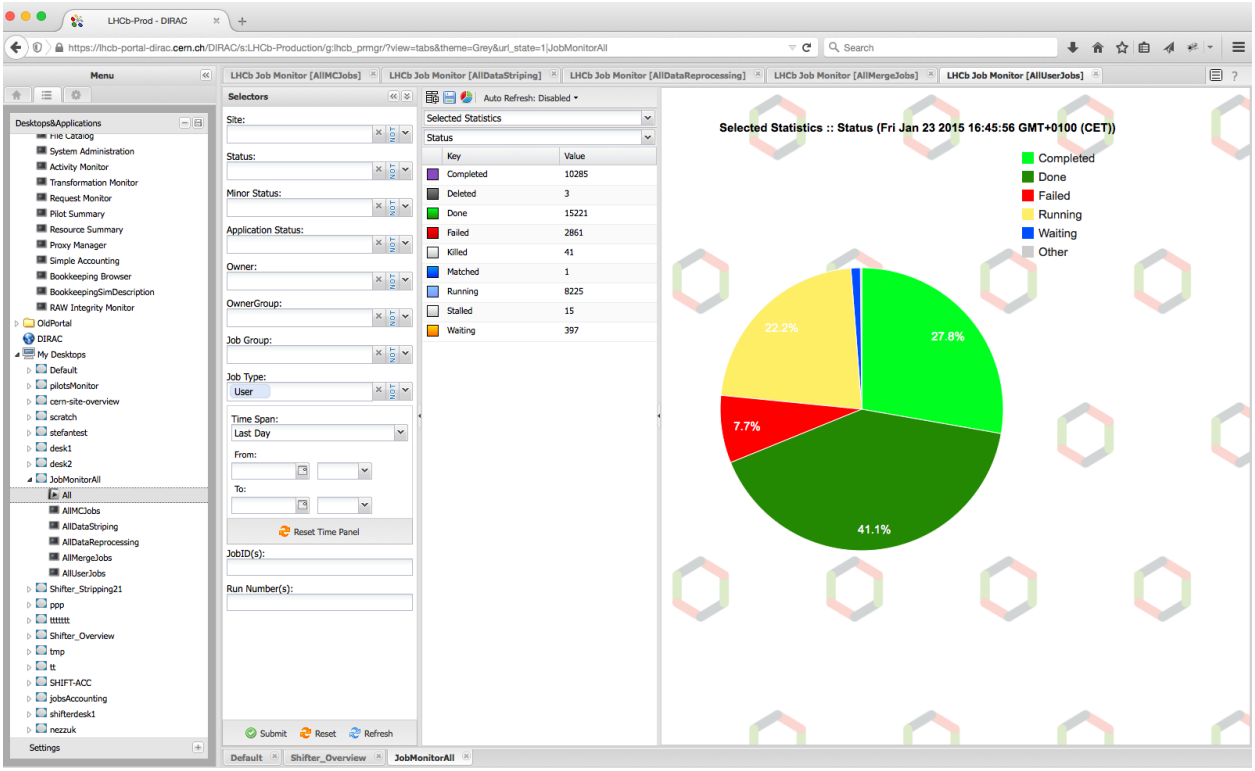
Presenter view

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).



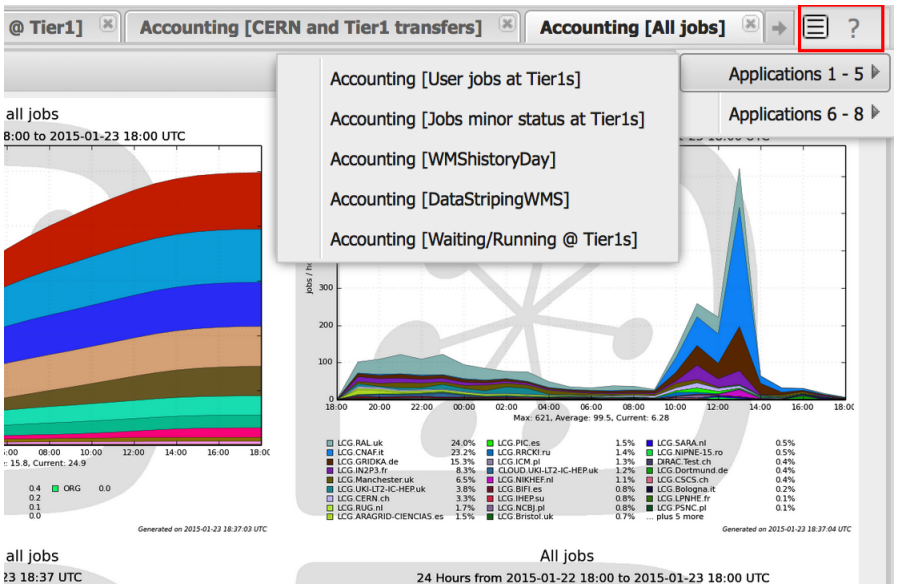
Tab view

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

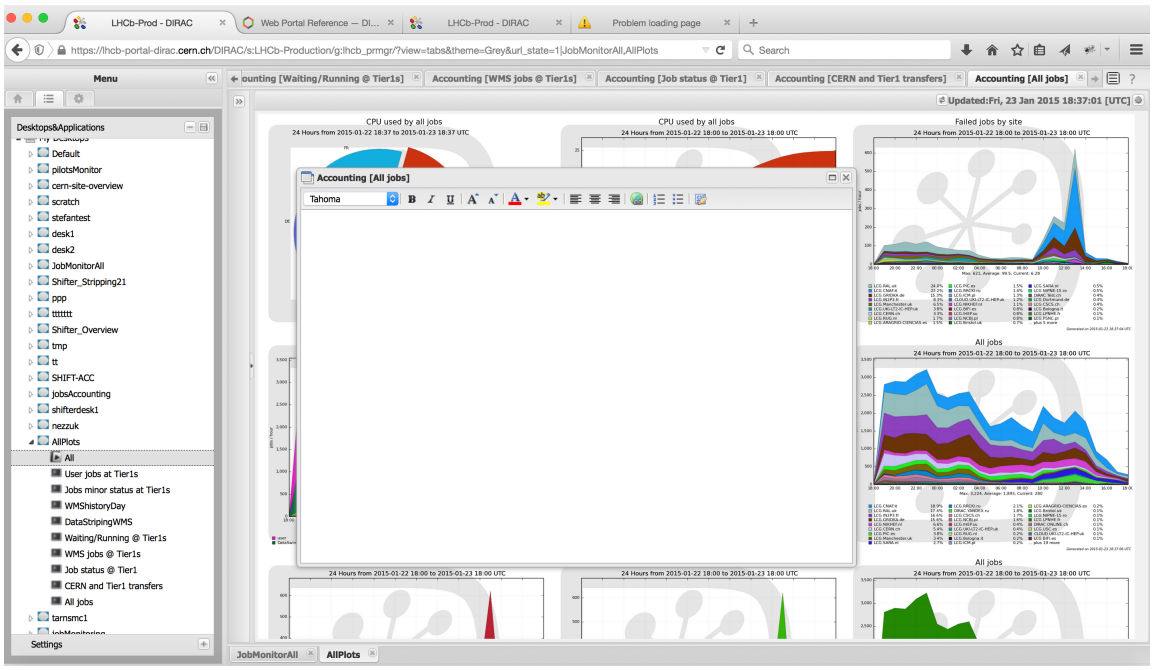


In the right corner of the Tab theme you can see two icons.

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.



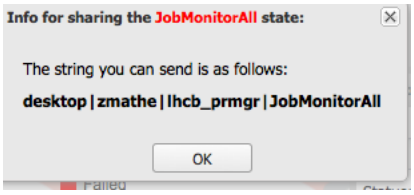
Share application and desktop

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

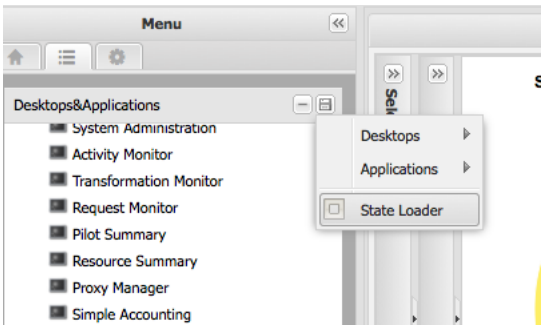
You have to do the following steps to share an application/desktop:

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

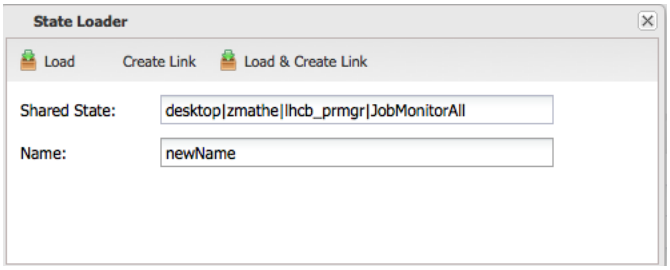


Load a shared application or desktop

You have to use the *State Loader* menu item:



The State Loader widget is the following:

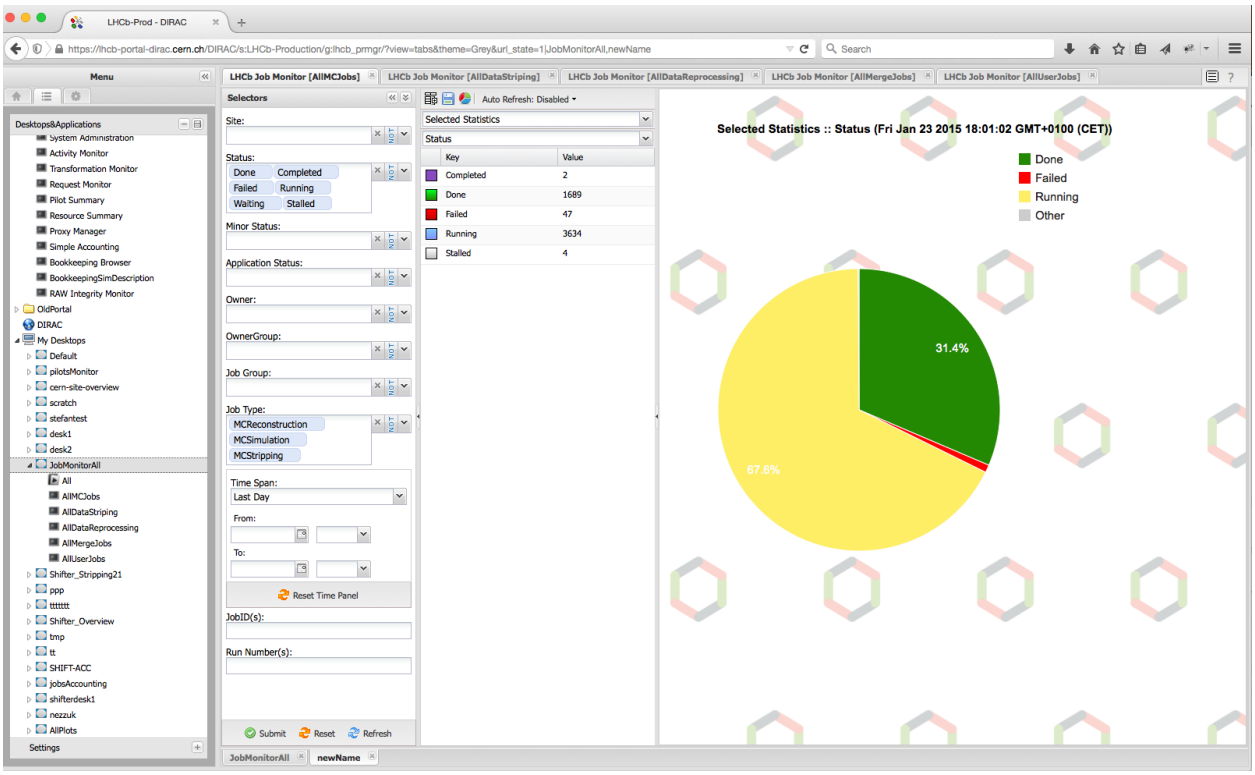


You have to provide the Shared State (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and a name (for example: newName). You have tree different way to load a shared state:

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

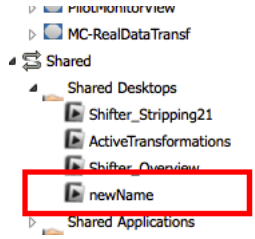
Load

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



Create Link

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

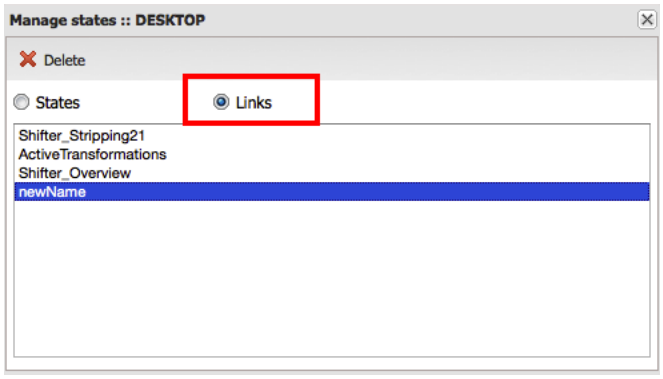


Load & Create Link

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

Delete shared applications/desktops

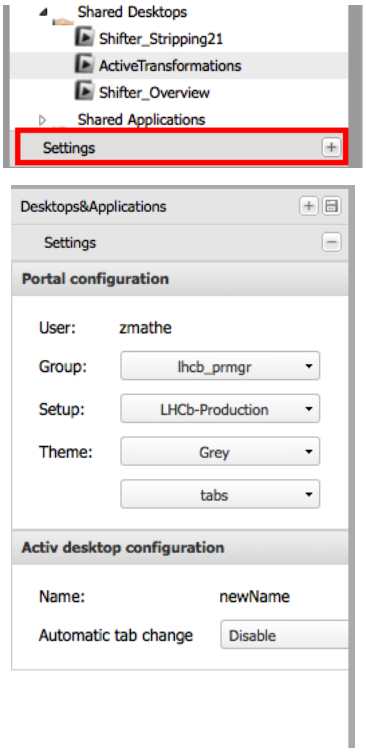
You have to click on the menu *Manage application and desktop* and then select application or desktop depending what you want to delete. For example: Let's delete the **newName** shared desktop.



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.

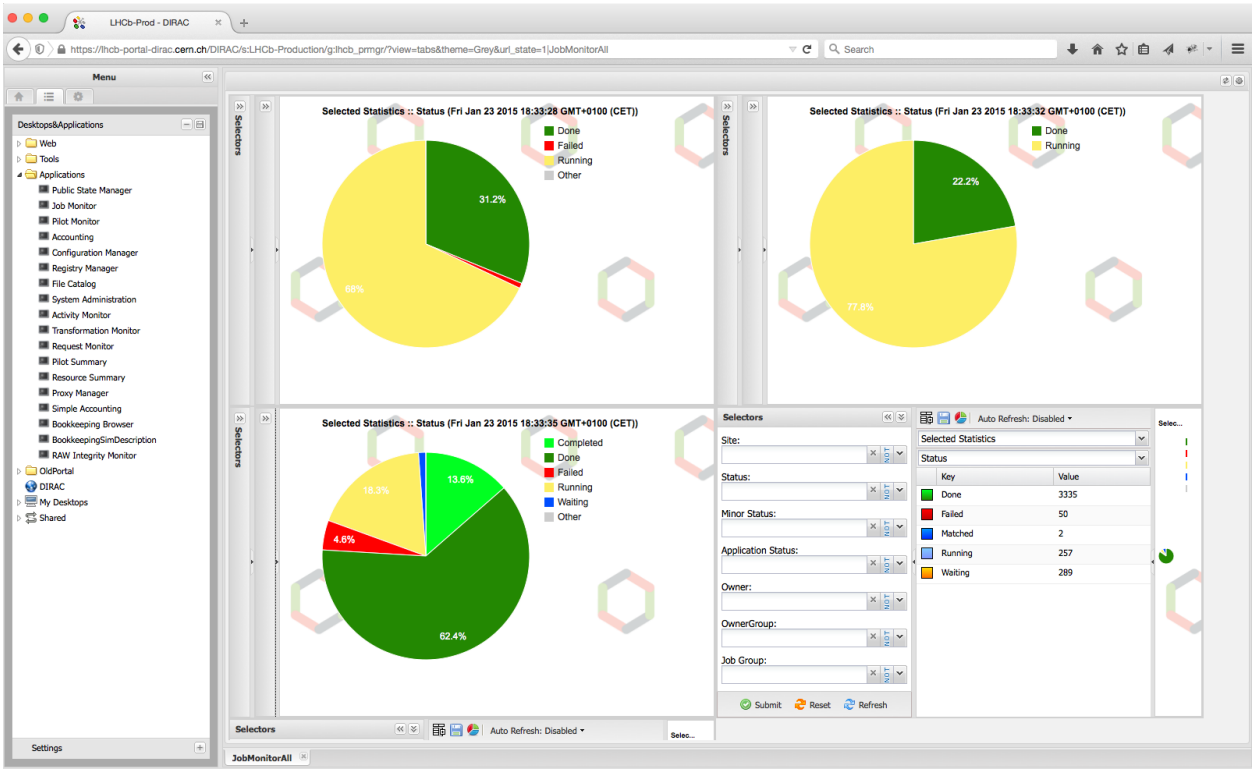
We have 3 look and feels:

1. Grey it is the default
2. Neptune

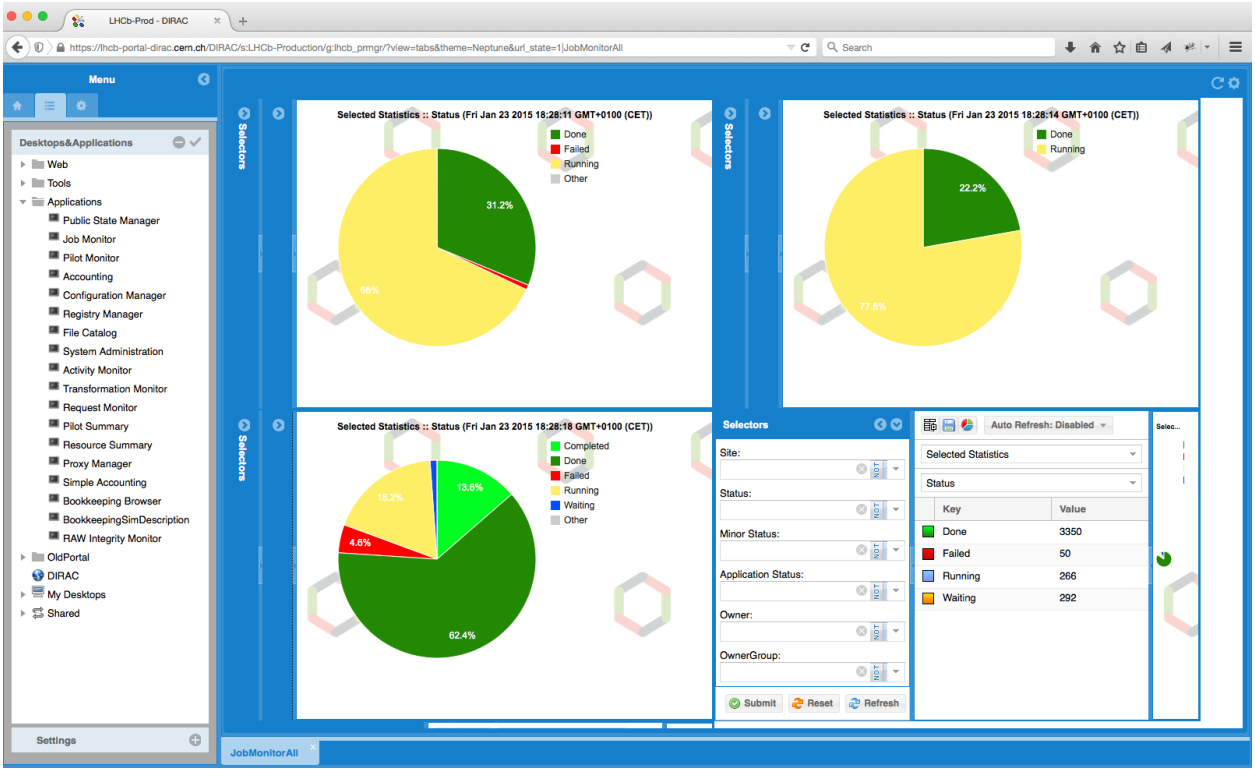
3. Classic

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.

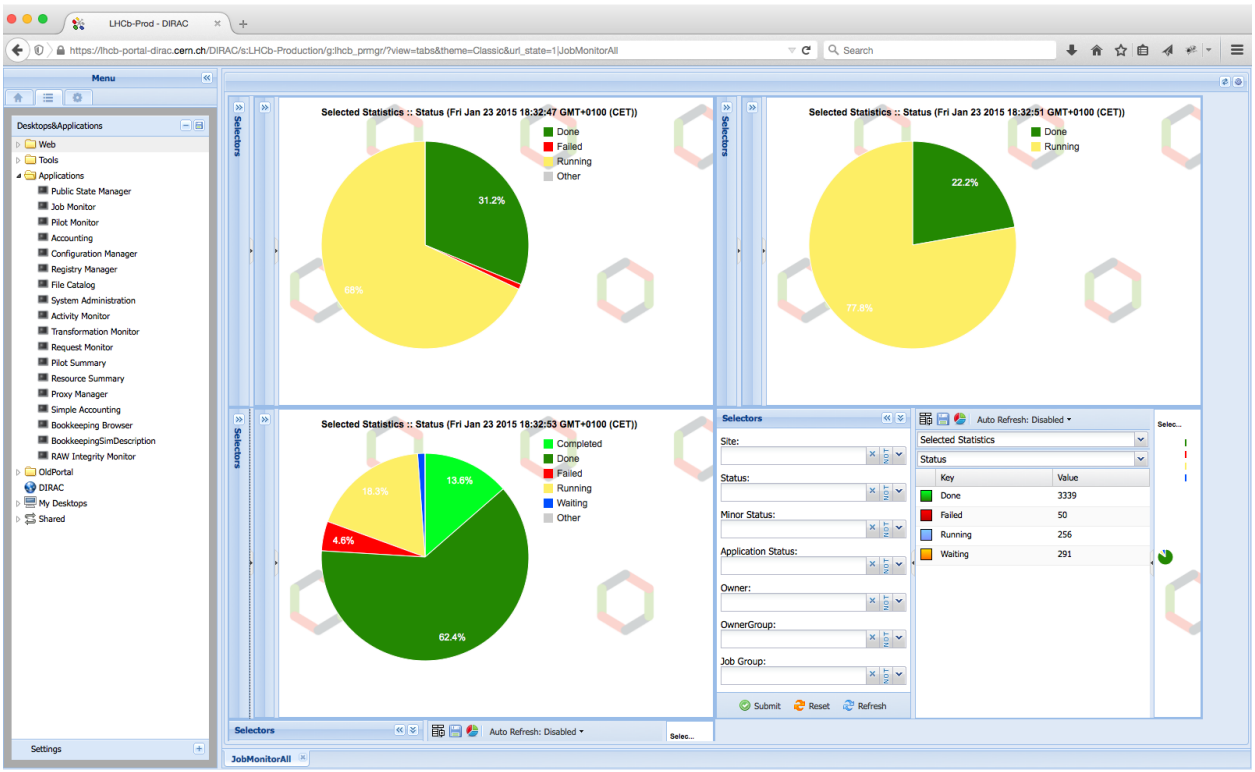
Grey



Neptune


LHCb-Prod - DIRAC
Menu
Desktops&Applications
Web
Tools
Applications
Public State Manager
Job Monitor
Pilot Monitor
Accounting
Configuration Manager
Registry Manager
File Catalog
System Administration
Activity Monitor
Transformation Monitor
Request Monitor
Pilot Summary
Resource Summary
Proxy Manager
Simple Accounting
Bookkeeping Browser
BookkeepingSimDescription
RAW Integrity Monitor
OldPortal
DIRAC
My Desktops
Shared
Settings
Selected Statistics :: Status (Fri Jan 23 2015 18:28:11 GMT+0100 (CET))
Done
Failed
Running
Other
31.2%
68%
Selected Statistics :: Status (Fri Jan 23 2015 18:28:14 GMT+0100 (CET))
Done
Running
22.2%
77.8%
Selected Statistics :: Status (Fri Jan 23 2015 18:28:18 GMT+0100 (CET))
Completed
Done
Failed
Running
Waiting
Other
13.6%
18.2%
4.6%
62.4%
Selectors
Site:
Status:
Minor Status:
Application Status:
Owner:
OwnerGroup:
Submit
Reset
Refresh
Auto Refresh: Disabled
Selected Statistics
Status
Key
Value
Done
3350
Failed
50
Running
266
Waiting
292
JobMonitorAll

Classic


LHCb-Prod - DIRAC
Menu
Desktops&Applications
Web
Tools
Applications
Public State Manager
Job Monitor
Pilot Monitor
Accounting
Configuration Manager
Registry Manager
File Catalog
System Administration
Activity Monitor
Transformation Monitor
Request Monitor
Pilot Summary
Resource Summary
Proxy Manager
Simple Accounting
Bookkeeping Browser
BookkeepingSimDescription
RAW Integrity Monitor
OldPortal
DIRAC
My Desktops
Shared
Settings
Selected Statistics :: Status (Fri Jan 23 2015 18:32:47 GMT+0100 (CET))
Done
Failed
Running
Other
31.2%
68%
Selected Statistics :: Status (Fri Jan 23 2015 18:32:51 GMT+0100 (CET))
Done
Running
22.2%
77.8%
Selected Statistics :: Status (Fri Jan 23 2015 18:32:53 GMT+0100 (CET))
Completed
Done
Failed
Running
Waiting
Other
13.6%
18.3%
4.6%
62.4%
Selectors
Site:
Status:
Minor Status:
Application Status:
Owner:
OwnerGroup:
Job Group:
Submit
Reset
Refresh
Auto Refresh: Disabled
Selected Statistics
Status
Key
Value
Done
3339
Failed
50
Running
256
Waiting
291
JobMonitorAll

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.

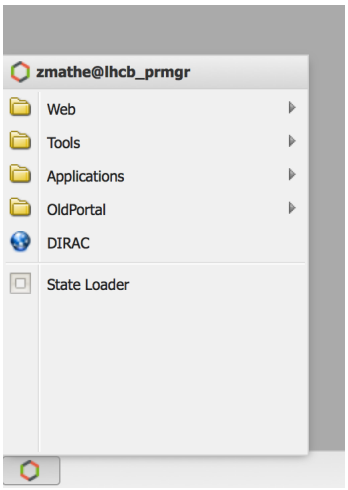
You can define the following:

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

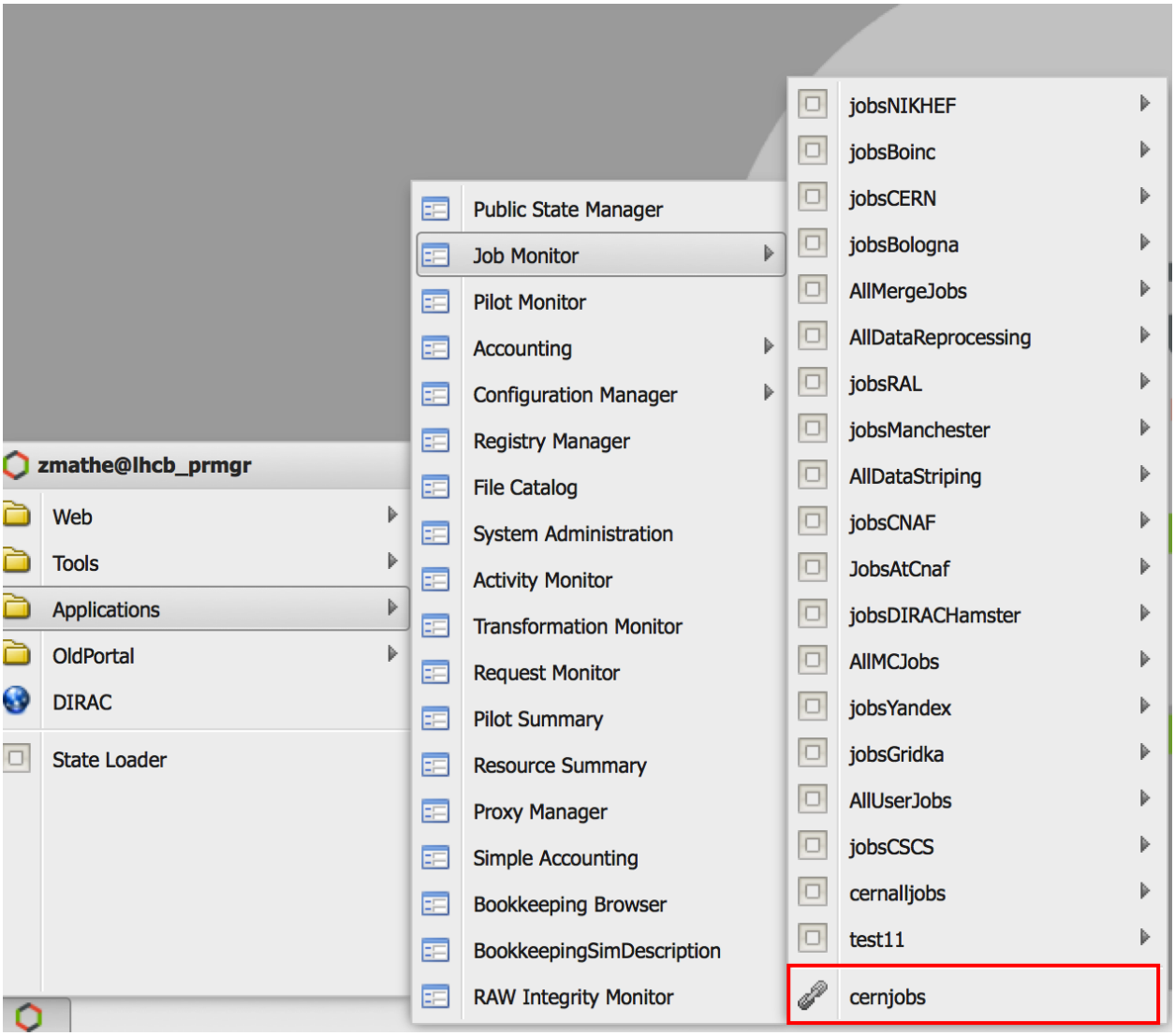
Menu structure

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.

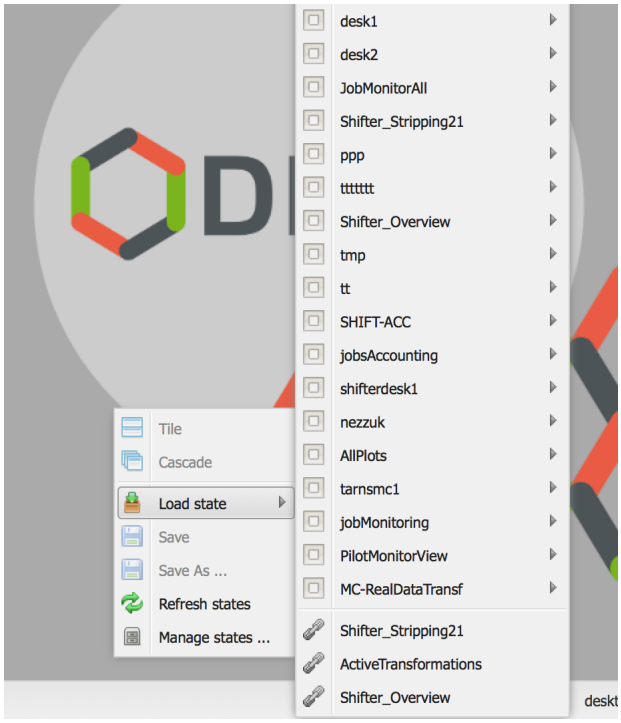


The states of the applications are available when you click on the application name.



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

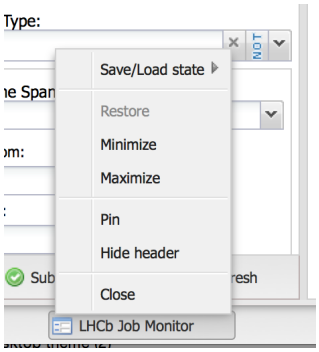
Applications

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.

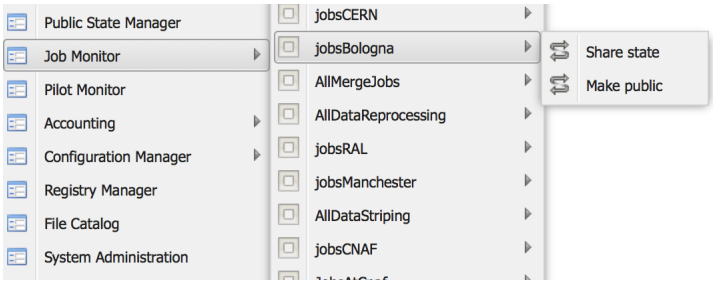
Desktops

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

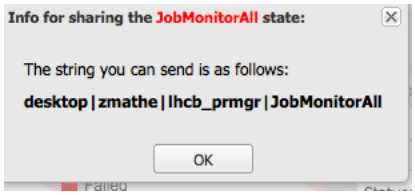
Share an application/desktop

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



You have to do:

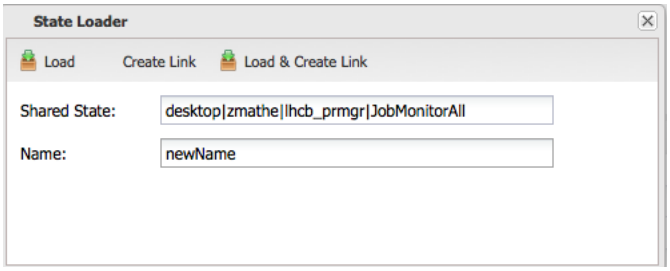
1. click on the menu item: Share
2. copy the text (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and click OK on the pop up window:
3. send the text (desktop|zmathe|lhcb_prmgr|JobMonitorAll) to the person



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:

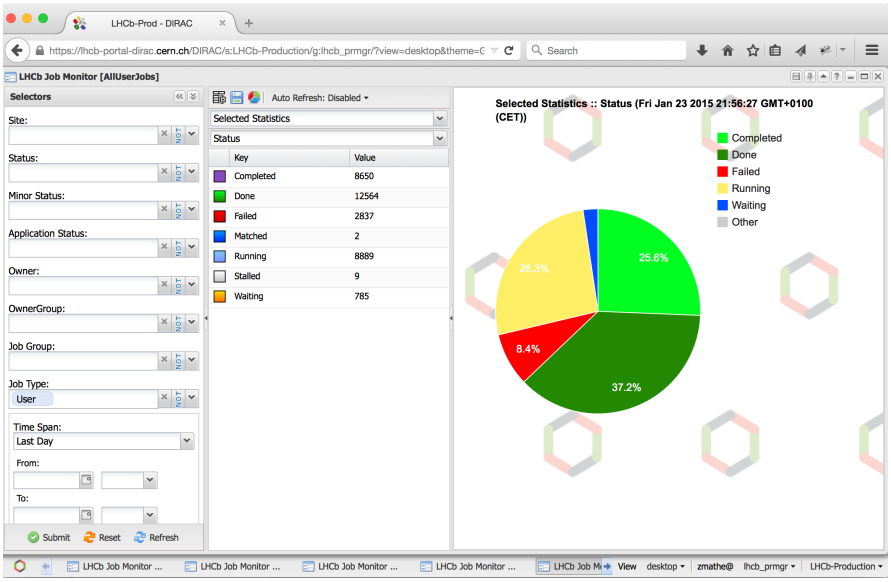


You have to provide the Shared State (for example: desktop|zmathe|lhcb_prmgr|JobMonitorAll) and a name (for example: newName). You have tree different way to load a shared state:

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

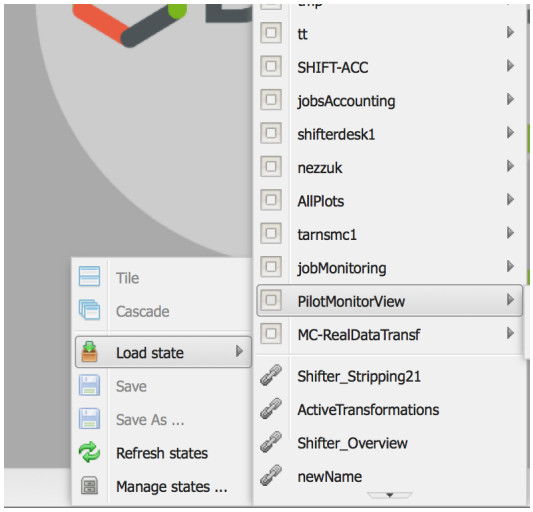
Load

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



Create Link

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

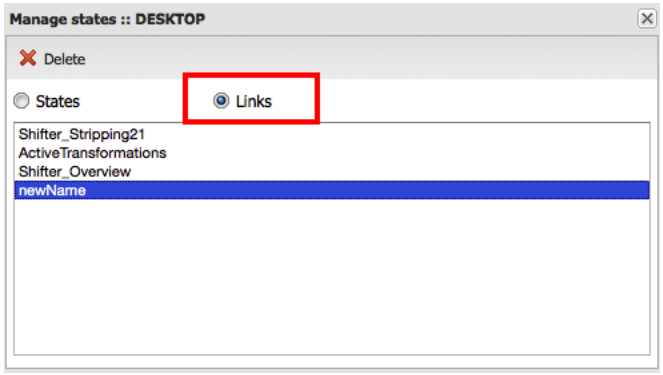


Load & Create Link

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.



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:

```
LFN:      Logical File Name
Path:     Local path to the file
SE:       DIRAC Storage Element
GUID:     GUID to use in the registration (optional)
```

OR

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]
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

Options:

```
-f  --force                   : Force overwrite of existing file
```

Example:

```
$ 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 specifed file

Usage:

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

Example:

```
$ 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:

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

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:

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

Example:

```
$ dirac-dms-clean-directory /formation/user/v/vhamar/newDir
Cleaning directory /formation/user/v/vhamar/newDir ...  OK
```

dirac-dms-create-removal-request

Create a DIRAC RemoveReplicalRemoveFile request to be executed by the RMS

Usage:

```
dirac-dms-create-removal-request [option|cfgfile] ... SE LFN ...
```

Arguments:

```
SE:       StorageElement|All
LFN:      LFN or file containing a List of LFNs
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

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-directory-sync

Provides basic rsync functionality for DIRAC Usage:

```
 dirac-dms-directory-sync Source Destination

e.g.: Download
  dirac-dms-directory-sync LFN Path
 or Upload
  dirac-dms-directory-sync Path LFN SE
```

Arguments:

```
LFN:           Logical File Name (Path to directory)
Path:          Local path to the file (Path to directory)
SE:            DIRAC Storage Element
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

Options:

```
-D  --sync                    : Make target directory identical to source
-j  --parallel <value>        : Multithreaded download and upload
```

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
cd     chown  get    id    meta   register  rm         size
chgrp  exit   group  lcd   mkdir  replicas  rmdir      unregister

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

FC:/>
```

dirac-dms-find-lfns

Find files in the FileCatalog using file metadata Usage:

```
dirac-dms-find-lfns [options] metaspec [metaspec ...]
```

Arguments:

```
metaspec:    metadata index specification (of the form: "meta=value" or "meta<value",
↪"meta!=value", etc.)
```

Examples:

```
$ dirac-dms-find-lfns Path=/lhcb/user "Size>1000" "CreationDate<2015-05-15"
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

Options:

```
-   --Path=                   :     Path to search for
-   --SE=                     :     (comma-separated list of) SEs/SE-groups to be␣
↪searched
```

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-ftsdb-summary

monitor FTSDB content Usage:

```
dirac-dms-ftsdb-summary [option|cfgfile]
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

dirac-dms-move-replica-request

Create a DIRAC MoveReplica request to be executed by the RMS

Usage:

```
dirac-dms-move-replica-request [option|cfgfile] ... sourceSE LFN targetSE1 [targetSE2␣
↪...]
```

Arguments:

```
sourceSE:   source SE
targetSE: target SE
LFN:      LFN or file containing a List of LFNs
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

dirac-dms-put-and-register-request

create and put 'PutAndRegister' request with a single local file

> **warning: make sure the file you want to put is accessible from DIRAC production hosts,** i.e. put file on network fs (AFS or NFS), otherwise operation will fail!!!

Usage:

```
dirac-dms-put-and-register-request [option|cfgfile] requestName LFN localFile targetSE
```

Arguments:

```
requestName: a request name
        LFN: logical file name   localFile: local file you want to put
   targetSE: target SE
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

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-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-and-register-request

create and put 'ReplicateAndRegister' request Usage:

```
dirac-dms-replicate-and-register-request [option|cfgfile] requestName LFNs targetSE1␣
↪[targetSE2 ...]
```

Arguments:

```
requestName: a request name
      LFNs: single LFN or file with LFNs
  targetSE: target SE
 2017-11-24 18:26:04 UTC Framework NOTICE::
```

General options:

```
2017-11-24 18:26:04 UTC Framework NOTICE:   -o  --option <value>          :␣
↪Option=value to add
2017-11-24 18:26:04 UTC Framework NOTICE:   -s  --section <value>         : Set base␣
↪section for relative parsed options
2017-11-24 18:26:04 UTC Framework NOTICE:   -c  --cert <value>            : Use server␣
↪certificate to connect to Core Services
2017-11-24 18:26:04 UTC Framework NOTICE:   -d  --debug                   : Set debug␣
↪mode (-ddd is extra debug)
2017-11-24 18:26:04 UTC Framework NOTICE:   -   --autoreload              :␣
↪Automatically restart if there's any change in the module
2017-11-24 18:26:04 UTC Framework NOTICE:   -   --license                 : Show DIRAC
↪'s LICENSE
2017-11-24 18:26:04 UTC Framework NOTICE:   -h  --help                    : Shows this␣
↪help
2017-11-24 18:26:04 UTC Framework NOTICE::
```

Options:

```
2017-11-24 18:26:04 UTC Framework NOTICE:   -C  --Catalog <value>         :    Catalog␣
↪to use
2017-11-24 18:26:04 UTC Framework NOTICE::
```

dirac-dms-resolve-guid

Returns the LFN matching given GUIDs Usage:

```
dirac-dms-resolve-guid <GUIDs>
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

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-ftsjobs

display information about FTSJobs for a given requestID Usage:

```
dirac-dms-show-ftsjobs [option|cfgfile] requestID
```

Argument:

```
requestID: RequestDB.Request.RequestID
```

General options:

```
-o  --option <value>          : Option=value to add
-s  --section <value>         : Set base section for relative parsed options
-c  --cert <value>            : Use server certificate to connect to Core Services
-d  --debug                   : Set debug mode (-ddd is extra debug)
-   --autoreload              : Automatically restart if there's any change in the␣
↪module
-   --license                 : Show DIRAC's LICENSE
-h  --help                    : Shows this help
```

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
```

(continues on next page)

(continued from previous page)

```
/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
```

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-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',
                                                         'Size': 34L,
                                                         'Status': 1,
                                                         'UID': 2}}}
```

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-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}
↪}}
```

1.4.2 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 [option|cfgfile]
```

Options:

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

dirac-wms-get-normalized-queue-length

Report Normalized CPU length of queue

Usage:

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

Arguments:

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

Example:

```
$ 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:

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

Arguments:

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

Example:

```
$ 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:

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

Arguments:

```
JobID:    DIRAC Job ID
```

Example:

```
$ 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:

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

Arguments:

```
JobID:    DIRAC Job ID
```

Example:

```
$ 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',
```

(continues on next page)

(continued from previous page)

```
 'JobRequirements': '[             OwnerDN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/
↪CN=Vanessa Hamar;            OwnerGroup = eela_user;             Setup = EELA-
↪Production;            UserPriority = 1;            CPUTime = 0        ]',
 '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:

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

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

Arguments:

```
JobID:    DIRAC Job ID
```

Example:

```
$ dirac-wms-job-logging-info 1
Status                        MinorStatus                        ApplicationStatus  ␣
↪          DateTime
Received                      Job accepted                       Unknown            ␣
↪          2011-02-14 10:12:40
Received                      False                              Unknown            ␣
↪          2011-02-14 11:03:12
Checking                      JobSanity                          Unknown            ␣
↪          2011-02-14 11:03:12
Checking                      JobScheduling                      Unknown            ␣
↪          2011-02-14 11:03:12
Waiting                       Pilot Agent Submission             Unknown            ␣
↪          2011-02-14 11:03:12
Matched                       Assigned                           Unknown            ␣
↪          2011-02-14 11:27:17
Matched                       Job Received by Agent              Unknown            ␣
↪          2011-02-14 11:27:27
Matched                       Submitted To CE                    Unknown            ␣
↪          2011-02-14 11:27:38
Running                       Job Initialization                 Unknown            ␣
↪          2011-02-14 11:27:42
Running                       Application                        Unknown            ␣
↪          2011-02-14 11:27:48
```

(continues on next page)

(continued from previous page)

```
Completed                      Application Finished Successfully   Unknown                 ␣
↪          2011-02-14 11:28:01
Completed                      Uploading Output Sandbox            Unknown                 ␣
↪          2011-02-14 11:28:04
Completed                      Output Sandbox Uploaded             Unknown                 ␣
↪          2011-02-14 11:28:07
Done                           Execution Complete                  Unknown                 ␣
↪          2011-02-14 11:28:07
```

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-lcgpbs-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)CPU5110@1.60GHz',
 'NormCPUTime(s)': '1.02',
 'OK': 'True',
 'OutputSandboxMissingFiles': 'std.err',
 'PayloadPID': '604',
 'PilotAgent': 'EELADIRAC v1r1; DIRAC v5r12',
 'Pilot_Reference': 'https://lb2.eela.ufrj.br:9000/ktM6WWR1GdkOTm98_hwM9Q',
 'ScaledCPUTime': '115.6',
 'TotalCPUTime(s)': '0.15'}
```

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(s)

-    --Date=            : Date in YYYY-MM-DD format, if not specified default is today

-    --Maximum=         : Maximum number of jobs shown (default or 0 means all)
```

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
 ConfigurationServer : ['dips://dirac.in2p3.fr:9135/Configuration/Server']
 VirtualOrganization : vo.formation.idgrilles.fr
```

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
-------------------------------------------------------------------------------------------
-----------------
| UserDN                                          | UserGroup   | ExpirationTime      |
| PersistentFlag |
-------------------------------------------------------------------------------------------
-----------------
| /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar | dirac_user  | 2011-06-29 12:04:25|
| True           |
-------------------------------------------------------------------------------------------
-----------------
```

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

-r   --rfc             : Create and RFC proxy style (https://www.ietf.org/rfc/rfc3820.
↪txt)
```

Example:

```
$ dirac-proxy-init -g dirac_user --rfc
Enter Certificate password:
$
```

dirac-version

v6r0

Example:

```
$ dirac-version
v5r12-pre9
```

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 destinated for tutorials. For more information about various options of installing DIRAC Client see the *Getting Started guide in :ref:'dirac_install*.

1.1 Install script

Download the *dirac-install* script from here:

```
curl https://github.com/DIRACGrid/DIRAC/raw/master/Core/scripts/dirac-install.py --
↪output=dirac-install
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*:

```
./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:

```
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 v6r20p14
```

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 -> Privacy & Security -> View Certificates -> Select your certificate -> Backup

  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/OU=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:

```
$ 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:

```
cd $DIRAC_PATH (if you set it)
source bashrc
```

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

```
dirac-proxy-init
```

  the above will create a proxy from the certificate in ~/.globus, with a default role. The switches below will create a proxy of group "dirac_user" (if defined) and will securely upload such proxy to the DIRAC proxy store (ProxyManager), from where it could later be downloaded:

```
dirac-proxy-init --group dirac_user --upload
```

  The additional "–debug" switch (alias of "-ddd") can be used for debugging purposes, and its output would end up being similar to the following:

```
$ dirac-proxy-init --group dirac_user --upload --debug
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
path         : /tmp/x509up_u501
username     : atsareg

Proxies uploaded:
```

(continues on next page)

(continued from previous page)

```
 DN                                                              | Group       | Until␣
↪(GMT)
 /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev | dirac_user | 2012/02/
↪08 13:05
```

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. This proxy is stored in the local "/tmp/" directory, as shown.

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:

```
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 can be uploaded to the ProxyManager service. To check that:

```
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
-------------------------------------------------------------------------------
↪----------------------
| UserDN                                          | UserGroup   | ExpirationTime ␣
↪    | PersistentFlag |
-------------------------------------------------------------------------------
↪----------------------
| /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar | dirac_user  | 2011-06-29␣
↪12:04:25 | True           |
-------------------------------------------------------------------------------
↪----------------------
```

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

```
Applications -> Proxy Manager
```

  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:

```
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 transfered 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:

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

- And create a simple shell script.

  testJob.sh:

```
#!/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:test.txt to srm://se0.m3pec.u-bordeaux1.fr:8446/srm/managerv2?SFN=/
↪dpm/m3pec.u-bordeaux1.fr/home/vo.formation.idgrilles.fr/user/v/vhamar/test.txt
2010-10-17 17:15:13 UTC dirac-dms-add-file.py  INFO: SRM2Storage.__putFile:␣
↪Successfully put file to storage.
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:

```
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:

```
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 acces 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:

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

For example:

```
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:

```
dirac-wms-job-status <Job_ID>

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:

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

1.5.4 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:

```
FC:/>cd /vo.formation.idgrilles.fr/user/a/atsareg
FC:/vo.formation.idgrilles.fr/user/a/atsareg>
FC:/vo.formation.idgrilles.fr/user/a/atsareg>cd
FC:/>cd /vo.formation.idgrilles.fr/user/a/atsareg
FC:/vo.formation.idgrilles.fr/user/a/atsareg>cd ..
FC:/vo.formation.idgrilles.fr/user/a>cd -
```

- Listing directory:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>ls -l
-rwxrwxr-x 0 atsareg dirac_user      856 2010-10-24 18:35:18 test.txt
```

- Creating new directory:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>mkdir newDir
FC:/vo.formation.idgrilles.fr/user/a/atsareg>ls -l
-rwxrwxr-x 0 atsareg dirac_user      856 2010-10-24 18:35:18 test.txt
drwxrwxr-x 0 atsareg dirac_user        0 2010-10-24 11:00:05 newDir
```

- Changing ownership and permissions:

```
FC:/vo.formation.idgrilles.fr/user/a/atsareg>chmod 755 newDir
FC:/vo.formation.idgrilles.fr/user/a/atsareg>ls -l
```

(continues on next page)

(continued from previous page)

```
-rwxrwxr-x 0 atsareg dirac_user      856 2010-10-24 18:35:18 test.txt
drwxr-xr-x 0 atsareg dirac_user        0 2010-10-24 11:00:05 newDir
```

5.3 Managing files and replicas

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

```
add <LFN> <local_file> <SE>
```

For example:

```
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
-rwxrwxr-x 0 atsareg dirac_user      856 2010-10-24 18:35:18 test.txt
```

- Download grid file to the local directory:

```
get <LFN> [<local_directory>]
```

For example:

```
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
```

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

```
replicate <lfn> <SE>
```

For example:

```
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
```

- 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
DIRAC-USER      dips://dirac.in2p3.fr:9148/DataManagement/StorageElement/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
DIRAC-USER      dips://dirac.in2p3.fr:9148/DataManagement/StorageElement/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:

```
id
```

For example:

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

1.5.5 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::

```
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)::

```
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-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
```

7.1.1 Creating and submitting parametric Jobs using DIRAC APIs

DIRAC APIs are an easy and convenient way to create and submit parametric jobs:

```
from DIRAC.Interfaces.API.Job import Job
from DIRAC.Interfaces.API.Dirac import Dirac
# or extensions, e.g. from LHCbDIRAC.Interfaces.API.LHCbJob import LHCbJob for LHCb

J = Job()
J.setCPUTime(17800)
J.setInputSandbox('exe-script.py') # whatever
J.setParameterSequence("args", ['one', 'two', 'three'])
J.setParameterSequence("iargs", [1, 2, 3])
J.setExecutable("exe-script.py", arguments=": testing %(args)s %(iargs)s", logFile=
↪'helloWorld_%n.log')
print Dirac().submitJob(J)
```

InputData (in the form of LFNs – Logical File Names) can become also parameters in parametric jobs:

```
inputDataList = [ # a list of lists
 [
     '/lhcb/data/data1',
     '/lhcb/data/data2'
 ],
 [
     '/lhcb/data/data3',
     '/lhcb/data/data4'
 ],
 [
     '/lhcb/data/data5',
     '/lhcb/data/data6'
 ]

J.setParameterSequence('InputData', inputDataList, addToWorkflow=True)
```

and similarly for InputSandbox:

```
inputSBList = [ # a list of lists
 [
     '/localFile.txt',
     '/another/localFile.py',
     '/some/lfn/some/where'
 ]

J.setParameterSequence('InputSandbox', inputSBList, addToWorkflow=True)
```

The list of parameters, whatever they are have to have ALL the same lenghth, e.g. there should not be a parameter of length 2 and another of length 3.

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::

```
$ more application.sh
#!/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 "========================================="
export 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/$TMP_DIR
    export PATH=$PATH:$DIR
    for i in `mpdtrace`;
    do
      ssh $i.$DOMAIN mkdir -p  $DIR
      scp $PWD/$EXECUTABLE* $i.$DOMAIN:$DIR/;
      ssh $i.$DOMAIN ls -la $DIR
    done;
  else
    DIR=$MPI_SHARED_HOME/$TMP_DIR
    mkdir $DIR
    cp $EXECUTABLE.o $DIR;
  fi
  $MPIRUN -np $NUMPROC $DIR/$EXECUTABLE.o
  x=`echo $MPI_SHARED_HOME`;
  if [[  -z "$x" ||  "$x" == "no" ]]; then
    for i in `mpdtrace`;
    do
      ssh $i.$DOMAIN 'rm -rf $DIR';
```

(continues on next page)

(continued from previous page)

```
    done;
  else
    cd ..
    rm -rf $DIR
  fi
else
  exit
fi
```

- 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::

```
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::

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

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

```
$ 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:

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

j = Job()
j.setCPUTime(500)
j.setExecutable('echo',arguments='hello')
j.setExecutable('ls',arguments='-l')
j.setExecutable('echo', arguments='hello again')
j.setName('API')
```

(continues on next page)

(continued from previous page)

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

- Run the script:

```
python Test-API.py

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

7.3.2 Retrieving Job Status

- Create a script Status-API.py:

```
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:

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

$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:

```
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:

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

$python Output-API.py 196
```

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().submitJob(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/Local_zbDHRe_JobDir
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
2010-10-22 14:41:51 UTC /DiracAPI  INFO: Executing: /afs/in2p3.fr/home/h/hamar/
↪DIRAC5/scripts/dirac-jobexec jobDescription.xml -o LogLevel=info
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 dirac-jobexec.py/Script  INFO: Output written to Script1_
↪CodeOutput.log, execution complete.
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.setCPUTime(500)
j.setExecutable('echo',arguments='hello')
for i in range(5):
  j.setName('API_%d' % i)
  dirac = Dirac()
  jobID = dirac.submitJob(j)
  print 'Submission Result: ',jobID
```

- Execute the script:

```
$ python Test-API-Multiple.py
Submission Result:  {'OK': True, 'Value': 176}
Submission Result:  {'OK': True, 'Value': 177}
Submission Result:  {'OK': True, 'Value': 178}
```

7.3.6 Using APIs to create JDL files.

- Create a Test-API-JDL.py:

```
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.setPlatform("")
j.setCPUTime(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:

```
$ 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"
```

(continues on next page)

(continued from previous page)

```
        };
    StdOutput = "std.out";
    Site = "LCG.IN2P3.fr";
    Platform = "";
    OutputPath = "MyFirstAnalysis";
    InputSandbox = "jobDescription.xml";
    Arguments = "jobDescription.xml -o LogLevel=DEBUG";
    JobGroup = "vo.formation.idgrilles.fr";
    OutputSandbox =
        {
            "*.log",
            "summary.data",
            "Script1_CodeOutput.log",
            "std.err",
            "std.out"
        };
    MaxCPUTime = "21600";
    JobName = "APItoJDL";
    InputData =
        {
            "LFN:/vo.formation.idgrilles.fr/user/v/vhamar/test.txt",
            "LFN:/vo.formation.idgrilles.fr/user/v/vhamar/test2.txt"
        };
    JobType = "User";
```

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.

7.3.7 Submitting MultiProcessor (MP) jobs

Jobs that can (or should) run using more than 1 processor should be described as such, using the "Tag" mechanism:

```
j = Job()
j.setCPUTime(500)
j.setExecutable('echo',arguments='hello')
j.setExecutable('ls',arguments='-l')
j.setExecutable('echo', arguments='hello again')
j.setName('MP test')
```

<this is today possible by using setTag() but the specific Tag to use is not yet carved in stone>

<to expand, e.g. put about NumberOfProcessor = X that becomes XNumberOfProcessors>

7.3.8 Submitting jobs with specifc requirements (e.g. GPU)

<to expand, ~same as for MP jobs, i.e. use Tags>

1.6 HOW-TO Guides

This section lists how-to and FAQ

1.6.1 DataManagement

For an introduction about DataManagement concepts, please see the *introduction*

All the commands mentionned bellow can accept several StorageElements and LFNs as parameters. Please use *–help* for more details.

Basics

How to list one's own files

For a user to know its own file list:

```
[Dirac prod] chaen $ dirac-dms-user-lfns
Will search for files in /lhcb/user/c/chaen
/lhcb/user/c/chaen: 5 files, 2 sub-directories
/lhcb/user/c/chaen/GangaInputFile: 0 files, 1 sub-directories
/lhcb/user/c/chaen/GangaInputFile/Job_2: 1 files, 0 sub-directories
/lhcb/user/c/chaen/subDir: 1 files, 0 sub-directories
7 matched files have been put in lhcb-user-c-chaen.lfns
```

How to see the various replicas of a file

To list the SE where a file is stored:

```
[DIRAC prod] chaen $ dirac-dms-lfn-replicas /lhcb/user/c/chaen/diracTutorial.txt
Successful :
    /lhcb/user/c/chaen/diracTutorial.txt :
        CERN-USER : srm://srm-eoslhcb.cern.ch:8443/srm/v2/server?SFN=/eos/lhcb/grid/
↪user/lhcb/user/c/chaen/diracTutorial.txt
         RAL-USER : srm://srm-lhcb.gridpp.rl.ac.uk:8443/srm/managerv2?SFN=/castor/ads.
↪rl.ac.uk/prod/lhcb/user/c/chaen/diracTutorial.txt
```

How to get the xroot URL for my LFN

In order to get an xroot URL usable from for example ROOT:

```
[DIRAC prod] chaen $ dirac-dms-lfn-accessURL --Protocol=root,xroot /lhcb/user/c/chaen/
↪diracTutorial.txt RAL-USER
Successful :
    RAL-USER :
        /lhcb/user/c/chaen/diracTutorial.txt : root://clhcbstager.ads.rl.ac.uk//
↪castor/ads.rl.ac.uk/prod/lhcb/user/c/chaen/diracTutorial.txt?svcClass=lhcbUser
```

If you do not specify a StorageElement, DIRAC will check the URLs for all the replicas:

```
[DIRAC prod] chaen $ dirac-dms-lfn-accessURL --Protocol=root,xroot /lhcb/user/c/chaen/
↪diracTutorial.txt
Using the following list of SEs: ['CERN-USER', 'RAL-USER']
Successful :
    CERN-USER :
        /lhcb/user/c/chaen/diracTutorial.txt : root://eoslhcb.cern.ch//eos/lhcb/grid/
↪user/lhcb/user/c/chaen/diracTutorial.txt
```

(continues on next page)

(continued from previous page)

```
    RAL-USER :
        /lhcb/user/c/chaen/diracTutorial.txt : root://clhcbstager.ads.rl.ac.uk//
↪castor/ads.rl.ac.uk/prod/lhcb/user/c/chaen/diracTutorial.txt?svcClass=lhcbUser
```

How to upload a file to a grid storage

To put a local file to a GRID storage:

```
[DIRAC prod] chaen $ dirac-dms-add-file /lhcb/user/c/chaen/diracTutorial.txt ./
↪diracTutorial.txt CERN-USER
Could not obtain GUID from file through Gaudi, using standard DIRAC method

Uploading ./diracTutorial.txt as /lhcb/user/c/chaen/diracTutorial.txt
Successfully uploaded ./diracTutorial.txt to CERN-USER (0.7 seconds)
```

How to replicate an LFN to another storage

The file has to be already on a grid storage:

```
[DIRAC prod] chaen $ dirac-dms-replicate-lfn  /lhcb/user/c/chaen/diracTutorial.txt␣
↪RAL-USER
Successful :
    RAL-USER :
        /lhcb/user/c/chaen/diracTutorial.txt :
             register : 0.216005086899
            replicate : 5.27293300629
```

How to have the metadata of an file

To get the metadata of an LFN as stored in the catalog:

```
[DIRAC prod] chaen $ dirac-dms-lfn-metadata /lhcb/user/c/chaen/diracTutorial.txt
Successful :
    /lhcb/user/c/chaen/diracTutorial.txt :
                Checksum : 2a810562
            ChecksumType : Adler32
            CreationDate : 2018-12-20 18:33:40
                  FileID : 390920814
                     GID : 2746
                    GUID : 15C4C7B2-47F3-9BDE-CA19-60A1E348EF90
                    Mode : 775
        ModificationDate : 2018-12-20 18:33:40
                   Owner : chaen
              OwnerGroup : lhcb_prmgr
                    Size : 14
                  Status : AprioriGood
                     UID : 20269
```

To get the metadata of the file actually stored, you can use the following command (with or without SE specification):

```
[DIRAC prod] chaen $ dirac-dms-pfn-metadata /lhcb/user/c/chaen/diracTutorial.txt
Getting replicas for 1 files : completed in 0.1 seconds
Getting SE metadata of 2 replicas : completed in 1.5 seconds
Successful :
    /lhcb/user/c/chaen/diracTutorial.txt :
        CERN-USER :
                  Accessible : True
                    Checksum : 2a810562
                   Directory : False
                  Executable : False
                        File : True
              FileSerialNumber : 10376293541461674751
                     GroupID : 1470
                  LastAccess : 2018-12-20 19:33:39
                       Links : 1
                     ModTime : 2018-12-20 19:33:39
                        Mode : 400
                    Readable : True
                        Size : 14
                 StatusChange : 2018-12-20 19:33:39
                      UserID : 56212
                   Writeable : False

       RAL-USER :
                  Accessible : True
                      Cached : 1
                    Checksum : 2a810562
                   Directory : False
                  Executable : False
                        File : True
              FileSerialNumber : 0
                     GroupID : 46
                  LastAccess : Never
                       Links : 1
                        Lost : 0
                    Migrated : 0
                     ModTime : 2018-12-20 19:35:13
                        Mode : 644
                    Readable : True
                        Size : 14
                 StatusChange : 2018-12-20 19:35:13
                  Unavailable : 0
                       UserID : 45
                   Writeable : True

[DIRAC prod] chaen $ dirac-dms-pfn-metadata /lhcb/user/c/chaen/diracTutorial.txt CERN-
↪USER
Getting replicas for 1 files : completed in 0.1 seconds
Getting SE metadata of 1 replicas : completed in 1.0 seconds
Successful :
    /lhcb/user/c/chaen/diracTutorial.txt :
        CERN-USER :
                  Accessible : True
                    Checksum : 2a810562
                   Directory : False
                  Executable : False
                        File : True
```

(continues on next page)

(continued from previous page)

```
            FileSerialNumber : 10376293541461674751
                     GroupID : 1470
                  LastAccess : 2018-12-20 19:33:39
                       Links : 1
                     ModTime : 2018-12-20 19:33:39
                        Mode : 400
                    Readable : True
                        Size : 14
                StatusChange : 2018-12-20 19:33:39
                      UserID : 56212
                   Writeable : False
```

How to remove a replica of a file

In order to remove one of the replicas:

```
[DIRAC prod] chaen $ dirac-dms-remove-replicas /lhcb/user/c/chaen/diracTutorial.txt␣
↪CERN-USER
Removing replicas : completed in 1.8 seconds
Successfully removed 1 replicas from CERN-USER
```

How to remove a file from the grid

Watch out, this will remove all the replicas of a file:

```
[DIRAC prod] chaen $ dirac-dms-remove-files /lhcb/user/c/chaen/diracTutorial.txt
Removing 1 files : completed in 1.9 seconds
Successfully removed 1 files
```

DFC as a metadata catalog

This section supposes that the DFC is used as a Metadata Catalog. This is for example not the case of LHCb. Please ask your administrator if you are unsure. The exercises are performed using the File Catalog CLI interface. You can start the CLI with the command:

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

How to add metadata to a directory

From the CLI:

```
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
```

How to get directory metadata

From the CLI:

```
meta get <directory>
```

For example:

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

How to create metadata index

From the CLI:

```
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
```

How to show existing metadata indices

From the CLI:

```
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
```

How to find files with selection by metadata

From the CLI:

```
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
```

How to declare file's ancestors

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
```

How to query file's ancestors

It can be interrogated with the following commands:

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

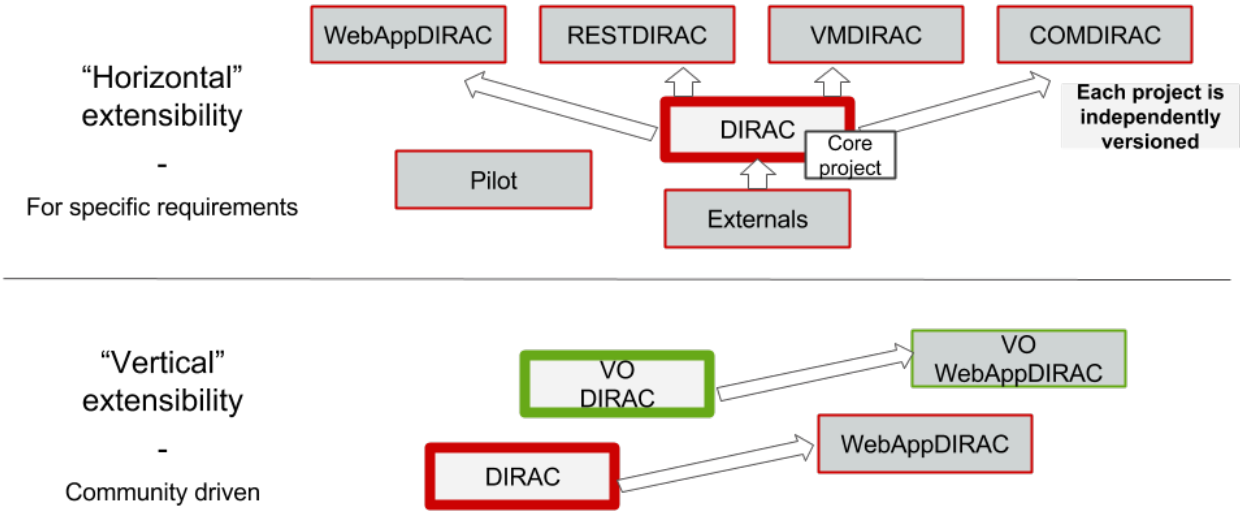
For example:

```
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
1      /vo.formation.idgrilles.fr/user/a/atsareg/file2
2              /vo.formation.idgrilles.fr/user/a/atsareg/file3
```

CHAPTER 2

Administrator Guide

DIRAC has been developed with extensibility and flexibility in mind. A DIRAC release is composed by few projects, like in the following picture. This administration documentation refers to the “Core” DIRAC project.



2.1 DIRAC Setup Structure

The basic DIRAC components are *Services*, *Agents*, and *Executors*.

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 active components, similar to cron jobs, which execution is invoked periodically. Agents are animating the whole system by executing actions, sending requests to the DIRAC or third party services.

Executors are also active components, similar to consumers of a message queue system, which execution is invoked at request. Executors are used within the DIRAC Workload Management System.

These components are combined together to form *Systems*. a *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 or Data Management System.

And then there are databses, which keep the persistent state of a *System*. They are accessed by Services and Agents as a kind of shared memory.

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*.

A *Setup* is the highest level of the DIRAC component hierarchy. *Setups* are combining together instances of *Systems*. Within a given installation there may be 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 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 following services, which constitute what is considered as a minimal DIRAC installation:

- The *Configuration Service (CS)*: the CS is backbone for the entire DIRAC system. Please refer to *DIRAC Configuration* for more information

- The *SystemAdministrator* service which, once installed, allows remote management of the DIRAC components directly on the server.
- The *Component Monitoring* service is for keeping track of installed components. Refer to static_component_monitoring for more info.
- The *Resource Status* service will keep track of the status of your distributed computing resources. Refer to *Resource Status System* for more info.

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.

2.2.1 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
```

- DIRAC extensions that need specific services which are not an extension of DIRAC used should better use ports 9201-9300 in order to avoid confusion. If this happens, the procedure above should be repeated to include the new range of ports.
- 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 usally 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)

2.2.2 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:

```
mkdir -p /opt/dirac/etc/grid-security/
cp hostcert.pem hostkey.pem /opt/dirac/etc/grid-security
```

  In case your host certificate is in the p12 format, you can convert it with:

```
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:

```
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:

```
mkdir /home/dirac/DIRAC
cd /home/dirac/DIRAC
curl https://github.com/DIRACGrid/DIRAC/raw/integration/Core/scripts/install_site.
↪sh -O
```

2.2.3 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 nto 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 complains.

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:

```
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.

2.2.4 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:

```
#
# 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 = v6r20p14
  #  Python version of the installation (default: 2.7)
  # PythonVersion = 27
  #  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 = v14r2
  #  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:
```

(continues on next page)

(continued from previous page)

```
  #   i.e. LHCb, LHCbWeb for LHCb
  Extensions = WebApp

  #
  #   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 (every installation can have multiple setups, but give a name to␣
↪the first one)
  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
  #  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
```

(continues on next page)

(continued from previous page)

```
  # List of Systems to be installed - by default all services are added
  Systems = Accounting
  Systems += Configuration
  Systems += DataManagement
  Systems += Framework
  Systems += Monitoring
  Systems += RequestManagement
  Systems += ResourceStatus
  Systems += StorageManagement
  Systems += Transformation
  Systems += WorkloadManagement
  #
  # List of DataBases to be installed (what's here is a list for a basic␣
↪installation)
  Databases = InstalledComponentsDB
  Databases += ResourceStatusDB
  #
  #  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
  #  List of Services to be installed (what's here is a list for a basic␣
↪installation)
  Services  = Configuration/Server
  Services += Framework/ComponentMonitoring
  Services += Framework/SystemAdministrator
  Services += ResourceStatus/ResourceStatus
  #  Flag determining whether the Web Portal will be installed
  WebPortal = yes
  WebApp = 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
    #  There are 2 flags for small and large installations Set either of them to␣
↪True/yes when appropriated
    # 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
```

(continues on next page)

(continued from previous page)

```
    #                       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). While not strictly necessary, it's advised that a version is added with the '-v' switch (pick the most recent one, see release notes in https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/release.notes):

```
./install_site.sh -v v6r20p14 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.:

```
                          Name : Runit     Uptime    PID
         Configuration_Server : Run           41    30268
Framework_SystemAdministrator : Run           21    30339
Framework_ComponentMonitoring : Run           11    30340
ResourceStatus_ResourceStatus : Run            9    30341
                     Web_httpd : Run            5    30828
                   Web_paster : Run             5    30829
```

Now the basic services - Configuration, SystemAdministrator, ComponentMonitoring and ResourceStatus - are installed, or at least their DBs should be installed, and their services up and running.

There are anyway a couple more steps that should be done to fully activate the ComponentMonitoring and the ResourceStatus. These steps can be found in the respective administration sessions of this documentation:

- static_component_monitoring for the static component monitoring (the ComponentMonitoring service)
- *Installation* and *Populate tables* for the Resource Status System

but, no hurry: you can do it later.

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:

```
#!/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
```

2.2.5 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:

```
#
# 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 = v6r20p14
  #  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 = v14r2
  #  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
  Externals =

  #
  #   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
```

(continues on next page)

(continued from previous page)

```
  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::

```
./install_site.sh -v v6r20p14 install.cfg
```

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*

2.2.6 Post-Installation step

In order to make the DIRAC components running we use the *runit* mechanism (http://smarden.org/runit/). You could also use the RPM provided by LHCb at http://cern.ch/lhcbproject/dist/rpm/lhcbdirac/[slc6/runit-2.1.2-1.el6.x86_64.rpm, centos7/runit-2.1.2-1.el7.cern.x86_64.rpm]. 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
```

or if using `systemd` (in CENTOS7 for example), add a file `/etc/systemd/system/multi-user.target.wants/dirac.service` containing:

```
[Service]
ExecStart=/opt/dirac/sbin/runsvdir-start
Restart=on-failure
```

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/*
killall runsv svlogd
killall 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.

2.2.7 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 --rfc
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>
```

2.2.8 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 comand 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
```

2.3 Installing WebAppDIRAC

The first section describes the install procedure of the web framework. The configuration of the web will be presented in the next sections. While not mandatory, NGINX (nginx.com) can be used to improve the performance of the web framework. The installation and configuration of NGINX will be presented in the last section.

2.3.1 Requirements

It is required CERN supported OS (slc6, CentOS 7, etc.) distribution. We recommend you to use the latest official OS version. Please follow the *Requirements* instructions to setup the machine. In principle there is no magic to install the web portal. It has to be installed as another DIRAC component... When the machine is ready you can start to install the web portal. But before that you need the install_site.sh script and a minimal configuration file.

Getting the install script

You can found the instruction about were to get the install_site.sh at the end of the *Requirements* section.

Configuration file

You can use a standard configuration file for example *Primary server installation*. Please make sure that the following lines are exists in the configuration file:

```
Externals = WebApp
WebApp = yes
```

$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 = v6r20p14
  #  Python version of the installation
  PythonVersion = 27
  #  To install the Server version of DIRAC (the default is client)
  InstallType = server
  #  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 extension to be installed
  # (WebApp 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 for example: Extensions = WebAppDIRAC,LHCb,LHCbWeb
  Externals = WebApp
  Project = DIRAC
  WebPortal = yes
  WebApp = yes
  Services = Framework/SystemAdministrator
  UseServerCertificate = yes
  SkipCADownload = yes
  Setup = your setup # for example: LHCb-Certification
  ConfigurationMaster = no
  ConfigurationServer = your configuration service
}
```

Before you start the installation please make sure that you have the host certificate in the /opt/dirac/etc directory. More info in the Server Certificates section in *Requirements* .

Create the configuration file:

```
  - vim /home/dirac/DIRAC/install.cfg
  - copy the lines above the this file...
  - cd /home/dirac/DIRAC
  - chmod +x install_site.sh
```

(continues on next page)

(continued from previous page)

```
  - ./install_site.sh install.cfg # use -v for specifying a version
  - source /opt/dirac/bashrc

Note: If you do not have the /home/dirac/DIRAC directory, please have a look the␣
↪instructions given in the :ref:`server_requirements` section.
```

Checks to be done after the installation

If the installation is successful, you will see the following lines:

```
Status of installed components:

   Name                            Runit Uptime PID
=====================================================
 1 Web_WebApp                      Run    6      19887
 2 Framework_SystemAdministrator Run    2      19941
```

Make sure that the portal is listening in the correct port:

```
Without NGinx::

tail -200f /opt/dirac/runit/Web/WebApp/log/current

2016-06-02 12:44:18 UTC WebApp/Web   INFO: Configuring in developer mode...
2016-06-02 12:44:18 UTC WebApp/Web NOTICE: Configuring HTTP on port 8080
2016-06-02 12:44:18 UTC WebApp/Web NOTICE: Configuring HTTPS on port 8443
2016-06-02 12:44:19 UTC WebApp/Web ALWAYS: Listening on https://0.0.0.0:8443/DIRAC/␣
↪and http://0.0.0.0:8080/DIRAC/


Using Nginx::

tail -200f /opt/dirac/runit/Web/WebApp/log/current

The output of the command::

2016-06-02 12:35:46 UTC WebApp/Web NOTICE: Configuring HTTP on port 8000
2016-06-02 12:35:46 UTC WebApp/Web ALWAYS: Listening on http://0.0.0.0:8000/DIRAC/
```

If you are not using NGINX and the web server is listening on 8000, please open vim /opt/dirac/pro/WebAppDIRAC/WebApp/web.cfg and add Balancer=None. Make sure that the configuration /opt/dirac/pro/etc/dirac.cfg file is correct. It contains Extensions = WebApp. For example:

```
DIRAC
{
  Setup = LHCb-Certification
  Configuration
  {
    Servers =
  }
  Security
  {
  }
  Extensions = WebApp
  Setups
```

(continues on next page)

(continued from previous page)

```
  {
    LHCb-Certification
    {
      Configuration = LHCb-Certification
      Framework = LHCb-Certification
    }
  }
}
```

- Update using: **dirac-admin-sysadmin-cli**
  - dirac-admin-sysadmin-cli -H hostname
  - update version of DIRAC, for example v8r1

2.3.2 Web configuration file

We use **web.cfg** configuration file, which is used to configure the web framework. It also contains the schema of the menu under Schema section, which is used by the users. The structure of the web.cfg file is the following:

```
WebApp
{
  Balancer = None #[nginx] in case you have installed nginx
  #NumProcesses = 1
  #SSLProrocol = "" [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/Production/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
```

(continues on next page)

(continued from previous page)

```
    }
    DIRAC = link|http://diracgrid.org
  }
}
```

Define external links:

```
Web
{
    Lemon Host Monitor
    {
      volhcb01 = link|https://lemonweb.cern.ch/lemon-web/info.php?entity=lbvobox01&
↪detailed=yes
    }
}
```

The default location of the configuration file is /opt/dirac/pro/WebAppDIRAC/WebApp/web.cfg. This is the default configuration file which provided by by the developer. If you want to change the default configuration file, you have to add the web.cfg to the directory where the dirac.cfg is found, for example: /opt/dirac/etc

If the web.cfg file exists in /opt/dirac/etc directory, this file will be used.

Note: The Web framework uses the Schema section for creating the menu. It shows the Schema content, without manipulating it for example: sorting the applications, or creating some structure. Consequently, if you want to sort the menu, you have to create your own configuration file and place the directory where dirac.cfg exists.

2.3.3 Running multiple web instances

If you want to run more than one instance, you have to use NGIX. The configuration of the NGINX is described in the next section. You can define the number of processes in the configuration file: /opt/dirac/pro/WebAppDIRAC/WebApp/web.cfg

NumProcesses = x (by default the NumProcesses is 1), where x the number of instances, you want to run Balancer = nginx

for example:: NumProcesses = 4, the processes will listen on 8000, 8001, ... 800n

You can check the number of instances in the log file (runit/Web/WebApp/log/current):

```
2018-05-09 13:48:28 UTC WebApp/Web NOTICE: Configuring HTTP on port 8000
2018-05-09 13:48:28 UTC WebApp/Web NOTICE: Configuring HTTP on port 8001
2018-05-09 13:48:28 UTC WebApp/Web NOTICE: Configuring HTTP on port 8002
2018-05-09 13:48:28 UTC WebApp/Web NOTICE: Configuring HTTP on port 8003
2018-05-09 13:48:28 UTC WebApp/Web ALWAYS: Listening on http://0.0.0.0:8002/DIRAC/
2018-05-09 13:48:28 UTC WebApp/Web ALWAYS: Listening on http://0.0.0.0:8000/DIRAC/
2018-05-09 13:48:28 UTC WebApp/Web ALWAYS: Listening on http://0.0.0.0:8001/DIRAC/
2018-05-09 13:48:28 UTC WebApp/Web ALWAYS: Listening on http://0.0.0.0:8003/DIRAC/
```

You have to configure NGINX to forward the requests to that ports:

```
upstream tornadoserver {
    #One for every tornado instance you're running that you want to balance
    server 127.0.0.1:8000;
    server 127.0.0.1:8001;
    server 127.0.0.1:8002;
    server 127.0.0.1:8003;
}
```

2.3.4 Install and configure NGINX

Note: you can run NGINX in a separate machine.

The official site of NGINX is the following: http://nginx.org/ The required NGINX version has to be grater than 1.4.

- Install Nginx using package manager:

```
yum install nginx
```

If your version is not grater than 1.4 you have to install NGINX manually.

- Manual install:

```
vim /etc/yum.repos.d/nginx.repo
```

CentOS:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=0
enabled=1
```

RHEL:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/rhel/$releasever/$basearch/
gpgcheck=0
enabled=1
```

Due to differences between how CentOS, RHEL, and Scientific Linux populate the $releasever variable, it is necessary to manually replace $releasever with either 5 (for 5.x) or 6 (for 6.x), depending upon your OS version. For example:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/rhel/6/$basearch/
gpgcheck=0
enabled=1
```

If it is successful installed:

```
 Verifying  : nginx-1.10.1-1.el6.ngx.x86_64
↪
↪
↪ 1/1
Installed:
   nginx.x86_64 0:1.10.1-1.el6.ngx
```

- Configure NGINX

  You have to find the nginx.conf file. You can see which configuration used in /etc/init.d/nginx. For example:

```
vim /etc/nginx/nginx.conf
```

  If the file contains 'include /etc/nginx/conf.d/*.conf;' line, you have to create a site.conf under /etc/nginx/conf.d/ otherwise you have to do: 'include /etc/nginx/site.conf'

The content of the site.conf (please modify it!!!):

```
#Generated by gen.py

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  lbvobox33.cern.ch;
  server_name dzmathe.cern.ch;

  root /opt/dirac/pro;

  location ~ ^/[a-zA-Z]+/(s:.*/g:.*/)?static/(.+\.(jpg|jpeg|gif|png|bmp|ico|pdf))$ {
    alias /opt/dirac/pro/;
    #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/pro/;
    #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;

    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)
```

(continues on next page)

(continued from previous page)

```
   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  lbvobox33.cern.ch;
  server_name  dzmathe.cern.ch;

  ssl_prefer_server_ciphers On;
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers␣
↪ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES
↪aNULL:!MD5:!DSS;

  #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/cas.pem;
#  ssl_crl /opt/dirac/pro/etc/grid-security/allRevokedCerts.pem;
  ssl_verify_client on;
  ssl_verify_depth 10;
  ssl_session_cache shared:SSL:10m;

  root /opt/dirac/pro;

  location ~ ^/[a-zA-Z]+/(s:.*/g:.*/)?static/(.+\.(jpg|jpeg|gif|png|bmp|ico|pdf))$ {
    alias /opt/dirac/pro/;
    #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/pro/;
    #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;
```

(continues on next page)

(continued from previous page)

```
  }
  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 can start NGINX now.

- Start, Stop and restart nginx:

```
/etc/init.d/nginx start|stop|restart
```

You have to add to the web.cfg the following lines in order to use NGINX:

```
DevelopMode = False
Balancer = nginx
NumProcesses = 1
```

In that case one process will be used and this process is listening on 8000 port. You can try to use the web portal. For example: http://dzmathe.cern.ch/DIRAC/ If you get 502 Bad Gateway error, you need to generate rules for SE linus. You can see the error in tail -200f /var/log/nginx/error.log:

```
016/06/02 15:55:24 [crit] 20317#20317: *4 connect() to 127.0.0.1:8000 failed (13:␣
↪Permission denied) while connecting to upstream, client: 128.141.170.23,␣
↪server: dzmathe.cern.ch, request: "GET /DIRAC/?view=tabs&theme=Grey&url_␣
↪state=1| HTTP/1.1", upstream: "http://127.0.0.1:8000/DIRAC/?view=tabs&
↪theme=Grey&url_state=1|", host: "dzmathe.cern.ch"
```

- **Generate the the rule::**

- grep nginx /var/log/audit/audit.log | audit2allow -M nginx
- semodule -i nginx.pp
- rferesh the page

2.4 VMDIRAC

Table of contents

2.4.1 Install VMDIRAC

On the server running the WMS:

- Install VMDIRAC extension as any other DIRAC extension using -e option *e.g.*:

```
./dirac-install -l $release-project -r $release_version -e VMDIRAC
```

Note that in the server local configuration you should have the following option set in the LocalInstallation and DIRAC sections:

```
Extensions = VMDIRAC
```

- Server local configuration

In the server local configuration add the User/Password information to connect to the Cloud endpoint. Also you should put a valid path for the host certificate, *e.g.*:

```
Resources
{
  Sites
  {
    Cloud
    {
      Cloud.LUPM.fr
      {
        Cloud
        {
          194.214.86.244
          {
            User = xxxx
            Password = xxxx
            HostCert = /opt/dirac/etc/grid-security/hostcert.pem
            HostKey = /opt/dirac/etc/grid-security/hostkey.pem
            CreatePublicIP = True
          }
        }
```

(continues on next page)

(continued from previous page)

```
      }
    }
  }
}
```

- Install the following components:
  - DB: VirtualMachineDB
  - Service: WorkloadManagement_VirtualMachineManager
  - Agent: WorkloadManagement_CloudDirector

2.4.2 Configuration

- In the CS Resources section, configure the cloud endpoint as in this example

```
Resources
{
  Sites
  {
    Cloud
    {
      Cloud.LUPM.fr
      {
        CE = 194.214.86.244
        Cloud
        {
          194.214.86.244
          {
            CEType = Cloud
            ex_security_groups = default
            ex_force_auth_url = http://194.214.86.244:5000/v3/auth/tokens
            ex_force_service_region = LUPM-CLOUD
            # This is the max number of VM instances that will be running in␣
↪parallel
            # Each VM can have multiple cores, each one executing a job
            MaxInstances = 4
            ex_force_auth_version = 3.x_password
            ex_tenant_name = dirac
            ex_domain_name = msfg.fr
            networks = dirac-net
            # This is the public key previously uploaded to the Cloud provider
            # It's needed to ssh connect to VMs
            keyname = cta_cloud_lupm
            # If this option is set, public IP are assigned to VMs
            # It's needed to ssh connect to VMs
            ipPool = ext-net
          }
          Images
          {
            # It can be a public or a private image
            Centos6-Officielle
            {
              ImageID = 35403255-f5f1-4c61-96dc-e59678942c6d
              FlavorName = m1.medium
```

(continues on next page)

(continued from previous page)

```
            }
          }
        }
      }
    }
  }
}
```

- CS Operation section

```
Operations
{
  CTA
  {
    Cloud
    {
      GenericCloudGroup = cta_genpilot
      GenericCloudUser = arrabito
      user_data_commands = vm-bootstrap
      user_data_commands += vm-bootstrap-functions
      user_data_commands += vm-pilot
      user_data_commands += vm-monitor-agent
      user_data_commands += pilotCommands.py
      user_data_commands += pilotTools.py
      user_data_commands += dirac-install.py
      user_data_commands += power.sh
      user_data_commands += parse-jobagent-log
      user_data_commands += dirac-pilot.py
      user_data_commands += save-payload-logs
      # url from which command scripts are downloaded. Usually the url of the web␣
↪server
      user_data_commands_base_url = http://cta-dirac.in2p3.fr/DIRAC/defaults
      Project = CTA
      Version = v1r40
    }
  }
}
```

- CS Registry section

  The host where VMDIRAC is installed and the certificate of which is used for the VMs, it should have these 2 properties set (as in the example below):

  - Properties = GenericPilot (needed to make pilots running on the VM matching jobs in the TaskQueue)
  - Properties = VmRpcOperation (needed by the VirtualMachineMonitorAgent running on the VM to be authorized to send Heartbeats to the VirtualMachineManager service)

```
Registry
{
  Hosts
  {
    dcta-agents01.pic.es
    {
      DN = /DC=org/DC=terena/DC=tcs/C=ES/ST=Barcelona/L=Bellaterra/
↪O=Institut de Fisica dAltes Energies/CN=dcta-agents01.pic.es
```

(continues on next page)

(continued from previous page)

```
      CA = /C=NL/ST=Noord-Holland/L=Amsterdam/O=TERENA/CN=TERENA eScience␣
↪SSL CA 3
      Properties = FullDelegation
      Properties += CSAdministrator
      Properties += ProxyManagement
      Properties += SiteManager
      Properties += Operator
      Properties += JobAdministrator
      Properties += CSAdministrator
      Properties += TrustedHost
      Properties += GenericPilot
      Properties += VmRpcOperation
    }
  }
}
```

2.4.3 Install VMDIRAC WebApp

- On the DIRAC web server install VMDIRAC WebApp as a normal extension. In the server local configuration you should set the following option in the LocalInstallation and DIRAC sections:

```
Extensions = VMDIRAC
```

- Create sym links for the bootstrap scripts

```
$ ll /opt/dirac/webRoot/www/defaults/bootstrap
total 0
lrwxrwxrwx 1 dirac dirac 50 Feb 21 08:46 dirac-install.py -> /opt/dirac/
↪pro/DIRAC/Core/scripts/dirac-install.py
lrwxrwxrwx 1 dirac dirac 71 Feb 21 08:49 dirac-pilot.py -> /opt/dirac/pro/
↪DIRAC/WorkloadManagementSystem/PilotAgent/dirac-pilot.py
lrwxrwxrwx 1 dirac dirac 76 Feb 21 08:50 parse-jobagent-log -> /opt/dirac/
↪pro/VMDIRAC/WorkloadManagementSystem/Bootstrap/parse-jobagent-log
lrwxrwxrwx 1 dirac dirac 73 Feb 21 08:51 pilotCommands.py -> /opt/dirac/
↪pro/DIRAC/WorkloadManagementSystem/PilotAgent/pilotCommands.py
lrwxrwxrwx 1 dirac dirac 70 Feb 21 08:51 pilotTools.py -> /opt/dirac/pro/
↪DIRAC/WorkloadManagementSystem/PilotAgent/pilotTools.py
lrwxrwxrwx 1 dirac dirac 66 Feb 21 08:52 power.sh -> /opt/dirac/pro/
↪VMDIRAC/WorkloadManagementSystem/Bootstrap/power.sh
lrwxrwxrwx 1 dirac dirac 75 Feb 21 08:52 save-payload-logs -> /opt/dirac/
↪pro/VMDIRAC/WorkloadManagementSystem/Bootstrap/save-payload-logs
lrwxrwxrwx 1 dirac dirac 70 Feb 21 11:47 vm-bootstrap -> /opt/dirac/pro/
↪VMDIRAC/WorkloadManagementSystem/Bootstrap/vm-bootstrap
lrwxrwxrwx 1 dirac dirac 80 Feb 21 08:52 vm-bootstrap-functions -> /opt/
↪dirac/pro/VMDIRAC/WorkloadManagementSystem/Bootstrap/vm-bootstrap-
↪functions
lrwxrwxrwx 1 dirac dirac 74 Feb 21 08:53 vm-monitor-agent -> /opt/dirac/
↪pro/VMDIRAC/WorkloadManagementSystem/Bootstrap/vm-monitor-agent
lrwxrwxrwx 1 dirac dirac 66 Feb 21 08:53 vm-pilot -> /opt/dirac/pro/
↪VMDIRAC/WorkloadManagementSystem/Bootstrap/vm-pilot
```

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 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
LHCbWeb 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

   System                         Name                            Type          Setup
↪Installed Runit    Uptime    PID
-------------------------------------------------------------------------------------------
↪-------------------------------
ResourceStatus                 ResourceStatus                service         SetUp
↪Installed Down    2532910        0
WorkloadManagement             SandboxStore                  service         SetUp
↪Installed Run        8390    20510
WorkloadManagement             JobMonitoring                 service         SetUp
↪Installed Run        8390    20494
...
```

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': {'Configuration': ['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:

```
mardirac1.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:

```
mardirac1.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
```

It is possible to check the history of installed components in DIRAC with show installations:

```
[sergiovm.cern.ch]> show installations

  Num          Host                    Name                    Module                 ␣
↪System            Type        Installed on      Uninstalled on
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------
|  1  |  sergiovm.cern.ch  | InstalledComponentsDB  | InstalledComponentsDB  |     ␣
↪Framework      |     DB     | 01-06-2015 16:12 |                  |
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------
|  2  |  sergiovm.cern.ch  |  ComponentMonitoring   |  ComponentMonitoring   |     ␣
↪Framework      |  service   | 01-06-2015 16:12 |                  |
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------
|  3  |  sergiovm.cern.ch  |         Server         |         Server         |     ␣
↪Configuration    |  service   | 01-06-2015 16:12 |                  |
-------------------------------------------------------------------------------------------
↪-------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
|  4  |  sergiovm.cern.ch  |  SystemAdministrator  |  SystemAdministrator  |  ␣
↪Framework      |  service  | 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 Installing and configuring: basic concepts

As seen in *DIRAC Setup Structure*, DIRAC provides you with several *components*, these components are organized in *systems*, and these components can be installed in a *DIRAC Server Installation* using the *System Administrator Console*.

The components don't need to be all resident on the same host, in fact it's common practice to have several hosts for large installations.

Normally, services are always exposed on the same port, which is defined in the configuration for each of them.

As a general rule, services can be duplicated, meaning you can have the same service running on multiple hosts, thus reducing the load. There are only 2 cases of DIRAC services that have a "master/slave" concept, and these are the Configuration Service and the Accounting/DataStore service. The WorkloadManagement/Matcher service should also not be duplicated.

Same can be said for executors: you can have many residing on different hosts.

The same can't be said for agents. Some of them can be duplicated, BUT require a proper configuration, and for this you need to read further in the guide.

2.6.1 Each component has a configuration

When you install a component, it comes with a default configuration. The configuration is available to all the components via the Configuration Service, and its content is exposed by the Configuration Service WebApp in the DIRAC web portal.

The next section, *DIRAC Configuration* keeps a reference of the configuration for each of the components. You don't need to read it all now, you just need to know it's there.

2.6.2 What to install

It depends!

Some components will be needed, whatever you do, e.g. as it should be clear already, you will need always the Configuration Service.

And almost certainly, a large part of what is part of DIRAC framework (the FrameworkSystem) is needed.

Then, it depends from what you want to do. So, if you just want to run some jobs, you'd need to install WorkloadManagementSystem components. If you need to do something else... then, again, it depends.

You need to keep reading.

2.7 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.

2.7.1 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.

2.7.2 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.

Name	Description	Posible values	Example
Extensions	Define which extensions are going to be used in the server	lhcb, eela	Extensions = lhcb
VirtualOrganization	This option define the default virtual organization	String	VirtualOrganization = defaultVO

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 upated by the Master Configuration Server to reflect the current situation of the system.

Name	Description	Example
AutoPublish		AutoPublish = yes
EnableAutoMerge	Allows Auto Merge. Takes a boolean value.	EnableAutoMerge = yes
MasterServer	Define the primary master server.	MasterServer = dips://cclcgvmli09.in2p3.fr:9135/Configuration/Serve
Name	Name of Configuration file	Name = Dirac-Prod
PropagationTime		PropagationTime = 100
RefreshTime	How many time the secondary servers are going to refresh configuration from master. Expressed as Integer and seconds as unit.	RefreshTime = 600
SlavesGraceTime		SlavesGraceTime = 100
Servers	List of Configuration Servers installed. Expressed as URLs using dips as protocol.	Servers = dips://cclcgvmli09.in2p3.fr:9135/Configuration/Serve
Version	CS configuration version used by DIRAC services as indicator when they need to reload the configuration. Expressed using date format.	Version = 2011-02-22 15:17:41.811223

DIRAC / Setups - Subsection

In this subsection all the installed Setups are defined.

Name	Description	Example
Accounting	Describe the instance to be used for this setup	Accounting = Production
Configuration	Describe the instance to be used for this setup	Configuration = Production
DataManagement	Describe the instance to be used for this setup	DataManagement = Production
Framework	Describe the instance to be used for this setup	Framework = Production
RequestManagement	Describe the instance to be used for this setup	RequestManagement = Production
StorageManagement	Describe the instance to be used for this setup	StorageManagement = Production
WorkloadManagement	Describe the instance to be used for this setup	WorkloadManagement = Production

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.

Name	Description	Example
CertFile	Directory where host certificate is located in the server.	CertFile = /opt/dirac/etc/grid-security/hostcert.pem
KeyFile	Directory where host key is located in the server.	KeyFile = /opt/dirac/etc/grid-security/hostcert.pem
SkipCAChecks	Boolean value this attribute allows to express if the CA certificates are or not be checked.	SkipCAChecks = No
UseServerCertificate	Use server certificate, expressed as boolean.	UseServerCertificate = yes

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.

Name	Description	Example
<SystemName>	This attribute define the e-mail of the person in charge of the system	Production = hamar@cppm.in2p3.fr Logging = hamar@cppm.in2p3.fr

Operations / DataManagement

- IgnoreMissingInFC (False): when removing a file/replica, trigger an error if the file is not on the SE
- UseCatalogPFN (True): when getting replicas with the DataManager, use the url stored in the catalog. If False, recalculate it
- SEsUsedForFailover ([]): SEs or SEGroups to be used as failover storages
- SEsNotToBeUsedForJobs ([]): SEs or SEGroups not to be used as input source for jobs
- SEsUsedForArchive ([]): SEs ir SEGroups to be used as Archive
- ForceSingleSitePerSE (True): return an error if an SE is associated to more than 1 site

- FTSVersion (FTS2): version of FTS to use. Possibilities: FTS3 or FTS2 (deprecated)
- FTSPlacement section:
  - FTS2 section: deprecated
  - FTS3 section:
    - ServerPolicy (Random): policy to choose between FTS3 servers (Random, Sequence, Failover)

Operations / InputDataPolicy - Subsection

In this subsection the Data Policy mechanism for input files used in the JobWrapper are defined.

Name	Description	Example
Default	Policy to be used to download input data files	Default = DIRAC.WorkloadManagementSystem.Client.DownloadInputData

Operations / JobDescription - Subsection

JobDescription subsection describes allowed options in submitted payload (needs further documentation of supported fields).

Name	Description	Example
AllowedJobTypes	List of users jobs accepted by the server	AllowedJobTypes = MPI AllowedJobTypes += User AllowedJobTypes += Test

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:

Parameter	Description	Default value
taskQueueCPUTimeIntervals	Possible cpu time values that the task queues can have.	360, 1800, 3600, 21600, 43200, 86400, 172800, 259200, 345600, 518400, 691200, 864000, 1080000
EnableSharesCorrection	Enable automatic correction of the priorities assigned to each task queue based on previous history	False
CheckJobLimits	Limit the amount of jobs running at sites based on their attributes	False
CheckMatchingDelay	Delay running a job at a site if another job has started recently and the conditions are met	False

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 that 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
```

(continues on next page)

(continued from previous page)

```
        {
          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
  {
    DIRAC.Somewhere.co
    {
      JobType
      {
        MonteCarlo = 10
      }
    }
  }
}
```

Transactional bulk job submission

When submitting parametric jobs (bulk submission), the job description contains a recipe to generate actual jobs per parameter value according to a formulae in the description. The jobs are generated by default synchronously in the call to the DIRAC WMS JobManager service. However, there is a risk that in case of an error jobs are partially generated without the client knowing it. To avoid this risk, an additional logic to ensure that no unwanted jobs are left in the system has been added together with DIRAC v6r20.

Pilot version

The */Operations/<vo>/<setup>/Pilot* section define What version of DIRAC will be used to submit pilot jobs to the resources.

Parameter	Description	Default value
Version	What project version will be used	Version with which the component that submits pilot jobs is installed
LCGBundleVersion	which lcgBundle version to install with the pilot. Be careful: if defined, this version will overwrite any possible version defined in the releases.cfg file	None
Project	What installation project will be used when submitting pilot jobs to the resources	DIRAC
CheckVersion	Check if the version used by pilot jobs is the one that they were submitted with	True

Operations / Shifter - Subsection

In this subsection administrators may specify a list of user/group pairs whom proxy certificates will be used for executing actions outside of the DIRAC environment.

Examples include, but are not limited to::

- issuing transfer requests to an external system (e.g. FTS3)
- querying grid databases (e.g. GOC DB)

Name	Description	Example
<ShifterName>	Name of service managers	Admin ProductionManager DataManager MonteCarloGeneration DataProcessing
<ShifterName>/User	DIRAC user name	User = vhamar
<ShifterName/Group	DIRAC user group	Group = dirac_admin

Running agents can use these "shifters" for executing the examples above: agents requiring to act with a credential can specify the option **shifterProxy**, or using a certain default, like "DataManager".

In general, to force any Agent to execute using a "shifter" credential, instead of the certificate of the server it is only necessary to add a valid **shifterProxy** option in its configuration (in the /Systems section).

Operations / VOs - Subsections

<VO_NAME> subsections allows to define pilot jobs versions for each setup defined for each VO supported by the server.

Name	Description	Example
<VO_NAME>	Subsection: Virtual organization name	vo.formation.idgrilles.fr
<VO_NAME>/<SETUP_NAME>/	Subsection: VO Setup name	Dirac-Production
<VO_NAME>/<SETUP_NAME>/Version/	Subsection: Version (Name fixed)	Version
<VO_NAME>/<SETUP_NAME>/Version/PilotVersion	DIRAC version to be installed for the pilots in the WNs	PilotVersion = v6r0-pre7

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.

:: from DIRAC.ConfigurationSystem.Client.Helpers.Operations import Operations op = Operations() op.getValue('VersionPilotVersion', '')

Operations / Transformations - Subsection

Operations / Transformations / Options

Operations / TransformationPlugins / Options

List of options

- *MainServers*: List of server names (no protocol, no port) to be used as MainServer

Registry - Section

This section allows to register users, hosts and groups in DIRAC way. Also some attributes applicable for all the configuration are defined.

Name	Description	Example
DefaultGroup	Default user group to be used	DefaultGroup = user
QuarantineGroup	Querantine user group is usually to be used in case you want to set users in groups by hand as a "punishment" for a certain period of time	QuarantineGroup = lowPriority_user
DefaultProxyTime	Default proxy time expressed in seconds	DefaultProxyTime = 4000

Registry / Groups - Subsections

This subsection is used to describe DIRAC groups registered in the server.

Name	Description	Example
<GROUP_NAME>	Subsection, represents the name of the group	dirac_user
<GROUP_NAME>/Users	DIRAC users logins than belongs to the group	Users = atsareg Users += msapunov
<GROUP_NAME>/Properties	Properties of the group, this will change the permissions of the group.	Properties = NormalUser
<GROUP_NAME>/VOMSRole	Role of the users in the VO	VOMSRole = /biomed
<GROUP_NAME>/VOMSVO	Virtual organization associated with the group	VOMSVO = biomed
JobShare	Just for normal users	JobShare = 200
AutoUploadProxy	Controls automatic Proxy upload by dirac-proxy-init	AutoUploadProxy = True
AutoUploadPilotProxy	Controls automatic Proxy upload by dirac-proxy-init for Pilot groups	AutoUploadPilotProxy = True
AutoAddVOMS	Controls automatic addition of VOMS extension by dirac-proxy-init	AutoAddVOMS = True

- 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.

A subsection called as DIRAC host name must be created and inside of this the following attributes must be included:

Name	Description	Example
<DIRAC_HOST_NAME>	Subsection DIRAC host name	host-dirac.in2p3.fr
<DIRAC_HOST_NAME>/DN	Host distinguish name obtained from host certificate	DN = /O=GRID-FR/C=FR/O=CNRS/OU=CC-IN2P3/CN=dirac.in2p3.fr
<DIRAC_HOST_NAME>/Properties	Properties associated with the host	Properties = JobAdministrator Properties += FullDelegation Properties += Operator Properties += CSAdministrator Properties += ProductionManagement Properties += AlarmsManagement Properties += ProxyManagement Properties += TrustedHost

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:

Name	Description	Example
<DIRAC_USER_NAME>/DN	Distinguish name obtained from user certificate (Mandatory)	DN = /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
<DIRAC_USER_NAME>/CA	Canonical name of certification authority who sign the certificate.	CN = /C=FR/O=CNRS/CN=GRID2-FR
<DIRAC_USER_NAME>/Email	User email (Mandatory)	Email = atsareg@in2p3.fr
<DIRAC_USER_NAME>/mobile	Cellular phone number	mobile = +030621555555
<DIRAC_USER_NAME>/Quota	Quota assigned to the user. Expressed in MBs.	Quota = 300

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 manadatory 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.

Name	Description	Example
<VO_NAME>/VOAdmin	VO administrator user name	VOAdmin = joel
<VO_NAME>/VOMSName	VOMS VO name	VOMSName = lhcb
<VO_NAME>/SubmitPools	Default Submit Pools for the users belonging to the VO	SubmitPools = lhcbPool

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.

Name	Description	Example
<VOMS_SERVER>/DN	DN of the VOMS server certificate	DN = /O=GRID-FR/C=FR/O=CNRS/OU=CC-IN2P3/CN=cclcgvomsli01.in2p3.fr
<VOMS_SERVER>/Port	The VOMS server port	Port = 15003
<VOMS_SERVER>/CA	CA that issued the VOMS server certificate	CA = /C=FR/O=CNRS/CN=GRID2-FR

VOMSServices subsection

This subsection contains URLs to obtain specific VOMS informations.

Name	Description	Example
VOMSAttributes	URL to get VOMS attributes	VOMSAttributes = https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAttributes
VOMSAdmin	URL to get VOMS administrator info	VOMSAdmin = https://voms2.cern.ch:8443/voms/lhcb/services/VOMSAdmin
VOMSCompatibility	URL to get VOMS compatibility info	VOMSCompatibility = https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCompatibility
VOMSCertificates	URL to get VOMS certificate info	VOMSCertificates = https://voms2.cern.ch:8443/voms/lhcb/services/VOMSCertificates

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.

Name	Description	Example
FileCatalog	Subsection used to configure DIRAC File catalog	FileCatalog
FileCatalog/AccessType	Access type allowed to the particular catalog	AccessType = Read-Write
FileCatalog/Status	To define the catalog as active or inactive	Status = Active
FileCatalog/MetaCatalog	If the Catalog is a MetaDataCatalog	MetaCatalog = True

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.

Name	Description	Example
<DIRAC_SITE_NAME>	Subsection named with the site name	LCG.CPPM.fr
<DIRAC_SITE_NAME>/Name	Site name gave by the site administrator e.g.: the name of the site in GOCDB (optional)	Name = in2p3
<DIRAC_SITE_NAME>/CE	List of CEs using CE FQN These CEs are updated by the BDII2CSAgent in the CEs section	CE = ce01.in2p3.fr CE += ce02.in2p3.fr
*<DIRAC_SITE_NAME>/MoUTierLevel	Tier Level (optional)	MoUTierLevel = 1
*<DIRAC_SITE_NAME>/CEs	Subsection used to describe each CE available	CEs
*<DIRAC_SITE_NAME>/Coordinates	Site geographical coordinates (optional)	Coordinates = -8.637979:41.152461
*<DIRAC_SITE_NAME>/Mail	Mail address site responsable (optional)	Mail = atsareg@in2p3.fr
*<DIRAC_SITE_NAME>/SE	Closest SE respect to the CE (optional)	SE = se01.in2p3.fr

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.

Name	Description	Example
<CE_NAME>	Subsection named as the CE fully qualified name	ce01.in2p3.fr
<CE_NAME>/architecture	CE architecture	architecture = x86_64
<CE_NAME>/CEType	Type of CE, can take values as LCG or CREAM	CEType = LCG
<CE_NAME>/OS	CE operating system in a DIRAC format	OS = ScientificLinux_Boron_5.3
<CE_NAME>/Pilot	Boolean attributes than indicates if the site accept pilots	Pilot = True
<CE_NAME>/SubmissionMode	If the CE is a cream CE the mode of submission	SubmissionMode = Direct
<CE_NAME>/wnTmpDir	Worker node temporal directory	wnTmpDir = /tmp
<CE_NAME>/MaxProcessors	Maximum number of available processors on worker nodes	MaxProcessors = 12
<CE_NAME>/WholeNode	CE allows *whole node* jobs	WholeNode = True
<CE_NAME>/Tag	List of tags specific for the CE	Tag = GPU,96RAM
<CE_NAME>/RequiredTag	List of required tags that a job to be eligible must have	RequiredTag = GPU,96RAM
<CE_NAME>/Queues	Subsection. Queues available for this VO in the CE	Queues
<CE_NAME>/Queues/<QUEUE_NAME>	Name of the queue exactly how is published	jobmanager-pbs-formation
<CE_NAME>/Queues/<QUEUE_NAME>/CEQueueName	Name of the queue in the corresponding CE if not the same as the name of the queue section	CEQueueName = pbs-grid
<CE_NAME>/Queues/<QUEUE_NAME>/maxCPUTime	Maximum CPU time allowed to jobs to run in the queue	maxCPUTime = 1440
<CE_NAME>/Queues/<QUEUE_NAME>/MaxTotalJobs	In the CE 7 in a CREAM CE the maximum number of jobs in all the status	MaxTotalJobs =200
<CE_NAME>/Queues/<QUEUE_NAME>/MaxWaitingJobs	In the CE 7 in a CREAM CE the maximum number of jobs in waiting status	MaxWaitingJobs = 70
<CE_NAME>/Queues/<QUEUE_NAME>/OutputURL	In the CE 7 in a CREAM CE the URL where to find the outputs	OutputURL = gsiftp://localhost
<CE_NAME>/Queues/<QUEUE_NAME>/SI00	CE SI00 Scaling Reference	SI00 = 2130
<CE_NAME>/Queues/<QUEUE_NAME>/MaxProcessors	Overwrites *<CE_NAME>/MaxProcessors* at queue level	MaxProcessors = 12
<CE_NAME>/Queues/<QUEUE_NAME>/WholeNode	Overwrites *<CE_NAME>/WholeNode* at queue level	WholeNode = True
<CE_NAME>/Queues/<QUEUE_NAME>/Tag	List of tags specific for the Queue	Tag = GPU,96RAM
<CE_NAME>/Queues/<QUEUE_NAME>/RequiredTag	List of required tags that a job to be eligible must have	RequiredTag = GPU,96RAM

An example for this session follows:

```
Sites
{
  LCG
  {
```

(continues on next page)

(continued from previous page)

```
    LCG.CERN.cern
    {
      SE = CERN-RAW
      SE += CERN-RDST
      SE += CERN-USER
      CE = ce503.cern.ch
      CE += ce504.cern.ch
      Name = CERN-PROD
      Coordinates = 06.0458:46.2325
      Mail = grid-cern-prod-admins@cern.ch
      MoUTierLevel = 0
      Description = CERN European Organization for Nuclear Research
      CEs
      {
        ce503.cern.ch
        {
          wnTmpDir = .
          architecture = x86_64
          OS = ScientificCERNSLC_Carbon_6.4
          SI00 = 0
          Pilot = False
          CEType = HTCondorCE
          SubmissionMode = Direct
          Queues
          {
            ce503.cern.ch-condor
            {
              VO = lhcb
              VO += LHCb
              SI00 = 3100
              MaxTotalJobs = 5000
              MaxWaitingJobs = 200
              maxCPUTime = 7776
            }
          }
          VO = lhcb
          MaxRAM = 0
          UseLocalSchedd = False
          DaysToKeepLogs = 1
        }
        ce504.cern.ch
        {
          wnTmpDir = .
          architecture = x86_64
          OS = ScientificCERNSLC_Carbon_6.4
          SI00 = 0
          Pilot = False
          CEType = HTCondorCE
          SubmissionMode = Direct
          Queues
          {
            ce504.cern.ch-condor
            {
              VO = lhcb
              VO += LHCb
              SI00 = 3100
              MaxTotalJobs = 5000
```

(continues on next page)

(continued from previous page)

```
            MaxWaitingJobs = 200
            maxCPUTime = 7776
          }
        }
      }
    }
  }
}
DIRAC
{
  DIRAC.HLTFarm.lhcb
  {
    Name = LHCb-HLTFARM
    CE = OnlineCE.lhcb
    CEs
    {
      OnlineCE.lhcb
      {
        CEType = CREAM
        Queues
        {
          OnlineQueue
          {
            maxCPUTime = 2880
          }
        }
      }
    }
    AssociatedSEs
    {
      Tier1-RDST = CERN-RDST
      Tier1_MC-DST = CERN_MC-DST-EOS
      Tier1-Buffer = CERN-BUFFER
      Tier1-Failover = CERN-EOS-FAILOVER
      Tier1-BUFFER = CERN-BUFFER
      Tier1-USER = CERN-USER
      SE-USER = CERN-USER
    }
  }
}
VAC
{
  VAC.Manchester.uk
  {
    Name = UKI-NORTHGRID-MAN-HEP
    CE = vac01.blackett.manchester.ac.uk
    CE += vac02.blackett.manchester.ac.uk
    Coordinates = -2.2302:53.4669
    Mail = ops@NOSPAMtier2.hep.manchester.ac.uk
    CEs
    {
      vac01.blackett.manchester.ac.uk
      {
        CEType = Vac
        architecture = x86_64
        OS = ScientificSL_Carbon_6.4
        wnTmpDir = /scratch
```

(continues on next page)

(continued from previous page)

```
          SI00 = 2200
          MaxCPUTime = 1000
          Queues
          {
            default
            {
              maxCPUTime = 1000
            }
          }
        }
        vac02.blackett.manchester.ac.uk
        {
          CEType = Vac
          architecture = x86_64
          OS = ScientificSL_Carbon_6.4
          wnTmpDir = /scratch
          SI00 = 2200
          MaxCPUTime = 1000
          Queues
          {
            default
            {
              maxCPUTime = 1000
            }
          }
        }
      }
    }
  }
}
```

Resources / StorageElements and StorageElementBases- Subsections

All the storages elements available for the users are described in these subsections. Base Storage Elements, corresponding to abstract Storage Element, must be defined in the Resources/StorageElementBases section while other Storage Elements, like inherited and simple Storage Elements, must be configured in the Resources/StorageElement section. This information will be moved bellow the Sites section.

Name	Description	Example
DefaultProtocols	Default protocols than can be used to interact with the storage elements.	DefaultProtocols = rfio DefaultProtocols += file DefaultProtocols += root DefaultProtocols += gsiftp
SITE-disk	Subsection. DIRAC name for the storage element	CPPM-disk
SITE-disk/BackendType	Type of storage element. Possible values are: dmp, DISET, dCache, Storm	BackendType = dpm
SITE-disk/ReadAccess	Allow read access Possible values are: Active, InActive	ReadAccess = Active
SITE-disk/WriteAccess	Allow write access Possible values are: Active, InActive	WriteAccess = Active
SITE-disk/RemoveAccess	Allow removal of files at this SE Possible values are: Active, InActive	RemoveAccess = Active
SITE-disk/SEType	Type of SE Possible values are: T0D1, T1D0, D1T0	SEType = T0D1
SITE-disk/AccessProtocol.<#>	Subsection. Access protocol number	AccessProtocol.1
SITE-disk/AccessProtocol.<#>/Access	Access type to the resource	Access = Remote
SITE-disk/AccessProtocol.<#>/Host	Storage element fully qualified host-name	Host = se01.in2p3.fr
SITE-disk/AccessProtocol.<#>/Path	Path in the SE just before the VO directory	Path = /dpm/in2p3.fr/home
SITE-disk/AccessProtocol.<#>/Port	Port number to access the data	Port = 8446
SITE-disk/AccessProtocol.<#>/Protocol	Protocol to be used to interact with the SE	Protocol = srm
SITE-disk/AccessProtocol.<#>/PluginName	Protocol name to be used to interact with the SE	PluginName = GFAL2_SRM2
SITE-disk/AccessProtocol.<#>/WSUrl	URL from WebServices	WSUrl = /srm/managerv2?SFN=

Resources / StorageElementGroups - Subsections

All the storages elements groups available for the users are described in this subsection.

Name	Description	Example
SE-USER	Default SEs to be used when uploading output data from Payloads	CERN-USER
Tier1-Failover	Default SEs to be used as failover SEs uploading output data from Payloads. This option is used in the Job Wrapper and, if set, requires the RequestManagementSystem to be installed	CERN-FAILOVER,CNAF-FAILOVER

Resources / Computing

In this section options for ComputingElements can be set

Location for Parameters

Options for computing elements can be set at different levels, from lowest to highest prority

/Resources/Computing/OSCompatibility

This section is used to define a compatibility matrix between dirac platforms (*dirac-platform*) and OS versions.

An example of this session is the following:

```
OSCompatibility
{
  Linux_x86_64_glibc-2.5 = x86_64_CentOS_Carbon_6.6
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Carbon_6.7
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Core_7.4
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Core_7.5
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Final_6.4
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Final_6.7
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Final_6.9
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Final_7.4
  Linux_x86_64_glibc-2.5 += x86_64_CentOS_Final_7.5
  Linux_x86_64_glibc-2.5 += x86_64_RedHatEnterpriseLinuxServer_6.7_Santiago
  Linux_x86_64_glibc-2.5 += x86_64_RedHatEnterpriseLinuxServer_7.2_Maipo
  Linux_x86_64_glibc-2.5 += x86_64_Scientific_6_6.9
  Linux_x86_64_glibc-2.5 += x86_64_Scientific_Carbon_6.8
  Linux_x86_64_glibc-2.5 += x86_64_Scientific_Carbon_6.9
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Boron_6.5
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.3
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.4
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.5
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.6
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.7
  Linux_x86_64_glibc-2.5 += x86_64_ScientificCERNSLC_Carbon_6.9
  Linux_x86_64_glibc-2.5 += x86_64_ScientificLinux-6.9_0_0
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Boron_6.4
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.10
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.3
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.4
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.5
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.6
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.7
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.8
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.9
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6x
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Carbon_6.x
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_Nitrogen_7.4
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_SL_6.4
  Linux_x86_64_glibc-2.5 += x86_64_ScientificSL_SL_6.5
  Linux_x86_64_glibc-2.5 += x86_64_SL_Nitrogen_7.2
}
```

What's on the left is an example of a dirac platform as determined the dirac-platform script (*dirac-platform*). This platform is declared to be compatible with a list of "OS" strings. These strings are identifying the architectures of computing elements. This list of strings can be constructed from the "Architecture" + "OS" fields that can be found in the CEs description in the CS (cs-sites).

This compatibility is, by default, used by the SiteDirector when deciding if to send a pilot or not to a certain CE: the SiteDirector matches "TaskQueues" to Computing Element capabilities.

Other subsections are instead used to describe specific types of computing elements:

/Resources/Computing/CEDefaults For all computing elements

/Resources/Computing/<CEType> For CEs of a given type, e.g., HTCondorCE or ARC

/Resources/Sites/<grid>/<site>/CEs For all CEs at a given site

/Resources/Sites/<grid>/<site>/CEs/<CEName> For the specific CE

Values are overwritten.

General Parameters

These parameters are valid for all types of computing elements

Name	Description	Example
GridEnv	Default environment file sourced before calling grid commands, without extension '.sh'.	/opt/dirac/gridenv (when the file is gridenv.sh)

ARC CE Parameters

Name	Description	Example
XRSLExtraString	Default additional string for ARC submit files	
XRSLMPExtraString	Default additional string for ARC submit files for multi-processor jobs.	
Host	The host for the ARC CE, used to overwrite the ce name	
WorkingDirectory	Directory where the pilot log files are stored locally.	/opt/dirac/pro/runit/WorkloadManagement/SiteDirectorArc

Singularity CE Parameters

Name	Description	Example
ContainerRoot	The root image location for the container to use.	/cvmfs/cernvm-prod.cern.ch/cvm3
ContainerExtraOpts	Extra options for dirac-install within the container.	-u 'http://other.host/instdir' -g 'v13r0'

HTCondorCE Parameters

Options for the HTCondorCEs

Name	Description	Example
ExtraSubmitString	Additional string for the condor submit file. Separate entries with "\n".	request_cpus = 8 \n periodic_remove = ...
WorkingDirectory	Directory where the pilot log files are stored locally. Also temproray files like condor submit files are kept here. This option is only read from the global Resources/Computing/HTCondorCE location.	/opt/dirac/pro/runit/WorkloadManagement
UseLocalSchedd	If True use a local condor schedd to submit jobs, if False submit to remote condor schedd	Default is True
DaysToKeepLogFiles	How many days pilot log files are kept on the disk before they are removed	15

CREAM CE Parameters

Name	Description	Example
ExtraJDLParameters	Additional JDL parameters to submit pilot jobs to CREAm CE. Separate entries with ";".	ExtraJDLParameters = GPUNumber=1; OneMore="value"

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.

Systems / Accounting / <INSTANCE> / Databases - Sub-subsection

Databases used by Accounting System. Note that each database is a separate subsection.

Name	Description	Example
<DATABASE_NAME>	Subsection. Database name	AccountingDB
<DATABASE_NAME>/DBName	Database name	DBName = AccountingDB
<DATABASE_NAME>/Host	Database host server where the DB is located	Host = db01.in2p3.fr
<DATABASE_NAME>/MaxQueueSize	Maximum number of simultaneous queries to the DB per instance of the client	MaxQueueSize = 10

The databases associated with Accounting System are: - AccountingDB

Systems / Accounting / <INSTANCE> / Service - Sub-subsection

All the services have common options to be configured for each one. Those options are presented in the following table:

Name	Description	Example
LogLevel	Level of log verbosity	LogLevel = INFO
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

Accounting system related services are:

Systems / AccountingManagement / <INSTANCE> / Service / DataStore - Sub-subsection

DataStore service is in charge of receiving Accounting data.

No special options must be configured to use this service.

Systems / AccountingManagement / <INSTANCE> / Service / ReportGenerator - Sub-subsection

ReportGenerator service is in charge of producing accounting reports (plots or CSV files).

No special options must be configured.

Systems / Accounting / <INSTANCE> / URLs - Sub-subsection

Accounting Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, value URL using dips protocol	DataStore = dips://dirac.eela.if.ufrj.br:9133/Accounting/DataStore

Services associated with Accounting System:

Service	Port
DataStore	9133
ReportGenerator	9134

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:

Name	Description	Example
HandlerPath	Relative path directory where the service is located	HandlerPath = DIRAC/ConfigurationSystem/Service/ConfigurationHand
Port	Port where the service is responding	Port = 9135
UpdatePilotCStoJSONFile	Optional flag to enable if you want that the configuration on the pilot is dumped in a JSON file and uploaded to a webserver	UpdatePilotCStoJSONFile = True Default is False
pilotFileServer	Web server where to upload the pilot file and its JSON configuration file	pilotFileServer = lbcertifdirac6.cern.ch
pilotRepo	Pointer to git repository of DIRAC pilot	pilotRepo = https://github.com/DIRACGrid/Pilot.git The value above is the default
pilotVORepo	Pointer to git repository of VO DIRAC extension of pilot	pilotVORepo = https://github.com/MyDIRAC/VOPilot.git
pilotScriptsPath	Path to the code, inside the Git repository	pilotScriptsPath = Pilot The value above is the default
pilotScriptsVOPath	Path to the code, inside the Git repository	pilotScriptsVOPath = VOPilot
Authorization	Subsection to configure authorization over the service	Authorization
Authorization/Default	Default authorization	Default = all
Authorization/commitNewData	Define who can commit new configuration	commitNewData = CSAdministrator
Authorization/getVersionContents	Define who can get version contents	getVersionContents = CSAdministrator
Authorization/rollBackToVersion	Define who can roll back the configuration to a previous version	rollBackToVersion = ServiceAdministrator rollBackToVersion += CSAdministrator

Systems / Configuration / <INSTANCE> / Agents - Sub-subsection

In this subsection each agent is described.

Name	Description	Example
Agent	Subsection named as the agent is called.	CE2CSAgent

Common options for all the agents:

Name	Description	Example
LogLevel	Log Level associated to the agent	LogLevel = DEBUG
LogBackends		LogBackends = stdout, server
MaxCycles	Maximum number of cycles made for Agent	MaxCycles = 500
MonitoringEnabled	Indicates if the monitoring of agent is enabled. Boolean values	MonitoringEnabled = True
PollingTime	Each many time a new cycle must start expresed in seconds	PollingTime = 2600
Status	Agent Status, possible values Active or Inactive	Status = Active
DryRun	If True, the agent won't change the CS	DryRun = False
WatchdogTime	If > 0 will kill the agent if the cycle exceeds WatchdogTime in seconds to force a restart of the agent	WatchdogTime = 3600 (default is 0)

Agents associated with Configuration System:

Systems / Configuration / <INSTANCE> / Agents /Bdii2CSAgent - Sub-subsection

Bdii2CSAgent is the agent in charge of updating sites parameters configuration for a specific VO:

- Queries BDII for Computing Elements (CEs) information and update the CS.
- Queries BDII for Storage Elements (SEs) information and update the CS.

The attributes of this agent are shown in the table below:

Name	Description	Example
AlternativeBDIIs	List of alternatives BDIIs	AlternativeBDIIs = bdii01.in2p3.fr
GLUE2URLs	URLs to use for GLUE2 in addition	top-bdii.cern.ch:2170
GLUE2Only	Only search GLUE2, not GLUE1. If true only the URL under *Host* is queried, not those under *GLUE2URLs*	False
Host	Host to query, must include port	lcg-bdii.cern.ch:2170
MailTo	E-mail of the person in charge of update the Sites configuration	MailTo = hamar@cppm.in2p3.fr
MailFrom	E-mail address used to send the information to be updated	MailFrom = dirac@mardirac.in2p3.fr
ProcessCEs	Process Computing Elements	ProcessCEs = True
ProcessSEs	Process Storage Elements	ProcessSEs = True
VirtualOrganization	Name of the VO	VirtualOrganization = vo.formation.idgrilles.fr

Systems / Configuration / <INSTANCE> / Agents /VOMS2CSAgent - Sub-subsection

VOMS2CSAgent queries VOMS servers and updates the users and groups as defined in the Configuration Registry for the given VO and groups in this VO. It performs the following operations:

- Extracts user info from the VOMS server using its REST interface
- Finds user DN's not yet registered in the DIRAC Registry
- For each new DN it constructs a DIRAC login name by a best guess or using the nickname VOMS attribute
- Registers new users to the DIRAC Registry including group membership
- Updates information for already registered users
- Sends report for performed operation to the VO administrator

The agent is performing its operations with credentials of the VO administrator as defined in the /Registry/VO/<VO_name> configuration section.

The configuration options of this agent are shown in the table below:

Name	Description	Example
VO	List of VO names	VO = biomed, eiscat.se, compchem
MailTo	E-mail of the person in charge of update the Sites configuration	MailTo = hamar@cppm.in2p3.fr
MailFrom	E-mail address used to send the information to be updated	MailFrom = dirac@mardirac.in2p3.fr
AutoAddUsers	If users will be added automatically	AutoAddUsers = True
AutoModifyUsers	If users will be modified automatically	AutoModifyUsers = True
AutoDeleteUsers	Users no more registered in VOMS are automatically deleted from DIRAC	AutoDeleteUsers = False
DetailedReport	Detailed report on users per group sent to the VO administrator	DetailedReport = True
MakeHomeDirectory	Automatically create user home directory in the File Catalog	MakeHomeDirectory = False

Remark: options *AutoAddUsers*, *AutoModifyUsers*, *AutoDeleteUsers* can be overridden by the corresponding options defined in the /Registry/VO/<VO_name> configuration section.

Systems / Configuration / <INSTANCE> / Agents /GOCDB2CSAgent - Sub-subsection

Synchronizes information between GOCDB and DIRAC configuration System (CS)

The attributes of this agent are showed in the table below:

Name	Description	Example
UpdatePerfSONARS	Sync perfSONAR end points to CS	UpdatePerfSONARS = True

Systems / Configuration / <INSTANCE> / URLs - Sub-subsection

Configuration Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, the value is a URL using dips protocol	dips://guaivira.lsd.ufcg.edu.br:9135/Configuration/Server

Services associated with Configuration System:

Service	Port
Server	9135

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.

Name	Description	Example
<DATABASE_NAME>	Subsection. Database name	FileCatalogDB
<DATABASE_NAME>/DBName	Database name	DBName = FileCatalogDB
<DATABASE_NAME>/Host	Database host server where the DB is located	Host = db01.in2p3.fr
<DATABASE_NAME>/MaxQueueSize	Maximum number of simultaneous queries to the DB per instance of the client	MaxQueueSize = 10

The databases associated with DataManagement System are: - FileCatalogDB - DataIntegrityDB - DataLoggingDB

Systems / DataManagement / <INSTANCE> / Service - Sub-subsection

All the services have common options to be configured for each one. Those options are presented in the following table:

Name	Description	Example
LogLevel	Level of log	LogLevel = INFO
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

DataManagement services are:

Systems / DataManagement / <INSTANCE> / Service / FileCatalog - Sub-subsection

FileCatalogHandler is a simple Replica and Metadata Catalog service. Special options are required to configure this service, showed in the next table:

Name	Description	Example
DefaultUmask	Default UMASK	DefaultUmask = 509
DirectoryManager	Directory manager	DirectoryManager = DirectoryLevelTree
FileManager	File Manager	FileManager = FileManager
GlobalReadAccess	Boolean Global Read Access	GlobalReadAccess = True
LFNPFNConvention	Boolean indicating to use LFN PFN convention	LFNPFNConvention = True
SecurityManager	Security manager to be used	SecurityManager = NoSecurityManager
SEManager	Storage Element manager	SEManager = SEManagerDB
ResolvePFN	Boolean indicating if resolve PFN must be done	ResolvePFN = True
VisibleStatus	Visible Status	VisibleStatus = AprioriGood
UniqueGUID	Use a unique GUID	UniqueGUID = False
UserGroupManager	User group manager	UserGroupManager = UserAndGroupManagerDB

Systems / DataManagement / <INSTANCE> / Service / StorageElement - Sub-subsection

StorageElementHandler is the implementation of a simple StorageElement service in the DISET framework

Name	Description	Example
BasePath	Directory path used as base for DIRAC SE	BasePath = /opt/dirac/data

Systems / DataManagement / <INSTANCE> / Service / StorageElementProxy - Sub-subsection

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.

Name	Description	Example
BasePath	Temporary directory use for transfers	BasePath = storageElement

Systems / DataManagement / <INSTANCE> / URLs - Sub-subsection

DataManagement Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, value URL using dips protocol	dips://dirac.eela.if.ufrj.br:9197/DataManagement/FileCatalog

Services associated with DataManagement System:

Service	Port
FileCatalog	9197
StorageElement	9148
StorageElementProxy	9139
TransferDBMonitoring	9191

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.

Name	Description	Example
<DATABASE_NAME>	Subsection. Database name	JobDB
<DATABASE_NAME>/DBName	Database name	DBName = JobDB
<DATABASE_NAME>/Host	Database host server where the DB is located	Host = db01.in2p3.fr
<DATABASE_NAME>/MaxQueueSize	Maximum number of simultaneous queries to the DB per instance of the client	MaxQueueSize = 10

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:

Name	Description	Example
LogLevel	Level of log verbosity	LogLevel = INFO
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

Some extra options are required to configure this service:

Name	Description	Example
MaxParametricJobs	Max number of jobs that can be submitted at once using parametric jobs mechanism, default = 20	MaxParametricJobs = 100

Systems / WorkloadManagement / <INSTANCE> / Service / JobMonitoring - Sub-subsection

JobMonitoringHandler 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

JobStateUpdateHandler 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:

Name	Description	Example
SSLSessionTime	Define duration time of ssl connections Expressed in seconds	SSLSessionTime = 86400

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:

Name	Description	Example
getActiveTaskQueues	Define DIRAC group allowed to get the active task queues in the system	getActiveTaskQueues = dirac_admin

Systems / WorkloadManagement / <INSTANCE> / Service / SandboxStore - Sub-subsection

SandboxHandler is the implementation of the Sandbox service in the DISET framework

Some extra options are required to configure this service:

Name	Description	Example
Backend		Backend = local
BasePath	Base path where the files are stored task queues in the system	BasePath = /opt/dirac/storage/sandboxes
DelayedExternalDeletion	Boolean used to define if the external deletion must be done	DelayedExternalDeletion = True
MaxSandboxSize	Maximum size of sanbox files expressed in MB	MaxSandboxSize = 10
SandboxPrefix	Path prefix where sandbox are stored	SandboxPrefix = Sandbox

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-subsection

In this subsection each agent is described.

Name	Description	Example
Agent	Subsection named as the agent is called.	InputDataAgent

Common options for all the agents are described in the table below:

Name	Description	Example
LogLevel	Log Level associated to the agent	LogLevel = DEBUG
LogBackends		LogBackends = stdout, server
MaxCycles	Maximum number of cycles made for Agent	MaxCycles = 500
MonitoringEnabled	Indicates if the monitoring of agent is enabled. Boolean values	MonitoringEnabled = True
PollingTime	Each many time a new cycle must start expressed in seconds	PollingTime = 2600
Status	Agent Status, possible values Active or Inactive	Status = Active

Agents associated with Configuration System:

Systems / WorkloadManagement / <INSTANCE> / Agents / PilotStatusAgent - Sub-subsection

The Pilot Status Agent updates the status of the pilot jobs if the PilotAgents database.

Special attributes for this agent are:

Name	Description	Example
GridEnv	Path where is located the file to load Grid Environment Variables	GridEnv = /usr/profile.d/grid-env
PilotAccountingEnabled	Boolean type attribute than allows to specify if accounting is enabled	PilotAccountingEnabled = Yes
PilotStalledDays	Number of days without response of a pilot before be declared as Stalled	PilotStalledDays = 3

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

Name	Description	Example
FailedTime-Hours	How much time in hours pass before a stalled job is declared as failed Note: Not actually in hours	FailedTimeHours = 6
StalledTime-Hours	How much time in hours pass before running job is declared as stalled Note: Not actually in hours	StalledTimeHours = 2
MatchedTime	Age in seconds until matched jobs are rescheduled	MatchedTime = 7200
Rescheduled-Time	Age in seconds until rescheduled jobs are rescheduled	RescheduledTime = 600
Completed-Time	Age in seconds until completed jobs are declared failed, unless their minor status is "Pending Requests"	CompletedTime = 86400
StalledJob-sTolerantSites	List of site for which the StalledJobAgent will increase the tolerance for stalled jobs	StalledJobsTolerantSites = siteA.cern.ch, siteB.cern.ch
StalledJob-sToleranceTime	Time in seconds to be added to the StalledTimeHours in order to increase the time tolerance for stalled jobs.	StalledJobsToleranceTime = 3000

Systems / WorkloadManagement / <INSTANCE> / Agents / StatesAccountingAgent - Sub-subsection

StatesAccountingAgent 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> / Executors - Sub-subsection

In this subsection each executor is described.

Name	Description	Example
Executor	Subsection named as the Executor is called.	InputData

Common options for all the executors are described in the table below:

Name	Description	Example
LogLevel	Log Level associated to the executor	LogLevel = DEBUG
LogBackends		LogBackends = stdout, server
Status	????Executor Status, possible values Active or Inactive	Status = Active

Executors associated with Configuration System:

Systems / WorkloadManagement / <INSTANCE> / Executors / InputData - Sub-subsection

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.

Name	Description	Example
FailedJobStatus	MinorStatus if Executor fails the job	FailedJobStatus = "Input Data Not Available"
CheckFileMeta-data	Boolean, check file metadata; will ignore Failover SE files	CheckFileMetadata = True

Systems / WorkloadManagement / <INSTANCE> / Executors / JobPath - Sub-subsection

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.

Name	Description	Example
BasePath	Path for jobs through the executors	BasePath = JobPath, JobSanity
VOPlugin	Name of a VO Plugin???	VOPlugin = ''
InputData	Name of the InputData instance	InputData = InputData
EndPath	Last executor for a job	EndPath = JobScheduling

Systems / WorkloadManagement / <INSTANCE> / Executors / JobSanity - Sub-subsection

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)

Name	Description	Example
InputDataCheck	Boolean, check if input data is prop- erly formated, default=True	InputDataCheck = True
MaxInputDataPerJob	Integer, Maximum number of input lfns	MaxInputDataPerJob=100
InputSandboxCheck	Check for input sandbox files	InputSandboxCheck = True
OutputDataCheck	Check if output data exists Not Implemented	OutputDataCheck = True

Systems / WorkloadManagement / <INSTANCE> / Executors / JobScheduling - Sub-subsection

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.

Name	Description	Example
RescheduleDelays	How long to hold job after rescheduling	RescheduleDelays=60, 180, 300, 600
ExcludedOnHold-JobTypes	List of job types to exclude from holding after rescheduling	
InputDataAgent	Name of the InputData executor instance	InputDataAgent = InputData
RestrictDataStage	Are users restricted from staging	RestrictDataStage = False
HoldTime	How long jobs are held for	HoldTime = 300
StagingStatus	Status when staging	StagingStatus = Staging
StagingMinorStatus	Minor status when staging	StagingMinorStatus = "Request To Be Sent"
AllowInvalidSites	If set to False, jobs will be held if any of the Sites specified are invalid.	AllowInvalidSites = False (default value is True)
CheckOnlyTapeSEs	If set to False, the optimizer will check the presence of all replicas	CheckOnlyTapeSEs = False (default value is True)
CheckPlatform	If set to True, the optimizer will verify the job JDL Platform setting.	CheckPlatform = True (default value is False)

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:

Name	Description	Example
BufferLimit	Size limit of the buffer used for transmission between the WN and DIRAC server	BufferLimit = 10485760
CleanUpFlag	Boolean	CleanUpFlag = True
DefaultCatalog	Default catalog where must be registered the output files if this is not defined by the user FileCatalog define DIRAC file catalog	DefaultCatalog = FileCatalog
DefaultCPUTime	Default CPUTime expressed in seconds	DefaultCPUTime = 600
DefaultErrorFile	Name of default error file	DefaultErrorFile = std.err
DefaultOutputFile	Name of default output file	DefaultOutputFile = std.out
DefaultOutputSE	Default output storage element	DefaultOutputSE = IN2P3-disk
MaxJobPeekLines	Maximum number of output job lines showed	MaxJobPeekLines = 20
OutputSandboxLimit	Limit of sandbox output expressed in MB	OutputSandboxLimit = 10

Systems / WorkloadManagement / <INSTANCE> / URLs - Sub-subsection

WorkloadManagement Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, value URL using dips protocol	JobManager = dips://dirac.eela.if.ufrj.br:9132/WorkloadManagement/Job

Services associated with WorkloadManagement System:

Service	Port
JobManager	9132
JobMonitoring	9130
JobStateUpdate	9136
Matcher	9170
MPIService	9171
SandboxStore	9196
WMSAdministrator	9145

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.

Name	Description	Example
<DATABASE_NAME>	Subsection. Database name Database name	RequestDB DBName = RequestDB
<DATABASE_NAME>/DBName	Database host server where the DB is located	Host = db01.in2p3.fr
<DATABASE_NAME>/Host	Maximum number of simultaneous queries to the	MaxQueueSize = 10
<DATABASE_NAME>/MaxQueueSize	DB per instance of the client	

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:

Name	Description	Example
LogLevel	Level of log verbosity	LogLevel = INFO
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

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:

Name	Description	Example
Path	Define the path where the request files are stored	Path = /opt/dirac/requestDB

Systems / RequestManagement / <INSTANCE> / URLs - Sub-subsection

RequestManagement Services URLs.

Name	Description	Example
<*SERVICE_NAME*>	URL associated with the service, value URL using dips protocol	RequestManager = dips://dirac.eela.if.ufrj.br:9143/RequestManagement/Re

Services associated with RequestManagement System:

Service	Port
RequestManager	9143

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.

Name	Description	Example
<DATABASE_NAME>	Subsection. Database name	ProxyDB
<DATABASE_NAME>/DBName	Database name	DBName = ProxyDB
<DATABASE_NAME>/Host	Database host server where the DB is located	Host = db01.in2p3.fr
<DATABASE_NAME>/MaxQueueSize	Maximum number of simultaneous queries to the DB per instance of the client	MaxQueueSize = 10

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:

Name	Description	Example
LogLevel	Level of logs	LogLevel = INFO
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 communicate with 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

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	SMSSwithc = 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.

You can automatically clean the versions directory adding the KeepSoftwareVersions option to the CS. For example:

KeepSoftwareVersions = 5

it will keep the last 5 version of the software.

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 contest 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.

Name	Description	Example
Agent	Subsection named as the agent is called.	CAUpdateAgent

Common options for all the agents:

Name	Description	Example
LogLevel	Log Level associated to the agent	LogLevel = DEBUG
LogBackends		LogBackends = stdout, server
MaxCycles	Maximum number of cycles made for Agent	MaxCycles = 500
MonitoringEnabled	Indicates if the monitoring of agent is enabled. Boolean values	MonitoringEnabled = True
PollingTime	Each many time a new cycle must start expresed in seconds	PollingTime = 2600
Status	Agent Status, possible values Active or Inactive	Status = Active

Agents associated with Framework 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:

Name	Description	Example
MinValidity	Proxy Minimal validity time expressed in seconds	MinValidity = 10000
PollingTime	Polling time in seconds	PollingTime = 1800
ValidityPeriod	The period for which the proxy will be extended. The value is in hours	ValidityPeriod = 15

Systems / Framework / <INSTANCE> / Agents / SystemLoggingDBCleaner - Sub-subsection

SystemLoggingDBCleaner 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:

Name	Description	Example
RemoveDate	Each many days the database must be clean Expressed in days	RemoveDate = 30

Systems / Framework / <INSTANCE> / Agents / TopErrorMessagesReportes - Sub-subsection

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:

Name	Description	Example
MailList	List of DIRAC users than the reporter going to receive Top Error Messages	MailList = mseco@in2p3.fr
NumberOfErrors	Number of top errors to be reported	NumberOfErrors = 10
QueryPeriod	Each how many time the agent is going to make the query, expressed in days	QueryPeriod = 7
Reviewer	Login of DIRAC user in charge of review the error message monitor	Reviewer = mseco
Threshold		Threshold = 10

Systems / Framework / <INSTANCE> / URLs - Sub-subsection

Framework Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, the value is a URL using dips protocol	Plotting = dips://dirac.eela.if.ufrj.br:9157/Framework/Plotting

Services associated with Framework System:

Service	Port
BundleDelivery	9158
Monitoring	9142
Notification	9154
Plotting	9157
ProxyManagement	9152
SecurityLogging	9153
SystemAdministrator	9162
SystemLogging	9141
SystemLoggingReport	9144
UserProfileManager	9155

StorageManagement System configuration

In this subsection are described the databases, services and URLs related with RequestManagement System for each setup.

Systems / StorageManagement / <INSTANCE> / Databases - Sub-subsection

Databases used by RequestManagement System. Note that each database is a separate subsection.

Name	Description	Example
<DATABASE_NAME> *<DATABASE_NAME>/DBName* *<DATABASE_NAME>/Host* *<DATABASE_NAME>/MaxQueueSize*	Subsection. Database name Database name Database host server where the DB is located Maximum number of simultaneous queries to the DB per instance of the client	StorageManagementDB DBName = StorageManagementDB Host = db01.in2p3.fr MaxQueueSize = 10

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:

Name	Description	Example
LogLevel	Level of log verbosity	LogLevel = INFO
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

DataStore services are:

Systems / StorageManagement / <INSTANCE> / Service / StorageManager - Sub-subsection

Systems / RequestManagement / <INSTANCE> / URLs - Sub-subsection

RequestManagement Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, value URL using dips protocol	RequestManager = dips://dirac.eela.if.ufrj.br:9143/RequestManagement/Re

Services associated with RequestManagement System:

Service	Port
RequestManager	9143

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 the Transformation System:

See also the sections in `TransformationSystem.Agent`

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

Name	Description	Example
FullUpdatePeriod	Time after a full update will be done	86400
RefreshOnly	Only refresh new files, needs the DateKey	False
DateKey	Meta data key for file creation date	
TransformationTypes	TransformationTypes to handle in this agent instance	

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

Name	Description	Example
TransformationTypes		
TasksPerIteration		50
MaxFailureRate		30
MaxWatingJobs		1000
EnableFlag		

Systems / Transformation / <INSTANCE> / Agents / TransformationAgent - Sub-subsection

The TransformationAgent processes transformations found in the transformation database.

Specific options defined in this sub-sections are: * TransformationTypes : list of transformation types handled by this specific agent * transformationStatus : list of statues considered by the agent * MaxFilesToProcess : maximum number of files passed to the plugin. This can be overwritten for individual plugins (see below) * ReplicaCacheValidity : validity of hte replica cache (in days) * maxThreadsInPool : maximum number of threads to be used * NoUnusedDelay : number of hours until the plugin is called again in case there is no new Unused files since last time

Name	Example
PluginLocation	DIRAC.TransformationSystem.Agent.TransformationPlugin
transformationStatus	Active, Completing, Flush
MaxFilesToProcess	5000
TransformationTypes	Replication
ReplicaCacheValidity	2
maxThreadsInPool	1
NoUnusedDelay	6
Transformation	All

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
- MaxFilesToProcess: supersede the agent's setting
- NoUnusedDelay: supersede the agent's setting

Systems / Transformation / <INSTANCE> / Agents / ValidateOutputDataAgent - Sub-subsection

The ValidateOutputDataAgent runs few integrity checks.

Name	Description	Example
TransformationTypes		
DirectoryLocations		
TransfIDMeta		
EnableFlag		

Systems / Transformation / <INSTANCE> / Databases - Sub-subsection

Databases used by RequestManagement System. Note that each database is a separate subsection.

Name	Description	Example
<DATABASE_NAME> *<DATABASE_NAME>/DBName* *<DATABASE_NAME>/Host* *<DATABASE_NAME>/MaxQueueSize*	Subsection. Database name Database name Database host server where the DB is located Maximum number of simultaneous queries to the DB per instance of the client	TransformationDB DBName = TransformationDB Host = db01.in2p3.fr MaxQueueSize = 10

The databases associated to Transformation System are: - TransformationDB

Systems / Transformation / <INSTANCE> / Services - Sub-subsection

All the services have common options to be configured for each one. Those options are presented in the following table:

Name	Description	Example
LogLevel	Level of log verbosity	LogLevel = INFO
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

Transformation services are:

Systems / Transformation / <INSTANCE> / Services / TransformationManager - Sub-subsection

Systems / Transformation / <INSTANCE> / URLs - Sub-subsection

Transformation Services URLs.

Name	Description	Example
<SERVICE_NAME>	URL associated with the service, value URL using dips protocol	TransformationManager = dips://.eela.if.ufrj.br:9131/Transformation/TransformationManager

Services associated with RequestManagement System:

Service	Port
TransformationManager	9131

Default structure

In each system, per setup, you normally find the following sections:

- Agents: definition of each agent
- Services: definition of each service
- Databases: definition of each db
- URLs: Resolution of the URL of a given Service (like 'DataManagement/FileCatalog') to a list of real urls (like 'dips://<host>:<port>/DataManagement/FileCatalog'). They are tried in a random order.
- FailoverURLs: Like URLs, but they are only tried if no server in URLs was successfully contacted.

Main Servers

There might be setup in which all services are installed behind one or several dns alias(es) or gateways (typically orchestrator like Mesos/Kubernetes). When this is the case, it can be bothering to redefine the very same URL everywhere, especially the day the machine name changes.

For this reason, there is the possibility to define a entry in the Operation section which contains the list of servers:

```
Operations/<Setup>/MainServers = server1, server2
```

There should be no port, no protocol. In the system configuration, one can then write:

```
System
{
  URLs
  {
    Service = dips://$MAINSERVERS$:1234/System/Service
  }
}
```

This will resolve in the following 2 urls:

```
dips://server1:1234/System/Service, dips://server2:1234/System/Service
```

Using together the FailoverURLs section, it can be interesting for orchestrator's setup, where there is a risk for the whole cluster to go down:

```
System
{
  URLs
  {
```

(continues on next page)

(continued from previous page)

```
    Service = dips://$MAINSERVERS$:1234/System/Service
  }
  FailoverURLs
  {
    Service = dips://failover1:1234/System/Service,dips://failover2:1234/System/
↪Service
  }
}
Operations
{
  Defaults
  {
    MainServers = gateway1, gateway2
  }
}
```

This results in all calls going to gateway1 and gateway2, which could be frontend to your orchestrator, and only if none of them answers, then do we use failover1 and failover2, which can be installed on separate machines, independent from the orchestrator

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:

Option	Description	Example
AlarmsManagement	Allow to set notifications and manage alarms	
BookkeepingManagement	Allow Bookkeeping Management	
CSAdministrator	CS Administrator - possibility to edit the Configuration Service	
FileCatalogManagement	Allow FC Management	
FullDelegation	Allow getting full delegated proxies	
GenericPilot	Generic pilot	
JobAdministrator	Job Administrator	
JobSharing	Job sharing among members of a group	
LimitedDelegation	Allow getting only limited proxies (ie. pilots)	
NormalUser	Normal user operations	
Operator	Operator	
Pilot	Private pilot	
PrivateLimitedDelegation	Allow getting only limited proxies for one self	
ProductionManagement	Allow managing production	
ProxyManagement	Allow managing proxies	
PPGAuthority	Allow production request approval on behalf of PPG	
ServiceAdministrator	DIRAC Service Administrator	
SiteManager	Site Manager	
TrustedHost	Host defined in the system to be trusted	

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

Port	System	Service
9133	*Accounting*	DataStore
9134	*Accounting*	ReportGenerator
9135	*Configuration*	Server
9197	*DataManagement*	FileCatalog
9148	*DataManagement*	StorageElement
9149	*DataManagement*	StorageElementProxy
9158	*Framework*	BundleDelivery
9142	*Framework*	Monitoring
9154	*Framework*	Notification
9157	*Framework*	Plotting
9152	*Framework*	ProxyManager
9153	*Framework*	SecurityLogging
9162	*Framework*	SystemAdministrator
9141	*Framework*	SystemLogging
9144	*Framework*	SystemLoggingReport
9155	*Framework*	UserProfileManager
9143	*RequestManagement*	RequestManager
9132	*WorkloadManagement*	JobManager
9130	*WorkloadManagement*	JobMonitoring
9136	*WorkloadManagement*	JobStateUpdate
9170	*WorkloadManagement*	Matcher
9196	*WorkloadManagement*	SandboxStore
9145	*WorkloadManagement*	WMSAdministrator

Ordered by port number

Port	System	Service
9130	*WorkloadManagement*	JobMonitoring
9132	*WorkloadManagement*	JobManager
9133	*Accounting*	DataStore
9134	*Accounting*	ReportGenerator
9135	*Configuration*	Server
9136	*WorkloadManagement*	JobStateUpdate
9141	*Framework*	SystemLogging
9142	*Framework*	Monitoring
9143	*RequestManagement*	RequestManager
9144	*Framework*	SystemLoggingReport
9145	*WorkloadManagement*	WMSAdministrator
9148	*DataManagement*	StorageElement
9149	*DataManagement*	StorageElementProxy
9152	*Framework*	ProxyManager
9153	*Framework*	SecurityLogging
9154	*Framework*	Notification
9155	*Framework*	UserProfileManager
9157	*Framework*	Plotting
9158	*Framework*	BundleDelivery
9162	*Framework*	SystemAdministrator
9170	*WorkloadManagement*	Matcher
9196	*WorkloadManagement*	SandboxStore
9197	*DataManagement*	FileCatalog

Note: This configuration file can be edited by hand, but we strongly recommend you to configure using DIRAC Web Portal.

2.7.3 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*:

```
DIRAC
{
  Setups
  {
    Dirac-Production
    {
      Configuration = Production
      Framework = Production
    }
  }
}
```

2.8 Manage authentification and authorizations

2.8.1 Authentication

DIRAC uses X509 certificates to identify clients and hosts, by conception X509 certificates are a very strong way to identify hosts and client thanks to asymetric cryptography. DIRAC is based on the openSSL library.

To identify users DIRAC use RBAC model (Role Based Access Control)

- A role (called property in DIRAC) carries some authorization

- A hostname has a DN and some properties
- A username has a DN, and the groups in which it is included
- A user group has a number of properties

Before authorize or not some tasks you have to define these properties, hostnames, usernames and groups. For that you may register informations at `/DIRAC/Registry`. After registering users create a proxy with a group and this guarantees certain properties.

Bellow a simple example with only one user, one group and one host:

```
Registry
{
  Users
  {
    userName
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
      Email = youremail@yourprovider.com
    }
  }

  Groups
  {
    groupName
    {
      Users = userName
      Properties = CSAdministrator, JobAdministrator, ServiceAdministrator,␣
↪ProxyDelegation, FullDelegation
    }
  }

  Hosts
  {
    hostName
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=dirac.cern.ch/emailAddress=lhcb-dirac-ci@cern.
↪ch
      Properties = CSAdministrator, JobAdministrator, ServiceAdministrator,␣
↪ProxyDelegation, FullDelegation
    }
  }
}
```

2.8.2 Authorizations

All procedure have a list of required properties and user may have at least one property to execute the procedure. Be careful, properties are associated with groups, not directly with users!

There are two main ways to define required properties:

- "Hardcoded" way: Directly in the code, in your request handler you can write `` `auth_yourMethodName = listOfProperties` ``. It can be useful for development or to provide default values.
- Via the configuration system at `` `/DIRAC/Systems/(SystemName)/(InstanceName)/Services/(ServiceName)/Authorization/(methodName)` ``, if you have also define hardcoded properties, hardcoded properties will be ignored.

A complete list of properties is available in *System Authorization*. If you don't want to define specific properties you can use "authenticated", "any" and "all".

- "authenticated" allow all users registered in the configuration system to use the procedure (`/DIRAC/Registry/Users`).
- "any" and "all" have the same effect, everyone can call the procedure. It can be dangerous if you allow non-secured connections.

You also have to define properties for groups of users in the configuration system at `` `/DIRAC/Registry/Groups/(groupName)/Properties` ``.

2.9 DIRAC Systems in details

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.

2.9.1 Accounting System

Table of contents

The Accounting system is responsible to collect and store data regarding to the activities: data transfers, pilot jobs. It is designed for store historical data by creating time buckets. The data stored with properties, which are used to classify the records: user, site and also properties which can be measured: memory, CPU.

The data can be accessible through the DIRAC web framework using the Accounting application. The records are stored in the AccountingDB, in "two" different formats:

- raw records
- time buckets: this is displayed to the users

The system consists of the following accounting types:

- Job: for creating reports of the activity on the computing resources such as Grid, Cloud, etc.
- Pilot: for creating reports for pilot jobs running on different computing elements such as ARC CE, CREAM, VAC, etc.
- Data operation: for creating reports about data activities: transfers, replication, removal, etc.
- WMS History: This it used for monitoring the DIRAC Workload Management system. This type is replaced by the WMS monitoring which

is part of the Monitoring system. It is replaced, because the WMS History type is for real time monitoring and MySQL is not for storing time series with high resolution.

AccountingDB

It is based on MySQL. It stores the raw records and the time buckets and provides the functionalities for creating the accounting reports. According to the computing activities (for example running jobs) and the size of the DIRAC system the size of the db can be small: a single MySQL server or it can be a multiple instance. The system can allow to store the accounting types in different database instances using Multi-DB accounting.

Multi-DB accounting

Accounting types 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
        Password = dirac
        DBName = infernus
      }
      MultiDB
      {
        WMSHistory = Acc2
      }
    }
  }
}
```

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.

DataStore Helpers

From DIRAC v6r17p14 there is the possibility to run multiple 'DataStore' services, where one needs to be called 'DataStoreMaster', while all the others may be called anything else. The master will create the proper buckets and the helpers only insert the records to the 'in' table. For example:

```
install service Accounting DataStoreHelper -m DataStore -p RunBucketing=False -p␣
↪Port=9166
```

In the CS you have to define DataStoreMaster. For example:

```
URLs
{
  DataStore = dips://lbvobox105.cern.ch:9133/Accounting/DataStore
  DataStore += dips://lbvobox105.cern.ch:9166/Accounting/DataStoreHelper
  DataStore += dips://lbvobox102.cern.ch:9166/Accounting/DataStoreHelper
  ReportGenerator = dips://lbvobox106.cern.ch:9134/Accounting/ReportGenerator
  DataStoreHelper = dips://lbvobox105.cern.ch:9166/Accounting/DataStoreHelper
  DataStoreHelper += dips://lbvobox102.cern.ch:9166/Accounting/DataStoreHelper
  DataStoreMaster = dips://lbvobox105.cern.ch:9133/Accounting/DataStore
}
```

Report generator

It is used for creating the accounting reports. Note: the report generator is caching the plots using the local file system. It is very important for running a service in a hardware which are having very good disk.

Installation

In order to use the system, it requires to install the following components: AccountingDB, DataStore, ReportGenerator, for the WMSMonitoring the StatesAccountingAgent. The simplest is by using the SystemAdministrator CLI:

```
install db AccountingDB
install service Accounting DataStore
install service Accounting ReportGenerator
install agent WorkloadManagement StatesAccountingAgent
```

Accounting user interface

The Accounting web application can be used for creating the reports. If you do not have WebAppDIRAC, please install it following *Installing WebAppDIRAC* instructions.

2.9.2 Configuration System

Table of contents

2.9.3 Data Management System

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.

Concepts

The whole DataManagement System (DMS) of DIRAC relies on a few key concepts:

- Logical File Name (LFN): the LFN is the name of a file, a path. It uniquely identifies a File throughout the DIRAC namespace. A file can have one or several *Replica*.
- Replica: This is a physical copy of an LFN. It is stored at a *StorageElement*. The couple *(LFN,StorageElement)* uniquely identifies a physical copy of a file.
- StorageElement: This represents a physical storage endpoint.
- Catalog: This is the namespace of the DataManagement. Files and their metadata are listed there

Systems in DIRAC (other than DMS) or users, when dealing with files, only have to care about LFNs. If, for some (unlikely) reasons, they need to address a specific replica, then they should use the couple *(LFN, StorageElement name)*. At no point, anywhere, is there a protocol or a URL leaking out of the low level of the DMS.

Logical File Names

The LFN is the unique identifier of a file throughout the namespace. It takes the form of a path, where the first directory should be the VO name. For example */lhcb/user/c/chaen/myFile.txt*.

StorageElements

For details on how to configure them, please see *StorageElement*.

DIRAC provides an abstraction to the storage endpoints called *StorageElement*. They are described in the CS, together with all the configuration necessary to physically access the files. There is never any URL leaking from the StorageElement to the rest of the system.

Catalogs

The concept of Catalogs is just the one of a *Namespace*. it is a place where you list your files and their metadata (size, checksum, list of SEs where they are stored, etc). DIRAC supports having several catalogs: in general, any operation done to one catalog will be performed to the others.

For more details, please see *Catalog*.

Dirac File Catalog

The DIRAC File Catalog (DFC) is a full replica and metadata catalog integrated to DIRAC. It has a very modular structure, allowing for several backends. The interaction with the backend is handled by *Managers* in such a way that the interface exposed to the users remains always the same.

There are two main sets of managers:

- the historical ones, offering the full range of functionalities and used by most VO
- and the LHCb ones, where a subsets of the functionalities related to user defined metadata are not tested, but optimized for scaling and consistency. Any VO could of course use it.

The DFC can be used also as a Metadata catalog. 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. One can declare ancestor files for a given file. This is often needed in order to keep track of the derived data provenance path.

Installation

The installation and configuration procedure changes slightly between the historical managers and the LHCb ones.

The list of components you need to have installed is:

- FileCatalogDB: if you want the standard managers, you should use *FileCatalogDB.sql*, but *FilecatalogWithFkAndPsDB.sql* if you want the LHCb ones
- FileCatalogHandler: just the interface to the DB

FileCatalogDB

No special configuration there.

FileCatalogHandler

All the configuration of the DFC takes place there.

- *DatasetManager*: default *DatasetManager* Manager for the dataset
- *DefaultUmask*: default *0775* Umask in octal
- *DirectoryManager*: default *DirectoryLevelTree* Manager for the Directories
- *DirectoryMetadata*: default *DirectoryMetadata* Manager for the directory metadata
- *FileManager*: default *FileManager* Manager for the files
- *FileMetadata*: default *FileMetadata* Manager for the file metadata
- *GlobalReadAccess*: default *True*. If set to True, anyone can read anything
- *LFNPFNConvention*: default *Strong*.
- *ResolvePFN*: default *True*. Deprecated
- *SecurityManager*: default *NoSecurityManager*. Manager for authentication
- *SecurityPolicy* : if *SecurityManager* = *PolicyBasedSecurityManager*, path to the policy to use
- *SEManager*: default *SEManagerDB*. Managers for the strage elements
- *UniqueGUID*: default *False*. If *True*, the GUID has to be unique through the namespace
- *UserGroupManager*: default *UserAndGroupManagerDB*. Managers for groups and users
- *ValidFileStatus*: default *[AprioriGood,Trash,Removing,Probing]*. Status that are valid for Files
- *ValidReplicaStatus*: default *[AprioriGoodTrashRemovingProbing]*. Status that are valid for Replicas
- *VisibleFileStatus*: default *[AprioriGood]*. By default, only files in this status are returned
- *VisibleReplicaStatus*: default *[AprioriGood]* By default, only replicas in this status are returned

In order to use the LHCb handler you should:

- *FileManager = FileManagerPs*
- *DirectoryManager = DirectoryClosure*
- *UniqueGUID = True*
- *SecurityManager = PolicyBasedSecurityManager*
- *SecurityPolicy = DIRAC/DataManagementSystem/DB/FileCatalogComponents/SecurityPolicies/VOMSPolicy*

Security Manager

This manager takes care of the access permissions in the DFC. There are several of them:

- *NoSecurityManager*: offer yourself to whatever treatment the world reserves you
- *DirectorySecurityManager*: only look at directories for permissions
- *FullSecurityManager*:
- *DirectorySecurityManagerWithDelete*: same as *DirectorySecurityManager* but consider the parent's directory write bit for removal
- *PolicyBasedSecurityManager*: based on plugins. It will evaluate the permissions based on the path, the identity doing the request, and the action itself on a per method bases. Currently, only the *VOMSPolicy* exists in DIRAC.

The *VOMSPolicy* (`VOMSPolicy`) implements a 3-level posix permission (directory-file-replica), and groups the dirac group using their VOMS roles. Basically, if the owner does not match, the groups are used. But the group doing the request and the one owning the file do not need to be the same: it is enough if they share the same VOMS role.

LFN PFN convention

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.

The transfers using FTS come from the RequestManagementSystem (see *Request Management System*). It will receive the files to transfer, as well as the list of destinations. If no source is defined, it will choose one. The files will then be grouped together and submited as jobs to the fts servers. These jobs will be monitored, retried if needed, the new replicas will be registered, and the status of the files will be reported back to the RMS.

There are no direct submission possible to the FTS system, it has to go through the RMS.

In the current system, only the production files can be transfered using FTS, sine the transfers are done using the Shifter proxy

Enable FTS transfers in the RMS

In order for the transfers to be submitted to the FTS system:

- *Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister/FTSMode* must be True
- *Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister/FTSBannedGroup* should contain the list of groups that are not production groups (users, etc)

Operations configuration

- DataManagement/FTSVersion: FTS2/FTS3. Set it to FTS3...
- DataManagement/FTSPlacement/FTS3/ServerPolicy: Policy to choose the FTS server see below

FTS servers definition

The servers to be used are defined in the *Resources/FTSEndpoints/FTS3* section. Example:

```
CERN-FTS3 = https://fts3.cern.ch:8446
RAL-FTS3 = https://lcgfts3.gridpp.rl.ac.uk:8446
```

The option name is just the server name as used internaly. Note that the port number has to be specified, and should correspond to the REST interface

Components

The list of components you need to have installed is:

- FTSDB: guess...
- FTSManager: just the interface to the DB
- FTSAgent: this agent runs the whole show
- CleanFTSDBAgent: cleans up the database from old jobs.

FTSDB

Two tables:

- FTSFile: an LFN and a destination SE, potentially a source SE, the metadata of the LFN, and the relevant IDs to make the link with the RMS. Also an link to the FTSJob table if they are currently being transfered.
- FTSJob: a job submitted to the FTS servers

FTSManager

No specific configuration for that one

CleanFTSDBAgent

This agent is responsible for cleaning the database from old jobs. Besides the usual agent options, these are the possible configurations:

- DeleteGraceDays: number of days after we remove a job in final status
- DeleteLimitPerCycle: maximum number of jobs we delete per agent cycle

FTSAgent

This is the complex one. The agent is going to fetch the request in state *Scheduled* in the RMS, and look in the FTSDB for the associated FTSFiles. It is then going to monitor submitted jobs, submit new jobs with new files or files that failed previously, register files successfuly transfered

The agent still supports old FTS2 server, but since there are no such servers anymore, this behavior will not be detailed here.

Configuration options

- FTSPlacementValidityPeriod: deprecated (FTS2)
- MaxActiveJobsPerRoute: deprecated (FTS2)
- MaxFilesPerJob: maximum number of files in a single fts job
- MaxRequests: maximum number of requests to look at per agent's cycle
- MaxThreads: maximum number of threads
- MaxTransferAttempts: maximum number of time we attempt to transfer a file
- MinThreads: minimum number of threads
- MonitorCommand: deprecated (FTS2)
- MonitoringInterval: interval between two monitoring of an FTSJob (in second)
- PinTime: when staging, pin time requested in the FTS job (in second)
- ProcessJobRequests: True if this agent is meant to process job only transfers (see *Multiple FTSAgents*)
- SubmitCommand: deprecated (FTS2)

File registration

The FTSAgent runs with the DataManagement shifter proxy, and hense can register the files directly after they have been transfered. If the registration fails, the FTSAgent still considers the transfer as done, and adds a RegisterFile operation in the RMS Request from which the transfers originated

Multiple FTSAgents

It is not possible to have several FTSAgents running in parallel except in a very specific configuration, which is 1 agent taking care of the failover transfers, 1 agent taking care of the transformation transfers. This behavior is enabled by the *ProcessJobRequests* flag. But be careful, two agents taking care of the same case would lead to problems.

Without entering the details on how to install several instances of the same agent, if you want such a configuration, it would look something like.

```
FTSAgent
{
  # All the common options
}

FTSAgentTransformations
{
```

(continues on next page)

(continued from previous page)

```
  Module = FTSAgent
  ProcessJobRequests = False
  ControlDirectory = control/DataManagement/FTSAgentTransformations
  # whatever other options
  # ...
}

FTSAgentFailover
{
  Module = FTSAgent
  ProcessJobRequests = True
  ControlDirectory = control/DataManagement/FTSAgentFailover
  # whatever other options
  # ...
}
```

FTSServer policy

The FTS server to which the job is sent is chose based on the policy. There are 3 possible policy:

- Random: the default. makes a random choice
- Failover: pick one, and stay on that one until it fails
- Sequence: take them in turn, always change

FTS3 support in DIRAC

New in version v6r20.

Table of contents

DIRAC DMS can be configured to make use of FTS3 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.

The transfers using FTS come from the RequestManagementSystem (see *Request Management System*). It will receive the files to transfer, as well as the list of destinations. If no source is defined, it will choose one. The files will then be grouped together and submited as jobs to the fts servers. These jobs will be monitored, retried if needed, the new replicas will be registered, and the status of the files will be reported back to the RMS.

There are no direct submission possible to the FTS system, it has to go through the RMS.

This system is independent from the previous FTS system, and is **totally incompatible with it. Both systems cannot run at the same time**.

To go from the old one, you must wait until there are no more Scheduled requests in the RequestManagementSystem (RMS). For that, either you do not submit any transfer for a while (probably not possible), or you switch to transfers using the DataManager. Once you have processed all the Schedule request, you can enable the new FTS3 system.

FTS3 Installation

One needs to install an FTS3DB, the FTS3Manager, and the FTS3Agent. Install the FTS3DB with *dirac-install-db* or directly on your mysql server and add the Databse in the Configuration System.

> dirac-admin-sysadmin-cli -H diracserver034.institute.tld > install service DataManagement FTS3Manager > install agent DataManagement FTS3Agent

Then enable the *UseNewFTS3* flag for the ReplicateAndRegister operation as described in *FTS3TransferOperation*.

Enable FTS transfers in the RMS

In order for the transfers to be submitted to the FTS system:

- *Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister/FTSMode* must be True
- *Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister/FTSBannedGroup* should contain the list of groups for which you'd rather do direct transfers.
- *Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister/UseNewFTS3* should be True in order to use this new FTS system (soon to be deprecated)

Operations configuration

- DataManagement/FTSVersion: FTS2/FTS3. Set it to FTS3...
- DataManagement/FTSPlacement/FTS3/ServerPolicy: Policy to choose the FTS server see *FTSServer policy*.

FTS servers definition

The servers to be used are defined in the *Resources/FTSEndpoints/FTS3* section. Example:

```
CERN-FTS3 = https://fts3.cern.ch:8446
RAL-FTS3 = https://lcgfts3.gridpp.rl.ac.uk:8446
```

The option name is just the server name as used internaly. Note that the port number has to be specified, and should correspond to the REST interface

FTS3Agent

This agent is in charge of performing and monitoring all the transfers. Note that this agent can be duplicated as many time as you wish.

See `FTS3Agent` for configuration details.

FTS3 system overview

There are two possible tasks that can be done with the FTS3 system: transferring and staging.

Each of these task is performed by a dedicated FTS3Operation: *FTS3TransferOperation* and *FTS3StagingOperation*. These FTS3Operation contain a list of FTS3File. An FTS3File is for a specific targetSE. The FTS3Agent will take an FTS3Operation, group the files following some criteria (see later) into FTS3Jobs. These FTS3Jobs will then be submitted to the FTS3 servers to become real FTS3 jobs. These Jobs are regularly monitored by the FTS3Agent. When all the FTS3Files have reached a final status, the FTS3Operation callback method is called. This callback method depends on the type of FTS3Operation.

Note that by default, the FTS3Agent is meant to run without shifter proxy. It will however download the proxy of the user submitting the job in order to delegate it to FTS. This also means that it is not able to perform registration in the DFC, and relies on Operation callback for that.

FTS3TransferOperation

When enabled by the flag *UseNewFTS3* in the ReplicateAndRegister operation definition, the RMS will create one FTS3TransferOperation per RMS Operation, and one FTS3File per RMS File. This means that there can be several destination SEs, and potentially source SEs specified.

The grouping into jobs is done following this logic:

- Group by target SE
- Group by source SE. If not specified, we take the active replicas as returned by the DataManager
- Since their might be several possible source SE, we need to pick one only. The choice is to select the SE where there is the most files of the operation present. This increases the likely hood to pick a good old Tier1
- Divide all that according to the maximum number of files we want per job

Once the FTS jobs have been executed, and all the operation is completed, the callback takes place. The callback consists in fetching the RMS request which submitted the FTS3Operation, update the status of the RMS files, and insert a Registration Operation. Note that since the multiple targets are grouped in a single RMS operation, failing to transfer one file t one destination will result in the failure of the Operation. However, there is one Registration operation per target, and hence correctly transferred files will be registered.

FTS3StagingOperation

> **Warning:** Still in development, not meant to be used

This operation is meant to perform BringOnline. The idea behind that is to replace, if deemed working, the whole StorageSystem of DIRAC.

FTSServer policy

The FTS server to which the job is sent is chose based on the policy. There are 3 possible policy:

- Random: the default. makes a random choice
- Failover: pick one, and stay on that one until it fails

- Sequence: take them in turn, always change

FTS3 state machines

These are the states for FTS3File:

```
ALL_STATES = [ 'New',  # Nothing was attempted yet on this file
               'Submitted', # From FTS: Initial state of a file as soon it's dropped␣
↪into the database
               'Ready', # From FTS: File is ready to become active
               'Active', # From FTS: File went active
               'Finished', # From FTS: File finished gracefully
               'Canceled', # From FTS: Canceled by the user
               'Staging', # From FTS: When staging of a file is requested
               'Failed', # From FTS: File failure
               'Defunct', # Totally fail, no more attempt will be made
               'Started', # From FTS: File transfer has started
               ]

FINAL_STATES = ['Canceled', 'Finished', 'Defunct']
FTS_FINAL_STATES = ['Canceled', 'Finished', 'Done']
INIT_STATE = 'New'
```

These are the states for FTS3Operation:

```
ALL_STATES = ['Active',  # Default state until FTS has done everything
              'Processed',  # Interactions with FTS done, but callback not done
              'Finished',  # Everything was done
              'Canceled',  # Canceled by the user
              'Failed',  # I don't know yet
             ]
FINAL_STATES = ['Finished', 'Canceled', 'Failed' ]
INIT_STATE = 'Active'
```

States from the FTS3Job:

```
# States from FTS doc
ALL_STATES = ['Submitted',  # Initial state of a job as soon it's dropped into the␣
↪database
              'Ready', # One of the files within a job went to Ready state
              'Active', # One of the files within a job went to Active state
              'Finished', # All files Finished gracefully
              'Canceled', # Job canceled
              'Failed', # All files Failed
              'Finisheddirty',  # Some files Failed
              'Staging', # One of the files within a job went to Staging state
             ]

FINAL_STATES = ['Canceled', 'Failed', 'Finished', 'Finisheddirty']
INIT_STATE = 'Submitted'
```

The status of the FTS3Jobs and FTSFiles are updated every time we monitor the matching job.

The FTS3Operation goes to Processed when all the files are in a final state, and to Finished when the callback has been called successfully

2.9.4 Framework System

The DIRAC FrameworkSystem contains those components that are used for administering DIRAC installations. Most of them are an essential part of a server installation of DIRAC.

The functionalities that are exposed by the framework system include, but are not limited to, the Instantiation of DIRAC components, but also the DIRAC commands (scripts), the management and monitoring of components.

The management of DIRAC components include their installation and un-installation (the system will keep a history of them) and a monitoring system that accounts for CPU and memory usage, queries served, used threads, and other parameters.

Another very important functionality provided by the framework system is proxies management, via the ProxyManager service and database.

ComponentMonitoring, SecurityLogging, and ProxyManager services are only part of the services that constitute the Framework of DIRAC.

The following sections add some details for some of the Framework systems.

Table of contents

Static Component Monitoring

New in version v6r13.

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. Running this service is mandatory!

This information is accessible from both the system administration CLI and the Component History page in the Web Portal.

Installation

The service constitutes of one database (InstalledComponentsDB) and one service (Framework/ComponentMonitoring). These service and DB may have been installed already when DIRAC was installed the first time.

The script **dirac-populate-component-db** should then be used to populate the DB tables with the necessary information.

Interacting with the static component monitoring

Using the CLI (dirac-admin-sysadmin-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).

Dynamic Component Monitoring

It shows information about running DIRAC components such as CPU, Memory, Running threads etc. The information can be accessed from the 'dirac-admin-sysadmin-cli' using 'show profile'. The following parameters can be used:

```
- <system>: The name of the system for example: DataManagementSystem
- <component>: The component name for example: FileCatalog
- -s <size>: number of elements to be shown
- h <host>: name of the host where a specific component is running
- id <initial date DD/MM/YYYY> the date where from we are interested for the log of a␣
↪specific component
- it <initial time hh:mm> the time where from we are interested for the log of a␣
↪specific component
- ed <end date DD/MM/YYYY>: the date before we are interested for   the log of a␣
↪specific component
- et <end time hh:mm>: the time before we are interested for   the log of a specific␣
↪component
- show <size>: log lines of profiling information for a component in the machine
↪<host>
```

The Framework/monitoring service

The Framework/Monitoring service collects information from all the active DIRAC services and Agents. The information are collected in *rrd* files which are keeping the monitoring information. This information is available as time

dependent plots via the ActivityMonitor web portal application. You can access these plots via the "System overview plots" tab in this application. In particular, it shows the load of the services in terms of CPU/Memory but also numbers of queries served, numbers of active threads, pending queries, etc. These plots are very useful for understanding of your services behavior, for example, of your FileCatalog service.

The bookkeeping of the rrd files is kept in an sqlite database usually kept in /opt/dirac/data/monitoring/monitoring.db file. There is no cleaning procedure foreseen for the rrd files.

A Monitoring System based on ElasticSearch database as backend is possible, please read about it in *Monitoring*.

The Framework/Notification service

The Framework/Notification service is responsible for notification, like as send mail, sms or alarm window on DIRAC portal. Send an email with supplied body to the specified address using the Mail utility. If avoidSpam is True, then emails are first added to a set so that duplicates are removed, and sent every hour.

Configure

The Notification service have next SMTP configuration parameters:

```
Systems
{
  Framework
  {
    Notification
    {
      SMTP
      {
        Port = < port of smtp server >
        Host = < smtp host name >
        Login = < account on smtp >
        Password = ***
        Protocol = < smtp protocol SSL/TSL (default None) >
      }
    }
  }
}
```

2.9.5 Request Management System

The DIRAC Request Management System (RMS) is a very generic system that allows for asynchronous actions execution. Its application ranges from failover system (if a DIRAC service or a StorageElement is unavailable at a certain point in time) to asynchronous task list (typically, for large scale data management operations like replications or removals). The RMS service is itself resilient to failure thanks to Request Proxies that can be scattered around your installation.

In order to have the an RMS system working, please see *RMS Components*

Concepts

Requests, Operations, Files

At the core of the RMS are *Requests*, *Operations* and *Files*.

A *Request* is like a TODO list associated to a User and group. For example, this TODO list could be what is left to do at the end of a job (setting the job status, moving the output file to its final destination, etc).

Each item on this TODO list is described by an *Operation*. There are several types of *Operation*, for example *ReplicateAndRegister* (to copy a file), *RemoveFile* (guess..), *ForwardDISET* (to execute DISET calls), etc.

When an *Operation* acts on LFNs, *Files* corresponding to the LFNs are associated to the *Operation*.

The list of available *Operations*, as well as the state machines are described in *RMS objects*

ReqManager & ReqProxy

The *ReqManager* is the service that receives or distributes *Requests* to be excuted. Every operation is synchronous with the *ReqDB* database.

If the ReqManager is unreachable when a client wants to send a *Request*, the client will automatically failover to a *ReqProxy*. This proxy will accept the *Request*, hold it in a local cache, and will periodically try to send it to the *ReqManager* until it succeeds. This system ensures that no Request is lost.

RequestExecutingAgent

The *RequestExecutingAgent* (*RequestExecutingAgent*) is in charge of executing the *Requests*.

CleanReqDBAgent

Because the database can grow very large, the *CleanReqDBAgent* is in charge of removing old *Requests* in a final state.

RMS objects

Requests

A *Request* is like a TODO list, each of the task being an *Operation*. A *Request* has the following attributes:

- *CreationTime*: Time at which the Request was created
- *Error*: Error if any at execution
- *JobID*: The ID of the job that generated the request. If it comes from another source, it is 0
- *LastUpdate*: Last time the request was touched
- *NotBefore*: Time before which no execution will be attempted (automatic increments in case of failures, or unavailable SE)
- *OwnerDN*: DN of the owner of the request
- *OwnerGroup*: group of the owner of the request
- *RequestID*: Unique identifier of the request
- *RequestName*: Convenience name, does not need to be unique
- *SourceComponent*: Who/what created the request (unused)
- *Status*: Status of the request (see *RMS state machine*)
- *SubmitTime*: Time at which the request was submitted to the database

And of course, it has an ordered list of *Operations*

Operations

An *Operation* is a task to execute. It is ordered within its *Request*

- *Arguments*: Generic blob used as placeholder for some Operation types
- *Catalog*: If an operation should target specific *Catalogs* only
- *CreationTime*: Time at which the Operation was created
- *Error*: Error if any at execution
- *LastUpdate*: Last time the Operation was touched
- *Order*: Execution order within the Request
- *SourceSE*: Coma separated list of StorageElement used as source (used by some Operation types)
- *Status*: Status of the Operation (see *RMS state machine*)
- *SubmitTime*: Time at which the Operation was submitted to the database
- *TargetSE*: Coma separated list of StorageElement used as target (used by some Operation types)
- *Type*: Type of Operation (see *Operation types*)

In some cases, an *Operation also has a list of 'Files* associated to it

Files

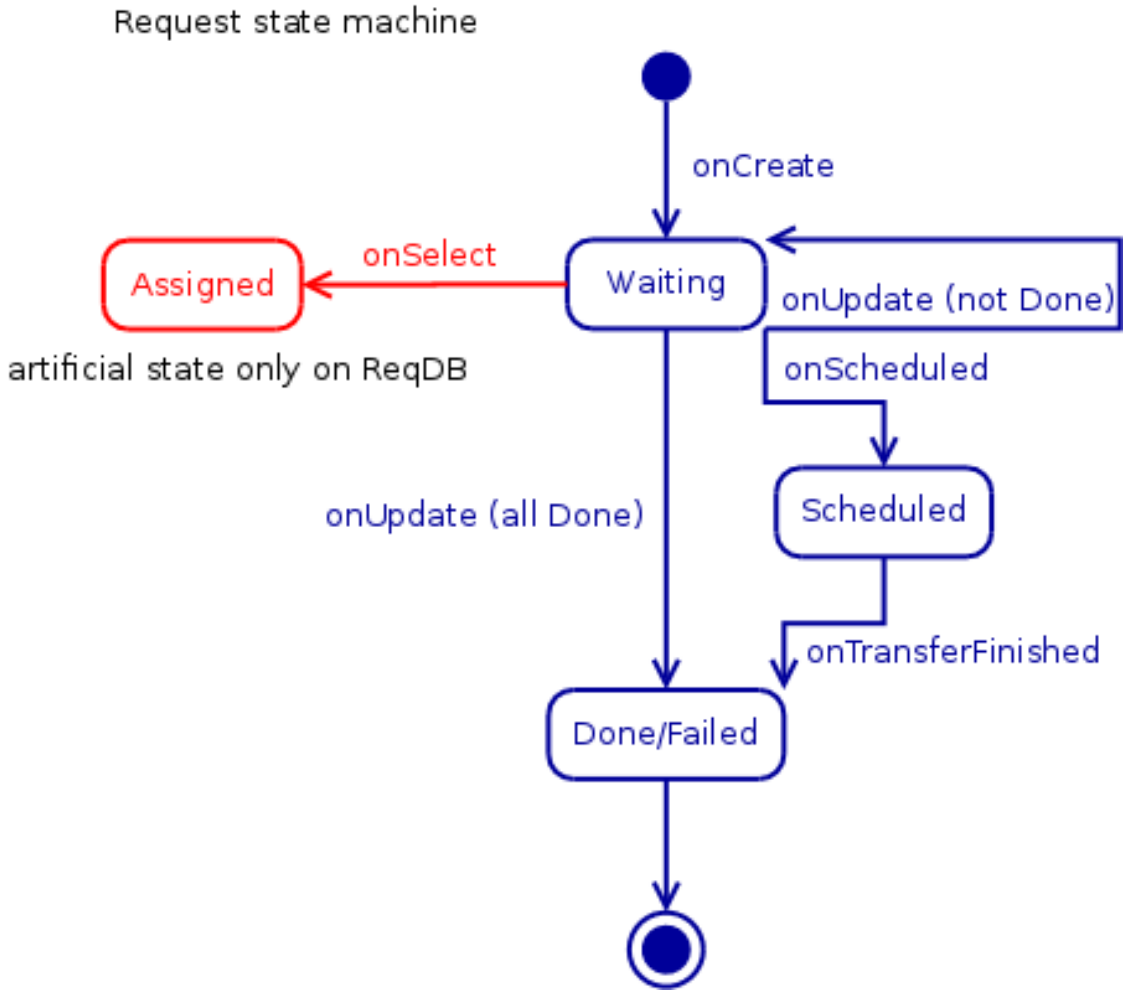
A *File* represents an LFN. Not all the *Operations* have *Files*. *Files'* attributes are

- *Attempt*: Number of time the *Operation* was attempted on that file
- *Checksum*: Checksum of the file
- *ChecksumType*: always *Adler32*
- *Error*: Error if any at execution
- *GUID*: file's GUID
- *LFN*: file's LFN
- *PFN*: file's URL, unused in practice
- *Size*: size of the file
- *Status*: Status of the File (see *RMS state machine*)

RMS state machine

The objects in the RMS obey a state machine in their execution. Each of them can have different statuses. The status of a *File* is determined by the success of the action we attempt to perform. The status of the Operation is inferred from the Files (if it has any, otherwise from the success of the execution). The status of the Request is inferred from the Operations

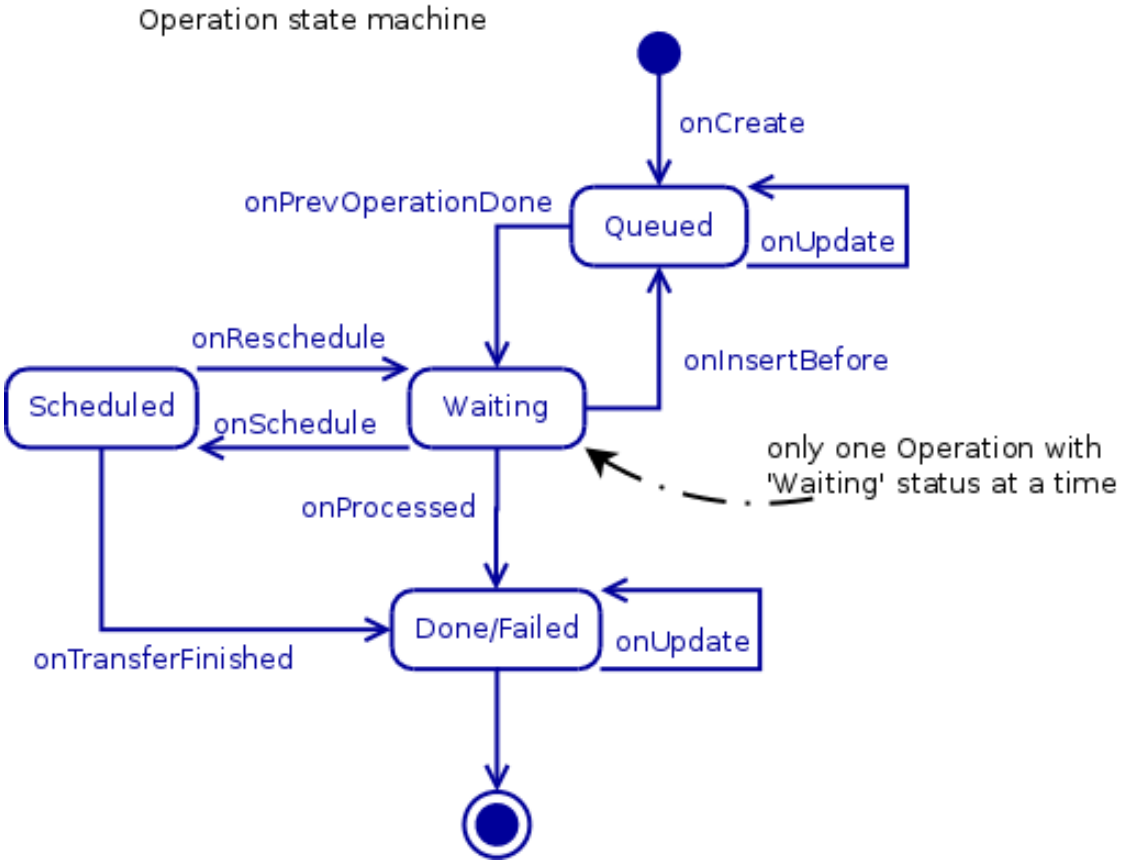
Request



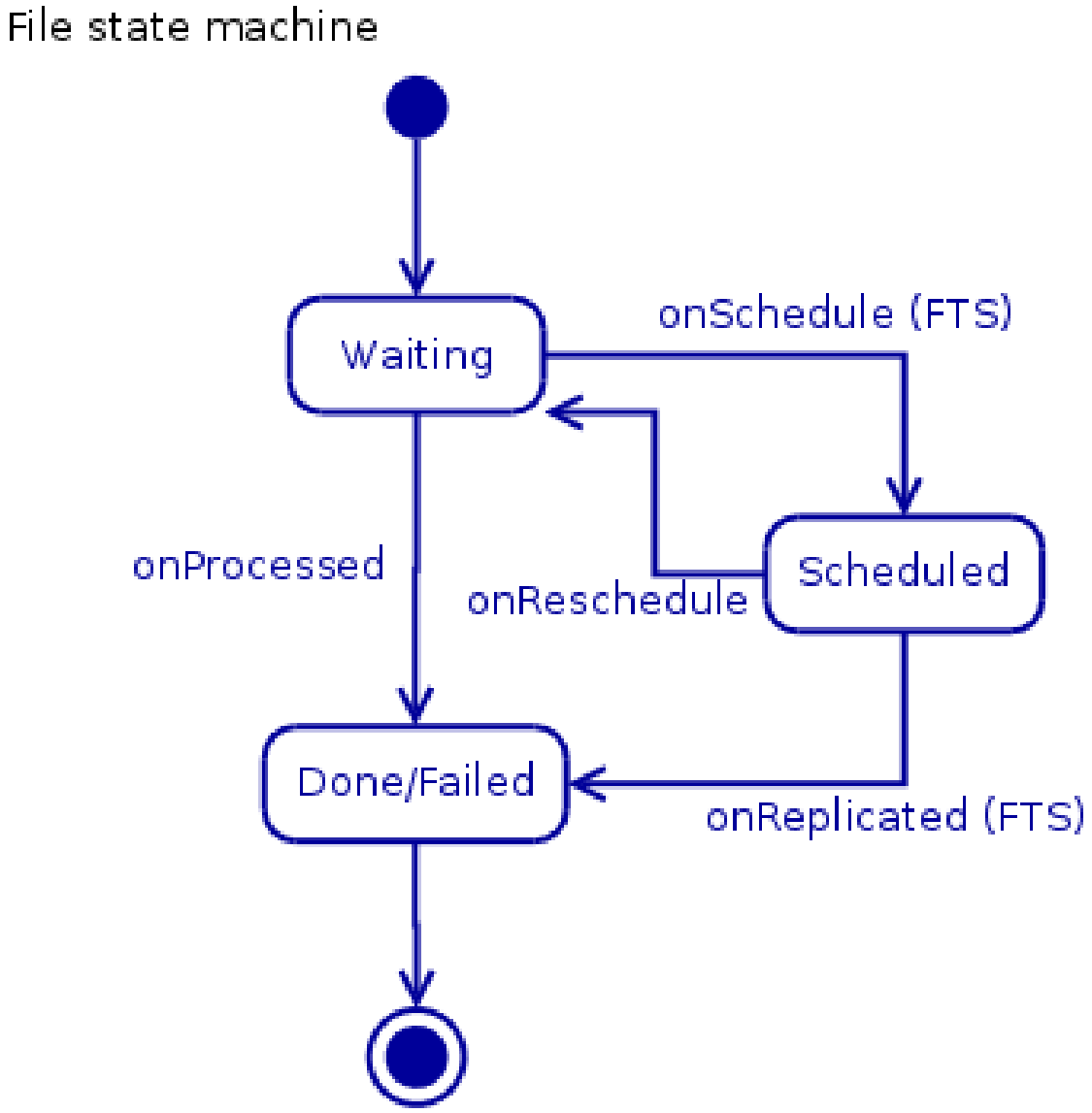
There are two special states for a Request:

- *Assigned*: this means that it has been picked up by a RequestExecutingAgent for execution
- *Canceled*: this means that we should stop trying. A Request can only be put manually in that state, and will remain as such (even if it was held by a RequestExecutingAgent, and set back)

Operation


Operation state machine
onCreate
onPrevOperationDone
Queued
onUpdate
onReschedule
onInsertBefore
Scheduled
onSchedule
Waiting
only one Operation with
'Waiting' status at a time
onProcessed
Done/Failed
onUpdate
onTransferFinished

File



Operation types

Each of this Type correspond to what can be found in the *Type* field of an *Operation*. In order to be executed, they need to be entered in the CS under */Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers*. Each of the Type must have its own section named after the type (for example */Systems/RequestManagementSystem/Agents/RequestExecutingAgent/OperationHandlers/ReplicateAndRegister*)

The OperationHandler sections share a few standard arguments:

- *Location*: (mandatory) Path (without .py) in the pythonpath to the handler
- *LogLevel*: self explanatory
- *MaxAttempts* (default 1024): Maximum attempts to try an Operation, after what, it fails. Note that this only works for Operations with *Files* (the others are tried forever).

- *TimeOut*: base timeout of the Operation
- *TimeOutPerFile*: additional timeout per file

If *TimeOut* is not specified, the default timeout of the RequestExecutingAgent is used. Otherwise, the total timeout when executing an operation is calculated with *TimeOut + NbOfFiles * TimeOutPerFile*

For more information on how to add new Operation type, see *Request Management System*

DataManagement Operations

For these operations, the *SourceSE*, *TargetSE* and *Catalog* fields of an *Operation* are used

MoveReplica

This handler moves replicas from source SEs to target SEs.

Details: `MoveReplica`

No specific configuration options

PutAndRegister

Put a local file on an SE and registers it. This is very useful for example to move data from the experiment site to the grid world.

Details: `PutAndRegister`

No specific configuration options

RegisterFile

Register files in the FileCatalogs

Details: `RegisterFile`

RegisterReplica

Register a replica in the FileCatalogs

Details: `RegisterReplica`

RemoveFile

Remove a file from all SEs and FC

Details: `RemoveFile`

RemoveReplica

Remove the replica of a file at a given SE and from the FC

Details: `RemoveReplica`

ReplicateAndRegister

This Operation replicates a file to one or several SE. The source does not need to be specified, but can be. This is typically useful in case of failover: if a job tries to upload a file to its final destination and fails, it will upload it somewhere else, and creates a *ReplicateAndRegister* Operation as well as a *RemoveReplica* (from the temporary storage) Operation. The replication can be performed either locally, or delegating it to the FTS system (*FTS3 support in DIRAC*)

Details: `ReplicateAndRegister`

Extra configuration options:

- *FTSMode*: If True, will use FTS to transfer files
- *FTSBannedGroups* : list of groups for which not to use FTS
- *UseNewFTS3*: (default False) If true, will use the new FTS3 system introduced in v6r20

Others

ForwardDISET

The ForwardDISET operation is an operation allowing to execute a DISET RPC call on behalf of another user. Typically, when a datamanagement operation is performed, some accounting information are sent to the DataStore service. If this service turns out to be unavailable, a *Request* containing a *ForwardDISET* Operation will be created, that will just replay the exact same action.

Details: `ForwardDISET`

SetFileStatus

This *Operation* is used as a failover by jobs to set the status of a File in a Transformation.

`SetFileStatus`

RMS Components

All the components described here MUST be installed in order to have a working RMS. The exception is the ReqProxy, which is optional.

RequestDB

This DB hosts the various *RMS objects*. No special configuration

ReqManager

This is the service in front of the DB. It has the following special configuration options:

- *constantRequestDelay*: (default 0 minut) if not 0, this is the constant retry delay we add when putting a Request back to the DB

RequestExecutingAgent

The RequestExecutingAgent (REA) is in charge of executing the Requests.It will fetch requests from the database, and process them in parallel (using `ProcessPool`), using the proxy of the user that created the Request (this means the machine on which the REA runs must have enough privileges).

A Request will be fetched from the DB, and all its operation executed in turns. The execution stops either because everything is done, or because there is an error, or because we delegated the work to FTS.

At the end of the execution, if the Request comes from a job, we set the job to (*Done*, *Request Done*), providing its previous status was (*Completed*, *Pending Request*). If the request fails, the job will stay in this status (uncool...).

The RequestExecutingAgent is one of the few that can be duplicated. There are protections to make sure that a Request is only processed by one REA at the time.

Configuration options

On top of the standard agent options, the REA accepts the following configuration

- *BulkRequest* (default 0): If a positive integer *n* is given, we fetch *n* requests at once from the DB. Otherwise, one by one
- *MinProcess* (default 2): minimum number of workers process in the *ProcessPool*
- *MaxProcess* (default 4): maximum number of workers process in the *ProcessPool*
- *OperationHandlers*: There should be in this section one section per OperationHandler (see *Operation types*)
- *ProcessPoolQueueSize* (default 20): queue depth of the *ProcessPool*
- *ProcessPoolTimeout* (default 900 seconds): timeout for the *ProcessPool* finalization
- *ProcessPoolSleep* (default 5 seconds): sleep time before retrying to get a free slot in the *ProcessPool*
- *RequestsPerCycle* (default 100): number of Requests to execute per cycle

Retry strategy

Operations are normally retried several times in case they fail. There is a delay between each execution, depending on the case:

- If the option *constantRequestDelay* is set in the *ReqManager*, then we apply that one
- If one of the StorageElement (source or target) is banned, then we wait 1 hour (except if the SE is always banned, then we fail the Operation)
- Otherwise the delay increases following a logarithmic scale with the number of attempts

CleanReqDBAgent

This agent cleans the DB from old Requests in final state. Special configuration options are

- *DeleteGraceDays*: (default 60) Delay after which Requests are removed
- *DeleteLimit*: (default 100) Maximum number of Requests to remove per cycle
- *DeleteFailed*: (default False) Whether to delete also Failed request
- *KickGraceHours*: (default 1) After how long we should kick the Requests in *Assigned*

- *KickLimit*: (default 10000) Maximum number of requests kicked by cycle

ReqProxy

The ReqProxy service is used as a failover for the ReqManager. A client will first attempt to send a Request to the ReqManager, but if it fails for whatever reason (service or DB down), it will send it to one of the ReqProxy. The ReqProxy will then store the Request on the local disk of the machine, and will periodically attempt to forward the Request to the ReqManager until it succeeds.

It is not mandatory to have ReqProxy, but highly recommended.

The only specific configuration option is for the URLs section, where it should be *ReqProxyURLs*, instead of *ReqProxy*

2.9.6 Resource Status System

Introduction

Table of contents

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 ResourceStatusDB called *SiteStatus*).

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 SiteStatus. 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
- Resource
- Site

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 *ElementTypes*. 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 =
                              ...
                      ...
```

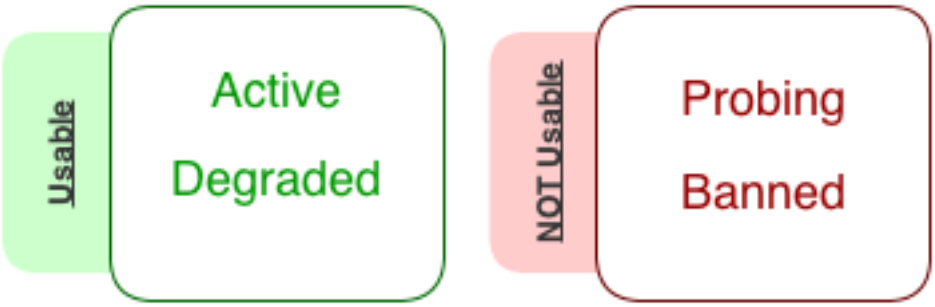
Each CE Resource has any number of Nodes, in this case of the ElementType Queue.

The list of ElementTypes per Element may vary depending on the CS/Resources section !

State

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:



If the Element status is:

- 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 *SiteStatus*. 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 *SiteStatus* 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.

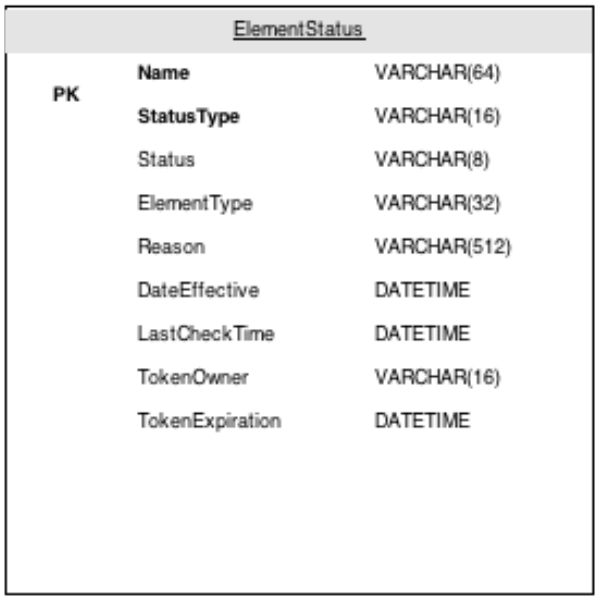
Resources() Helper

Warning: The Resources() Helper still needs to be developed.

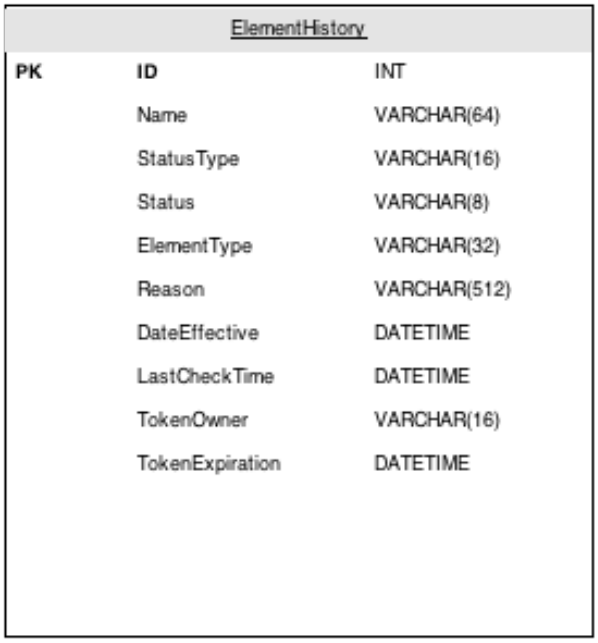
Database schema

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*).

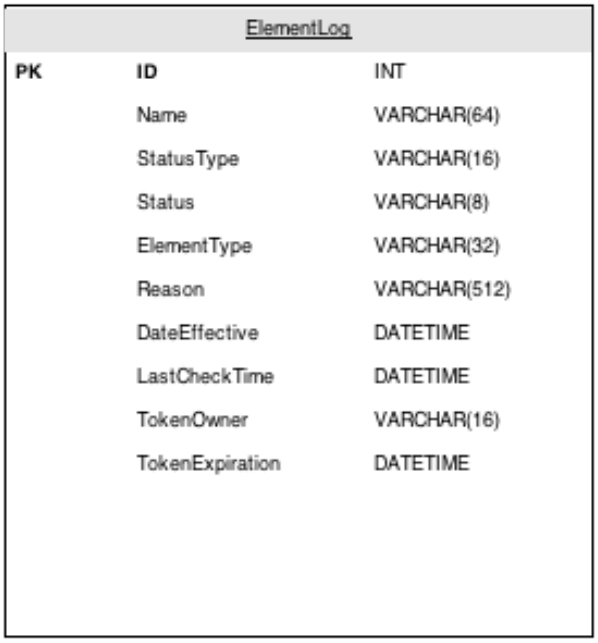
On each set there is a main table, called <element>Status (replace <element> with Site, Resource or Node), which contains all status information regarding that Elements family. The Status tables are enough to start running the RSS. However, if we need to keep track of the History of our Elements, the next two tables come into scene: <element>Log and <element>History.



ElementStatus		
PK	**Name**	VARCHAR(64)
	StatusType	VARCHAR(16)
	Status	VARCHAR(8)
	ElementType	VARCHAR(32)
	Reason	VARCHAR(512)
	DateEffective	DATETIME
	LastCheckTime	DATETIME
	TokenOwner	VARCHAR(16)
	TokenExpiration	DATETIME



ElementHistory		
PK	**ID**	INT
	Name	VARCHAR(64)
	StatusType	VARCHAR(16)
	Status	VARCHAR(8)
	ElementType	VARCHAR(32)
	Reason	VARCHAR(512)
	DateEffective	DATETIME
	LastCheckTime	DATETIME
	TokenOwner	VARCHAR(16)
	TokenExpiration	DATETIME



ElementLog		
PK	**ID**	INT
	Name	VARCHAR(64)
	StatusType	VARCHAR(16)
	Status	VARCHAR(8)
	ElementType	VARCHAR(32)
	Reason	VARCHAR(512)
	DateEffective	DATETIME
	LastCheckTime	DATETIME
	TokenOwner	VARCHAR(16)
	TokenExpiration	DATETIME

Every change on <element>Status is automatically recorded on <element>Log and kept for a *configurable amount of time*. The last table, <element>History summarizes <element>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

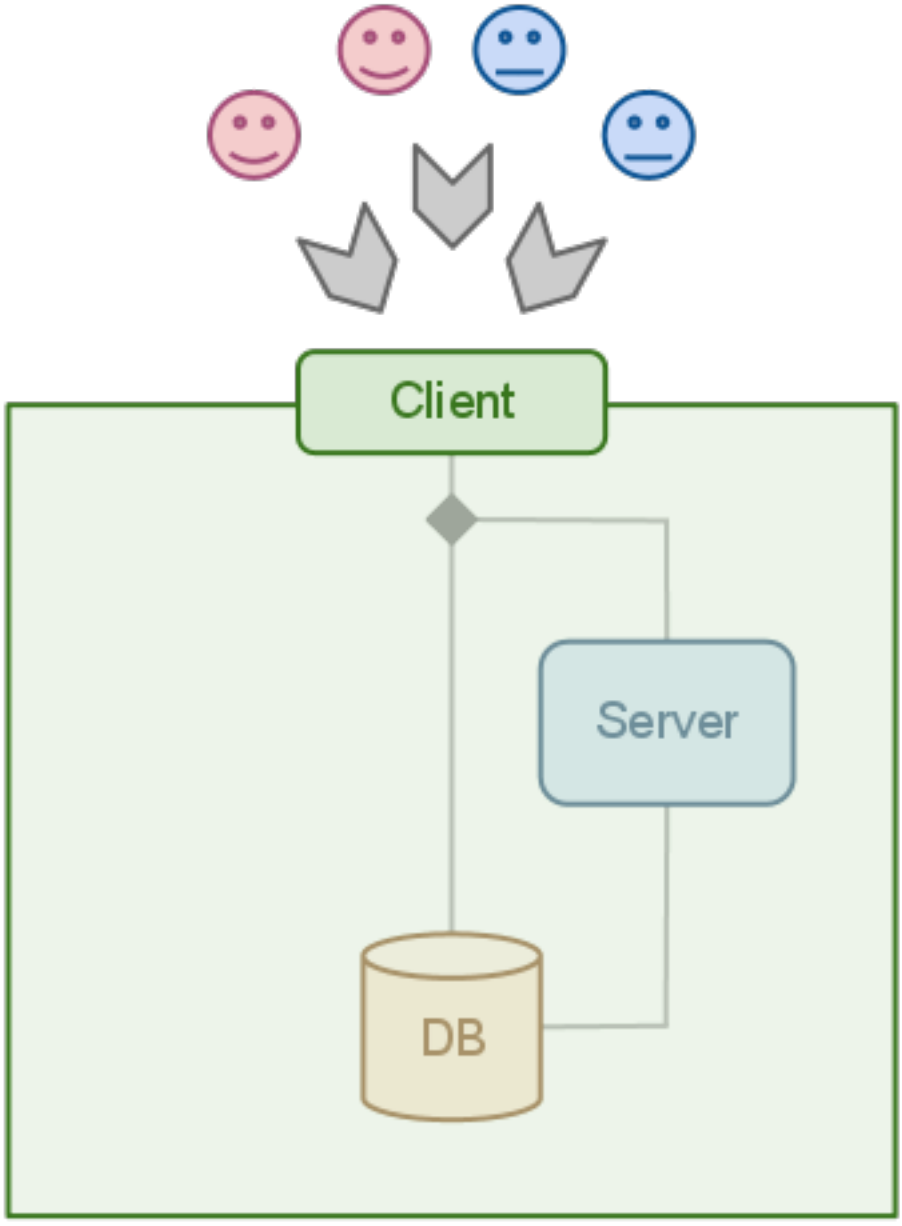
The Synchronizer is the code linking the CS and the RSS (in that direction, not viceversa !). Every change on the CS in terms of Element addition or deletion is reflected on the RSS. With other words, it populates the ResourceStatusDB

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

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:

```
/Operations/[Defaults|SetupName]/ResourceStatus
```

Please, make sure you have the following schema:

```
/Operations/[Defaults|SetupName]/ResourceStatus
  /Config
    State        = InActive
    Cache        = 300
    /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:

```
$ 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: ResourceStatus/ResourceStatus, ResourceStatus/ResourceManagement please install them using the dirac-admin-sysadmin-cli command, and make sure it is running.:

```
install service ResourceStatus ResourceManagement
install service ResourceStatus ResourceStatus
install service ResourceStatus Publisher
```

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:

```
$ 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:

```
$ dirac-rss-sync --init -o LogLevel=VERBOSE
```

WARNING: If the StorageElement does not have a particular StatusType declared

WARNING: on the CS, this script will set it to Banned. If that happens, you will

WARNING: have to issue the dirac-rss-status script over the elements that need

WARNING: 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:

```
$ dirac-rss-list-status --element Resource --elementType StorageElement --status␣
↪Banned
```

If is there any SE to be modified, you can do it as follows:

```
$ dirac-rss-set-status --element Resource --name CERN-USER --statusType ReadAccess --
↪status Active --reason "Why not?"
# This matches all StatusTypes
$ 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:

```
/Operations/[Defaults|SetupName]/ResourceStatus/Config/State = Active
```

Agents

The agents that are required:

- CacheFeederAgent
- SummarizeLogsAgent

The following agents are also necessary, but they won't do nothing until some policies are defined in the CS. The policy definitions is explained in *Advanced Configuration*

```
- ElementInspectorAgent
- SiteInspectorAgent
- TokenAgent
- EmailAgent
```

Please, install them and make sure they are up and running. The configuration of these agents can be found `Here`.

RSS Configuration

The basic configuration for the RSS is minimal, and must be placed under the Operations section, preferably on Defaults subsection.

```
/Operations/Defaults/ResourceStatus
                        /Config
                            State       = Active
                            Cache       = 720
                            FromAddress = email@address
                            /StatusTypes
                                default = all
                                StorageElement = ReadAccess,WriteAccess,CheckAccess,
↪RemoveAccess
```

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 t osend 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

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.

Helper

Let's get some statuses.

```
from DIRAC.ResourceStatusSystem.Client.ResourceStatus import ResourceStatus
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.

```
from DIRAC.ResourceStatusSystem.Client.ResourceStatus import ResourceStatus
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' )[ 'OK' ]
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 visble. 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 that 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:

LastCheckTime + lifetime(Status) < now()

where **LastCheckTime** is a timestamp column of the tables storing the element statuses and **lifetime(Status)** corresponds to the next table. The healthier the element is, the less often is checked.

Status	Lifetime (min)
Active	60
Degraded	30
Probing	30
Banned	30
Unknown	15
Error	15

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.

```
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).

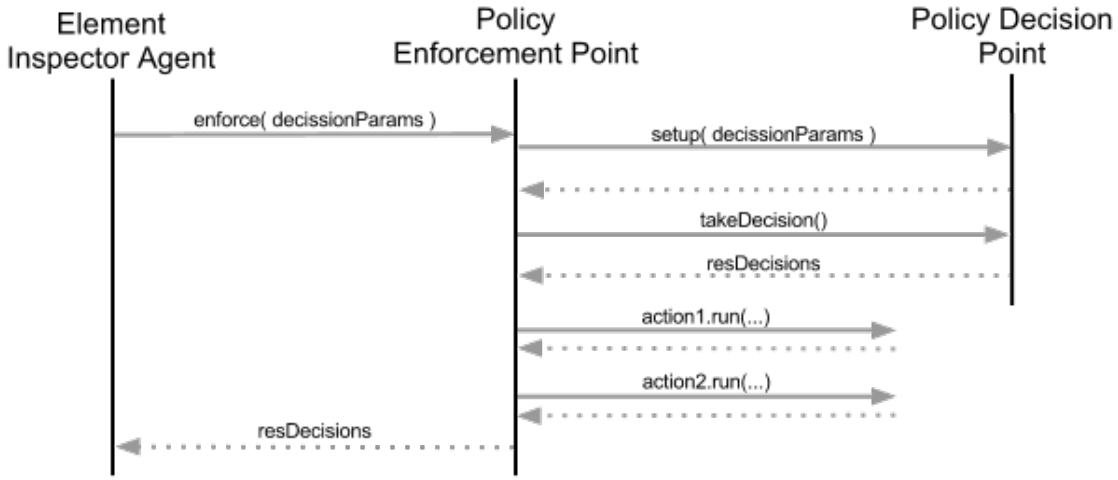
- Policy Enforcement Point (PEP)
- Policy Decision Point (PDP)
- Info Getter (IG)
- Policy Caller (PC)

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.

```
resDecisions = {
                'decissionParams'      : decisionParams,
                'policyCombinedResult' : {
                                            'Status'       : 'Active',
                                            'Reason'       : 'A Policy that always␣
↪returns Active ###',
                                            '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:

- decissionParams : 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:

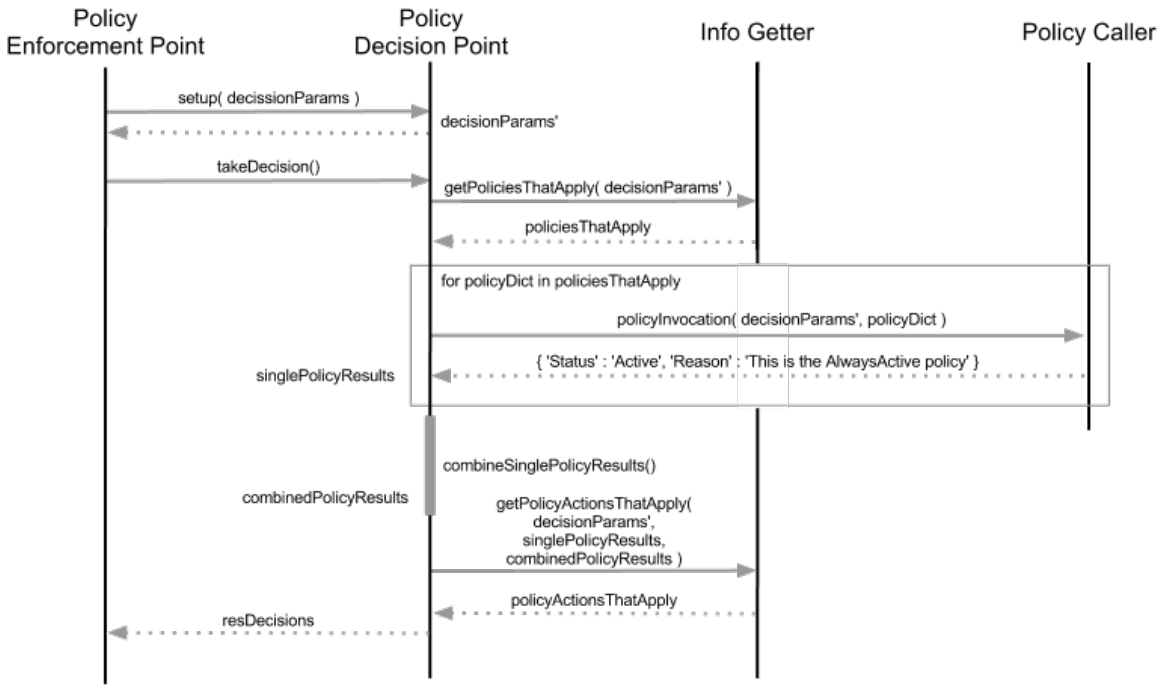
- **EmailAction** : sends an email notification
- **SMSAction** : sends a sms notification (not certified yet).
- **LogStatusAction** : updates the <element>Status table with the new computed status
- **LogPolicyResultAction** : updates the PolicyResult table with the results of the policies in singlePolicyResults.

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'*

```
# cannot name decisionParams' ( is not a valid python name ), decisionParams2 instead
decisionParams2 = {
                  'element'         : decisionParams.get( 'element'    , None ),
```

(continues on next page)

(continued from previous page)

```
                    '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 decission 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.

```
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**.

```
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:

```
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
```

(continues on next page)

(continued from previous page)

```
  # 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 decisioniParams 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 triger 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.

```
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:

```
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:

```
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|SetupName]/ResourceStatus
                          /Policies
                              /PolicyName
                                  policyType = policyType
                                  doNotCombineResult = something
                                  /matchParams
                                      element = element
                                      elementType = elementType
                                      name = name
                                      statusType = statusType
                                      status = status
                                      reason = reason
                                      tokenOwner = tokenOwner
                                      active = Active
```

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:

```
.../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.

```
/Operations/[Defaults|SetupName]/ResourceStatus
                          /PolicyActions
                              /PolicyActionName
                                  actionType = actionType
                                  notificationGroups = notificationGroups
                                  /matchParams
                                      element = element
                                      elementType = elementType
                                      name = name
                                      statusType = statusType
                                      status = status
                                      reason = reason
                                      tokenOwner = tokenOwner
                                      active = Active
                                  /combinedResult
                                      Status = Status
                                      Reason = Reason
                                  /policyResults
                                      policyName = policyStatus
```

Note: Mind te 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*.

Now that you have configured the policies, restart the ElementInspectorAgent and the SiteInspectorAgent, and see if the run the policies defined.

Notification

This section defines the notification groups (right now, only for EmailAction).

```
/Operations/[Defaults|SetupName]/ResourceStatus
                          /Notification
                              /NotificationGroupName
                                  users = email@address, email@address
```

- NotificationGroupName : human readable of what the group represents
- users : CSV with email addresses

The EmailAgent will take care of sending the appropriate Emails of notification.

Advanced Usage

Table of contents

scripts

Will come soon.

2.9.7 Storage Management System

Table of contents

2.9.8 Transformation System

Table of contents

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.

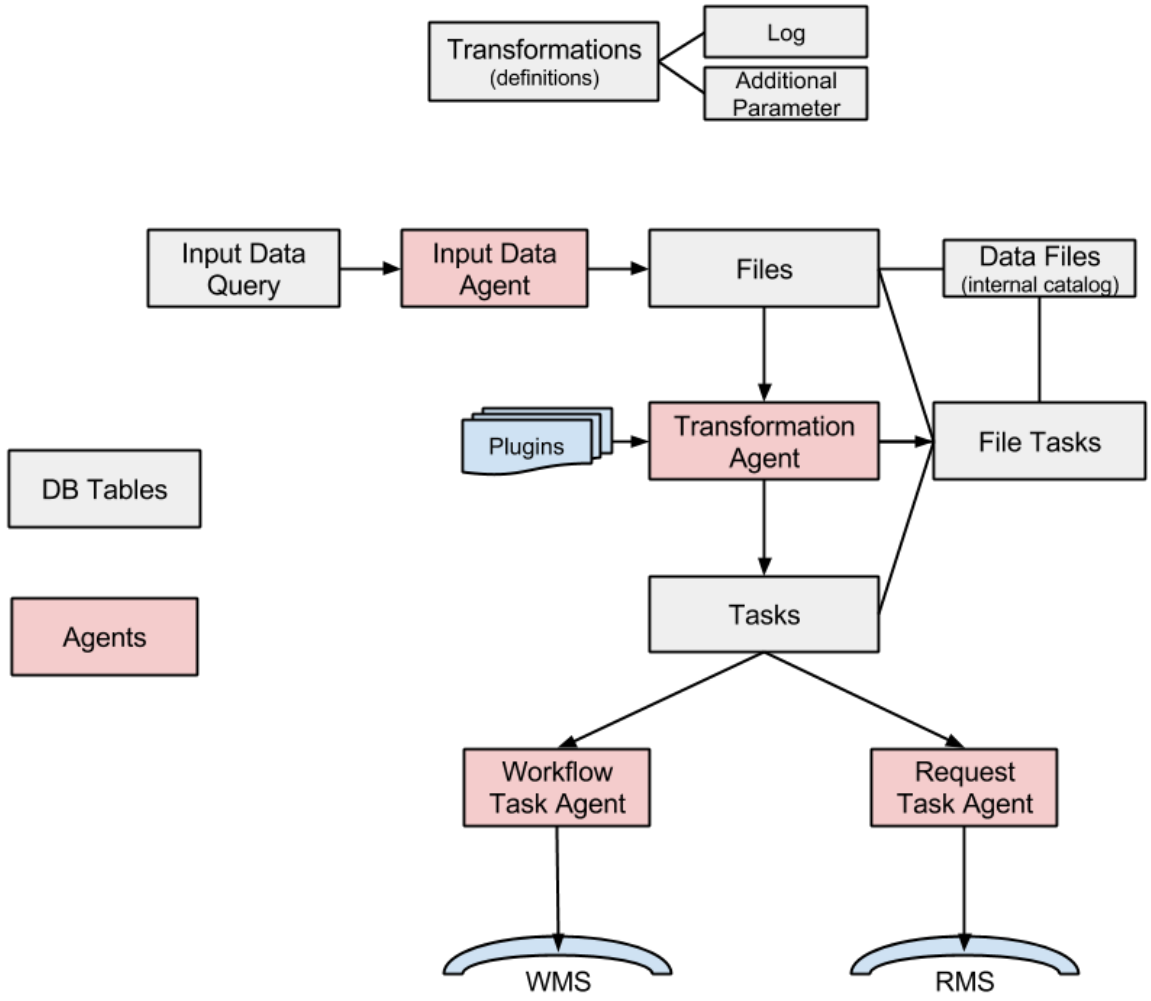
  New in version v6r20p3: Bulk submission of jobs is working for the transformations, so job submission can be sped up considerably.

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:

```
mysql> use TransformationDB;
Database changed
mysql> show tables;
+------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Tables_in_TransformationDB   |
+------------------------------+
| AdditionalParameters         |
| DataFiles                    |
| TaskInputs                   |
| TransformationFileTasks      |
| TransformationFiles          |
| TransformationInputDataQuery |
| TransformationLog            |
| TransformationTasks          |
| Transformations              |
+------------------------------+
```

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 it 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 WorkflowTasks 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]/[SETUP]/Transformations or Operations/Defaults/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/[SETUP]/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 different 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
```

(continues on next page)

(continued from previous page)

```
  MonitorFiles = yes
}
WorkflowTaskAgent-RealData
{
  TransType = DataReconstruction
  TransType += DataStripping
  shifterProxy = DataProcessing
  LoadName = WorkflowTaskAgent-RealData
  Module = WorkflowTaskAgent
}
WorkflowTaskAgent-Simulation
{
  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.

One possibility is represented by the **ByJobType** TaskManager plugin, that allows to specify different rules for site destinations for each JobType. This plugin allows so-called "mesh processing", i.e. depending on the job type, some sites may become eligible for "helping" other sites to run jobs that normally would only be running at the site where data is located. 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.*:

```
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:

```
JobTypeMapping
{
  AutoAddedSites = LCG.CERN.cern
  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
  DataReconstruction
  {
    Exclude = ALL
    AutoAddedSites = LCG.IN2P3.fr
    AutoAddedSites += LCG.CNAF.it
    AutoAddedSites += LCG.PIC.es
    AutoAddedSites += LCG.GRIDKA.de
    AutoAddedSites += LCG.RAL.uk
    AutoAddedSites += LCG.RRCKI.ru
    Allow
    {
      CLOUD.CERN.cern = LCG.CERN.cern, LCG.SARA.nl
    }
  }
  DataReprocessing
  {
    Exclude = ALL
    Allow
    {
      LCG.NIKHEF.nl = LCG.SARA.nl
      LCG.UKI-LT2-QMUL.uk = LCG.RAL.uk
      LCG.CPPM.fr = LCG.SARA.nl
```

(continues on next page)

(continued from previous page)

```
      LCG.USC.es = LCG.PIC.es
      LCG.LAL.fr = LCG.CERN.cern
      LCG.LAL.fr += LCG.IN2P3.fr
      LCG.BariRECAS.it = LCG.CNAF.it
      LCG.CBPF.br = LCG.CERN.cern
      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.

If it contains 'WithStorage', all sites with an associated local storage will be added automatically.

- 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).
  - Optionally one may redefine the "AutoAddedSites" (including setting it empty)
  - "Allow": the list of 'helpers', specifying sites helping another site.

  For each "helper" one specifies a list of sites that it helps, i.e. if the input data is at one of these sites, the job is eligible to the helper site.
- In the example above all sites in "AutoAddedSites" are allowed to run jobs with input files in their local SEs.

For DataReprocessing jobs, jobs having input files at LCG.SARA.nl local SEs can run both at LCG.SARA.nl and at LCG.NIKHEF.nl, etc. For DataReconstruction jobs, jobs will run at the Tier1 where the input data is, except when the data is at CERN or SARA, where they will run exclusively at CLOUD.CERN.cern.

Use-cases

Transformations can have Input Files (*e.g.* Data-processing transformations), or not (*e.g.* MC Simulation transformations).

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:

```
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("MCSimulation")
t.setDescription("MC prod 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")
```

Data-processing

Generation of identical jobs with varying Input Files.

- **Agents**

```
TransformationAgent, WorkflowTaskAgent, InputDataAgent
```

Input Files can be attached to a transformation in two ways:

- Through a static list of files:
  - when the transformation is created, all the tasks necessary to treat the list of files are also created
- Through a catalog query:
  - when the transformation is created, all the tasks the tasks necessary to treat the files matching the catalog query are created. As soon as new files matching the catalog query are registered, new tasks are created to treat the new files

Using a static list of files

- Example:

```
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")
```

(continues on next page)

(continued from previous page)

```
t.setDescription("repro example")
t.setLongDescription( "This is the long description of my reprocessing" ) #␣
↪mandatory
t.setBody ( j.workflow.toXML() )
t.addTransformation() # transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
transID = t.getTransformationID()
tc.addFilesToTransformation(transID['Value'],infileList) # files are added here
```

Using a catalog query

There are two methods to add Input Files to a transformation through a catalog query:

- Using the InputDataQuery Agent
- Using the TSCatalog interface (starting from v6r17)

From the user point of view the two methods are equivalent, but the internal behaviour of the TS is different. In the first case, the InputDataQuery agent continuously queries the catalog to look for new files matching the defined query (called 'InputDataQuery'). In the second case, the files matching the defined query (called 'FileMask'), are directly added to the transformation through the TSCatalog interface (see RFC 21 for more details). Here below we give an example to create a data-processing transformation for each of these two methods.

- Example using the InputDataQuery Agent

```
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.setBody ( j.workflow.toXML() )
t.addTransformation() # transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
tc.createTransformationInputDataQuery(transID['Value'], {'particle': 'proton',
↪'prodName':'ConfigtestCorsika','outputType':'corsikaData'}) # files are added␣
↪here
```

- Example using the TSCatalog interface

  Both the TSCatalog and FileCatalog plugins must be configured in the Resources and Operations sections, *e.g.*:

```
Operations
{
    Services
    {
        Catalogs
```

(continues on next page)

(continued from previous page)

```
        {
        CatalogList = DIRACFileCatalog, TSCatalog
        DIRACFileCatalog
        {
        CatalogType = FileCatalog
        AccessType = Read-Write
        Status = Active
        CatalogURL = DataManagement/FileCatalog
        }
        TSCatalog
        {
        CatalogType = TSCatalog
        AccessType = Write
        Status = Active
        CatalogURL = Transformation/TransformationManager
        }
```

```
import json
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.Interfaces.API.Job import Job
j = myJob()
...
t = Transformation( )
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.setBody ( j.workflow.toXML() )
mqJson = json.dumps( {'particle':'gamma_diffuse', 'zenith':{"<=": 20}} )
t.setFileMask(mqJson) # catalog query is defined here
t.addTransformation() # transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
```

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") # this 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
- Example for Multiple Operations

```
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.TransformationSystem.Client.TransformationClient import␣
↪TransformationClient

infileList = []
...

t = Transformation( )
tc = TransformationClient( )
t.setTransformationName("DM_Moving") # Must be unique
#t.setTransformationGroup("Moving")
t.setType("Moving")
t.setPlugin("Standard") # Not needed. The default is 'Standard'
t.setDescription("dataset1 Moving")
t.setLongDescription( "Long description of dataset1 Moving" ) # Mandatory
t.setGroupSize(2) # Here you specify how many files should be grouped within he␣
↪same request, e.g. 100
```

(continues on next page)

(continued from previous page)

```
transBody = [ ( "ReplicateAndRegister", { "SourceSE":"FOO-SRM", "TargetSE":"BAR-
↪SRM" }),
              ( "RemoveReplica", { "TargetSE":"FOO-SRM" } ),
            ]

t.setBody ( transBody ) # 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
```

Data replication based on catalog query

- Example of data replication (file list as a result of a DFC query, example taken from CTA)

```
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'}) # Add files to Transformation␣
↪based on Catalog Query
```

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. When a transformation is flushed also its replica cache will be re-created (instead of after 24 hours).
- **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.

Multi VO Configuration

New in version v6r20p5.

There are two possibilities to configure the agents of the transformation system for the use in a multi VO installation.

- Use the same WorkflowTaskAgent and RequestTaskAgents for multiple VOs, no *shifterProxy* or *ShifterCredential* must be set for these agents. If neither of those options are set the credentials of the owner of the transformations are used to submit Jobs or Requests.
- Use a set of WorkflowTaskAgent and RequestTaskAgent for each VO. This requires that each VO uses a distinct set of Transformation Types, e.g. MCSimulation_BigVO. This allows one to set VO specific shifterProxies. This setup is recommended to create a dedicated WorkflowTaskAgent or RequestTaskAgent for a VO that will create a large number of jobs or requests.

It is possible to mix the two configurations and have one WorkflowTaskAgent treat transformations of many smaller VOs, while installing a dedicated instance for the larger ones:

```
WorkflowTaskAgent
{
  ...
  TransType = MCSimulation
  TransType += MCReconstruction
  ...
  #No shifterProxy / ShifterCredentials
}
WorkflowTaskAgent-BigVO
{
  ...
  TransType = MCSimulation_BigVO
  TransType += MCReconstruction_BigVO
  Module = WorkflowTaskAgent
  ...
  #shifterProxy / ShifterCredentials are optional
}
```

2.9.9 Workload Management System (WMS)

The DIRAC Workload Management System (WMS) realizes the task scheduling paradigm with Generic *Pilot* Jobs. 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.

The WMS provides high user jobs efficiency, hiding the heterogeneity of the the underlying computing resources.

Within *DIRAC jobs* users specify at least an executable, and maybe some argument, that DIRAC will start on the Worker Node. 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 DB (using JDL, Job Description Language) and added to a Task Queue of jobs with same or similar requirements. Jobs will start running when their JDL is picked up by a pilot job.

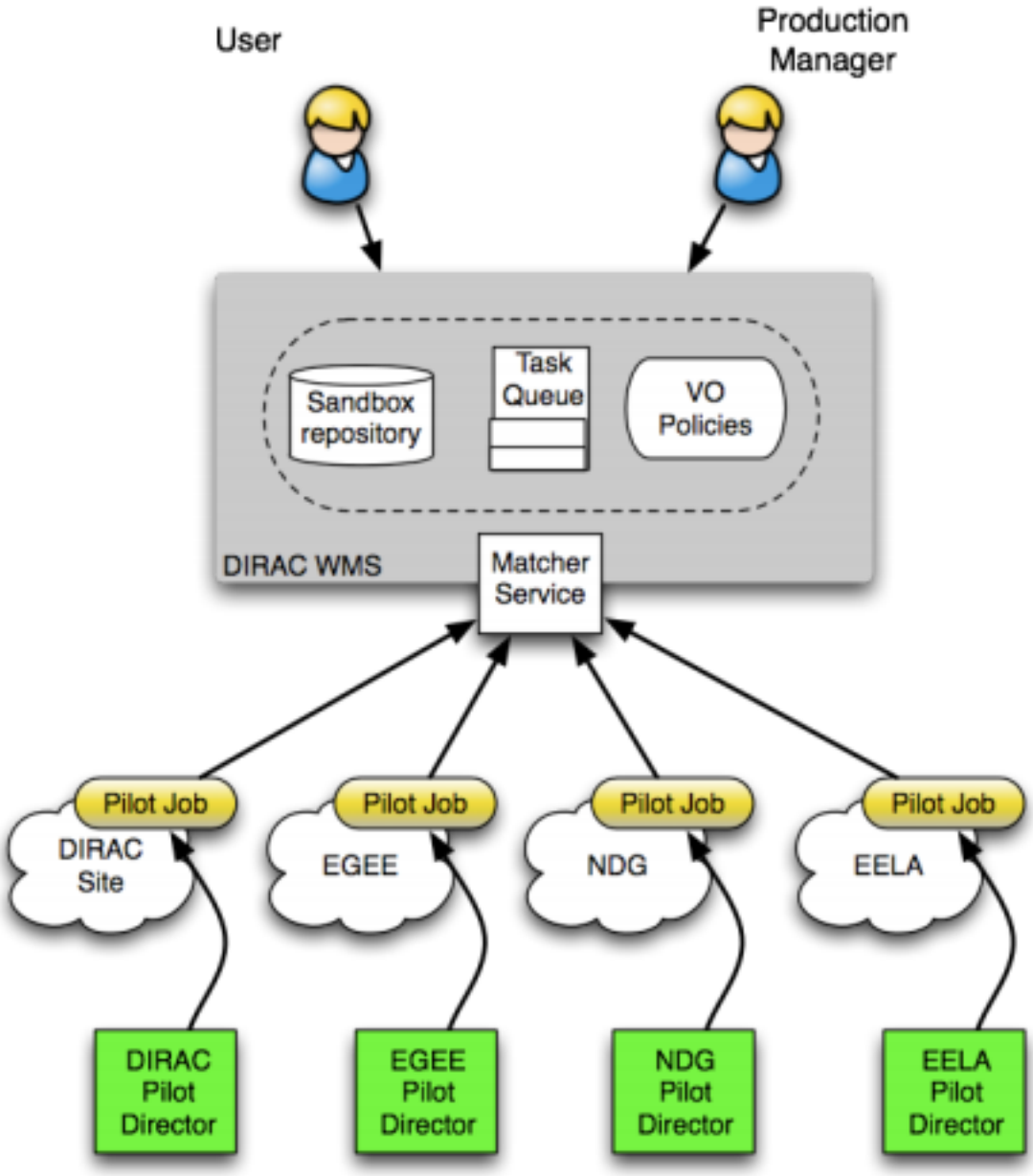
Pilot jobs are submitted to computing resources by specialized Pilot Directors. After the start, Pilots check the execution environment and form the resource description (OS, capacity, disk space, software, etc) The resources description

is presented to the Matcher service, which chooses the most appropriate user job from the Task Queue. The user job description is delivered to the pilot, which prepares its execution environment and executes the user application

One evident advantage is that the users' payloads are starting in an already verified environment. The environment checks can be tailored for specific needs of a particular community by customizing the pilot operations.

For the users all the internal WMS/pilots machinery is completely hidden. They see all the DIRAC operated computing resources as single large batch system.

The following picture shows a simplified view of how the system works



The *computing resources* that DIRAC can administer can be of different types.

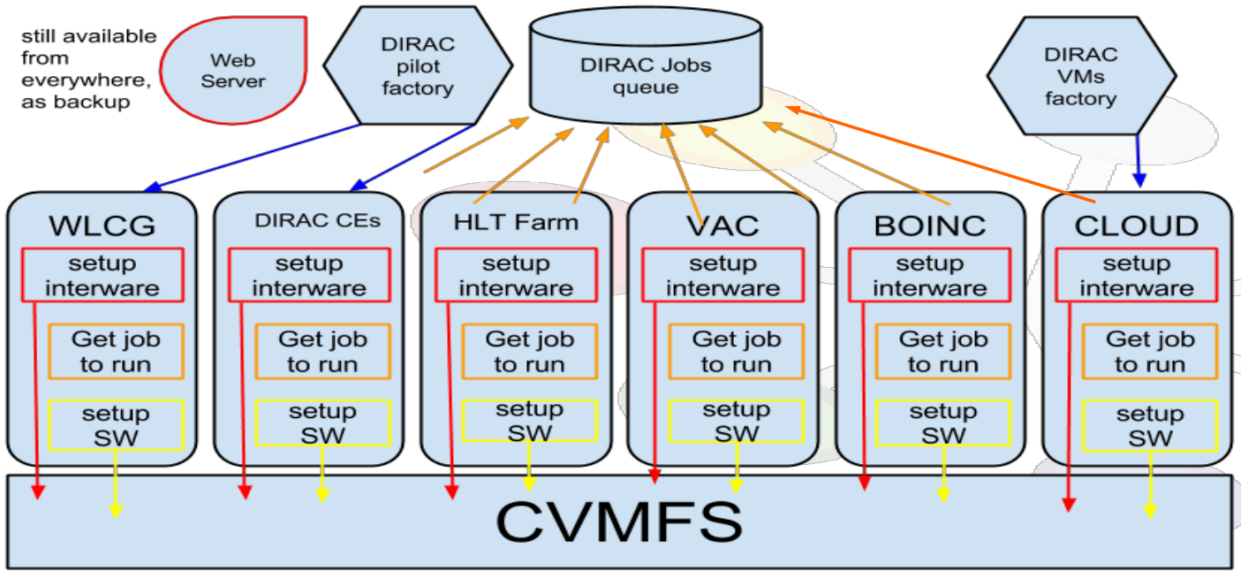
In any case, the following definitions apply:

- Sites: Administrative units that expose Grid resources

- Computing Element (CE): Sites managed computing resources entry points
- Worker Node (WN): a computing node where Pilots and Jobs run
- computing slot: a resource allocated by a resource provider for a community usage on a WN (a batch job slot, a VM slot. . .)

DIRAC alone can send pilots to several types of computing element, and recognizes several types of batch systems. You can find a presentation highlighting these concepts here.

In case more than one type of resource is available, specifically VM-based resources, the pilots scheduling should happen with other means then SiteDirectors, as exemplified in the following picture:



DIRAC alone does not administer directly clouds or any VM-based systems. A different mechanism should be used for starting pilots and jobs on worker nodes that can't be reached via Computing Elements. One mechanism for starting pilots on Clouds is in the *VMDIRAC* extension of DIRAC.

(Over-)simplified workflow

DIRAC WMS basically works as follows:

1. Users define and submit jobs. Jobs have requirements. Job descriptions are stored in DIRAC's Job DB.
2. DIRAC agents submit pilot jobs to sites. Alternatively, pilots are started on worker nodes in a different way.
3. Pilots will try to match the worker nodes' capabilities to Jobs requirements.
4. Jobs are started on WNs. DIRAC monitors its progress.

References

For more info on how the WMS work, please refer to this presentation.

The following sections add some detail for the WMS systems.

Workload Management System architecture

The WMS is a standard DIRAC system, and therefore it is composed by components in the following categories: Services, DBs, Agents, but also Executors.

Databases

JobDB Main WMS database containing job definitions and status information. It is used in most of the WMS components.

JobLoggingDB Simple Job Logging Database.

PilotAgentsDB Keep track of all the submitted grid pilot jobs. It also registers the mapping of the DIRAC jobs to the pilots.

SandboxMetadataDB Keep the metadata of the sandboxes.

TaskQueueDB The TaskQueueDB is used to organize jobs requirements into task queues, for easier matching.

All the DB above should be installed using the *system administrator console*.

Services

JobManager For submitting/rescheduling/killing/deleting jobs

JobMonitoring For monitoring jobs

Matcher For matching capabilities (of WNs) to requirements (of task queues –> so, of jobs)

JobStateUpdate For storing updates on Jobs' status

OptimizationMind For Jobs scheduling optimization

SandboxStore Frontend for storing and retrieving sandboxes

WMSAdministrator For administering jobs and pilots

All these services are necessary for the WMS. Each of them should be installed using the *system administrator console*. You can have several instances of each of them running, with the exclusion of the Matcher and the OptimizationMind [TBC].

Agents

SiteDirector send pilot jobs to Sites/CEs/Queues

JobCleaningAgent clean old jobs from the system

PilotStatusAgent update the status of the pilot jobs on the PilotAgentsDB

StalledJobAgent hunt 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.

All these agents are necessary for the WMS, and each of them should be installed using the *system administrator console*. You can duplicate some of these agents as long as you provide the correct configuration. A typical example is the SiteDirector, for which you may want to deploy even 1 for each of the sites managed.

Optional agents are:

StatesAccountingAgent or StatesMonitoringAgent produce monitoring plots then found in Accounting. Use one or the other.

A very different type of agent is the *JobAgent*, which is run by the pilot jobs and should NOT be run in a server installation.

Executors

Optimizer optimizers for jobs submission and scheduling. The 4 executors that are run are: InputData, JobPath, JobSanity, JobScheduling.

The optimizer executors are necessary for the WMS. They should be installed using the *system administrator console* and they can also be duplicated.

For detailed information on each of these components, please do refer to the WMS Code Documentation.

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 production version of pilots are sometimes dubbed as "Pilots 2.0", or "the pilots to fly in all the skies".

It's in pre-production a new generation of pilots, dubbed "Pilots 3". Pilots3 become, from version v6r20 of DIRAC, optional. Pilots3 development is done in the separate repository from that of DIRAC: https://github.com/DIRACGrid/Pilot The definitions that follow in this page are still valid for Pilots3. Some specific information about Pilots3 can be found in the next sections.

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.

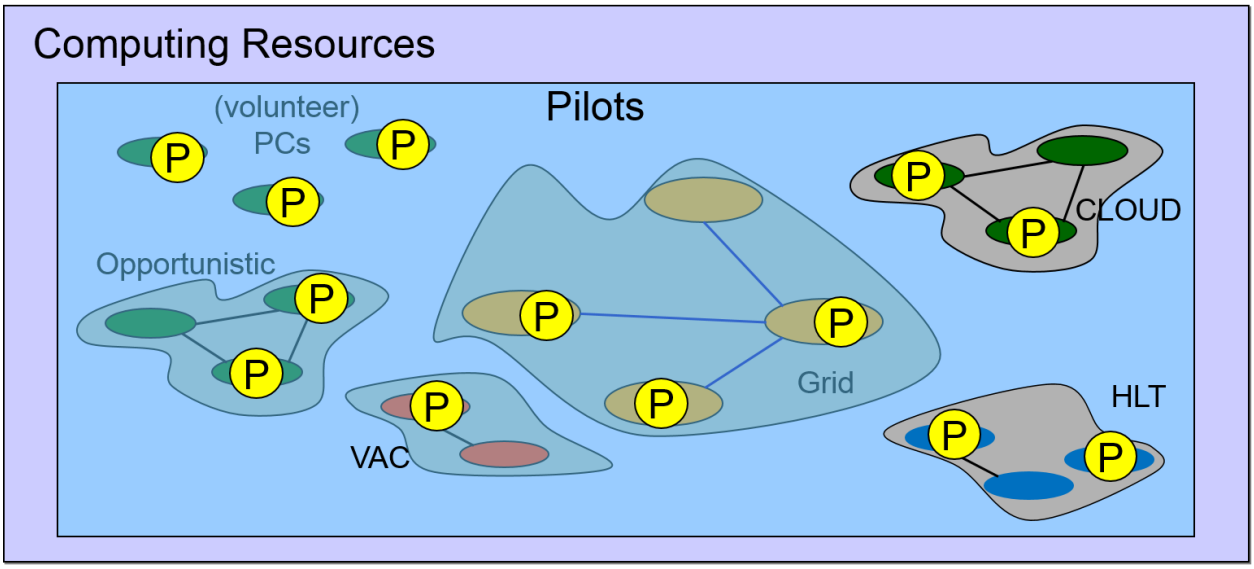
A pilot has, at a minimum, to:

- 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.



In more details the DIRAC WMS with Pilot Jobs is described here.

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 DIRAC pilots:

```
Operations/<Setup>/Pilot
```

These parameters will be interpreted by the WorkloadManagementSystem/SiteDirector agents, and by the WorkloadManagementSystem/Matcher. They can also be accessed by other services/agents, e.g. for syncing purposes.

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 = v6r20p14 #Version to install. Add the version of your extension if you have␣
↪one.
Project = myVO #Your project name: this will end up in the /LocalSite/ReleaseProject␣
↪option of the pilot cfg, and will be used at matching time
```

(continues on next page)

(continued from previous page)

```
Extensions = myVO #The DIRAC extension (if any)
Installation = mycfg.cfg #For an optional configuration file, used by the␣
↪installation script.
# For the Matcher
CheckVersion = False #True by default, if false any version would be accepted at␣
↪matching level (this is a check done by the WorkloadManagementSystem/Matcher␣
↪service).
```

When the *CheckVersion* option is "True", the version checking done at the Matcher level will be strict, which means that pilots running different versions from those listed in the *Versions* option will refuse to match any job. There is anyway the possibility to list more than one version in *Versions*; in this case, all of them will be accepted by the Matcher, and in this case the pilot will install the first in this list (e.g. if Version=v6r20p14,v6r20p13 then pilots will install version v6r20p14)

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/Pilot/blob/master/Pilot/pilotTools.py 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/master/WorkloadManagementSystem/PilotAgent for Pilot2, while the so-called "Pilot3" files are in the dedicated repository at https://github.com/DIRACGrid/Pilot/.

The main file in which you should look is 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:

```
#!/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
```

(continues on next page)

(continued from previous page)

```
    ;;
    --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 $0\``

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, that for Pilot3 are copied over to your WebPortal, e.g. like:
DIRAC_INSTALL='https://lhcb-portal-dirac.cern.ch/pilot/dirac-install.py'
DIRAC_PILOT='https://lhcb-portal-dirac.cern.ch/pilot/dirac-pilot.py'
DIRAC_PILOT_TOOLS='https://lhcb-portal-dirac.cern.ch/pilot/pilotTools.py'
DIRAC_PILOT_COMMANDS='https://lhcb-portal-dirac.cern.ch/pilot/pilotCommands.py'
LHCbDIRAC_PILOT_COMMANDS='https://lhcb-portal-dirac.cern.ch/pilot/LHCbPilotCommands.py'

#
##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 \
 --setup $LHCBDIRAC_SETUP \
 --project LHCb \
 --Name "$CE_NAME" \
 --name "$1" \
 --cert \
 --certLocation=/scratch/dirac/etc/grid-security \
```

DIRAC pilots 3

All concepts defined for Pilots 2 are valid also for Pilots3. There are anyway some differences for what regards their usage.

Bootstrap

Pilots3 need a JSON file for bootstrapping. We simply call this file the *pilot.json* file. The pilot.json file is created starting from information found in the Configuration Service.

The pilot wrapper (the script that starts the pilot, which is effectively equivalent to what SiteDirectors send) expects to find (download) such pilot.json file from a known location, which can be for example exposed by the DIRAC WebApp webserver.

The pilot.json file is therefore always kept in sync with the content of the Configuration Service. From DIRAC v6r20, there is the possibility to set the option *UpdatePilotCStoJSONFile* to True in the configuration of the Configuration/Server service (please see *Systems / Configuration / <INSTANCE> / Service / Server - Sub-subsection* for detais). If this option is set, at every configuration update, the pilot.json file content will also be updated (if necessary).

If *UpdatePilotCStoJSONFile* is True, then also the option *pilotFileServer* should be set to the webserver chosen for the upload. We suggest to use simply the DIRAC webserver.

Starting Pilots3 via SiteDirectors

New in version v6r20.

Since DIRAC v6r20, SiteDirectors can send "pilots2" or "pilots3". Pilots2 are the default, but the "Pilots3" flag can be used for sending Pilots3 files instead, see conf-SiteDirector

Pilot logging

Advanced pilot logging comes together with Pilots3. To enable... <to complete>

DIRAC jobs: definitions

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
- *multi-processor payload [job]*: a payload application that will try to use multiple cores on the same node
- *computing slot*: resource allocated by a provider where a pilot wrapper is running (batch job)
- *multi-processor [computing] slot*: allocated resource has more than one OS CPU core available in the same slot as opposed to a *single-processor [computing] slot*

Applications properties are reflected in payload properties.

The DIRAC APIs can be used to create and submit jobs. Specifically, objects of type Job represents a job. The API class Dirac and more specifically the call to submitJob submits jobs to the DIRAC WMS.

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 "UserPiorities" 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 witch 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:

```
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[3 - 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 it's 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:

```
hourCorrection = max ( min( hourCorrection, hourMax ), 1/hourMax )
weekCorrection = max ( min( weekCorrection, weekMax ), 1/weekMax )
finalCorrection = hourCorrection * hourWeight + weekCorrection * weekWeight
finalCorrection = max ( min( finalCorrection, globalMax ), 1/globalMax )
```

Matching WNs capabilities to Jobs requirements

Pilots determine the WNs capabilities and the JobAgent started by the pilot will contact the Matcher service to match a job, selected from the TaskQueueDB.

Capabilities and requirements include but are not limited to:

- *destination*: a (list of) site name(s)
- *CPUTime*: the (estimated) time, expressed in HS06s
- *platform*: the platform of the WN (which is determined by its OS, and not only), also refer to *Resources / Computing*
- *generic tags*: read about it in further sections

The JobAgent running on the Worker Node and started by the pilot presents capabilities in the form of a dictionary, like the following example:

```
{
  CPUTime:               1200000
  GridCE:                ce-01.somewhere.org
  GridMiddleware:        CREAM
  MaxRAM:                2048
  NumberOfProcessors:    1
  OwnerGroup:            diracAdmin,test,user
  PilotBenchmark:        19.5
  PilotInfoReportedFlag: False
  PilotReference:        https://ce-01.somewhere.org:8443/CREAM155256908
  Platform:              x86_64_glibc-2.12
  ReleaseProject:        VO
  ReleaseVersion:        v6r20p25
  Setup:                 VO-Certification
  Site:                  DIRAC.somewhere.org
  Tag:                   GPU
}
```

The WorkloadManagement/Matcher log will print out at the INFO log level dictionaries of capabilities presented to the service, like the example above. The matcher will try to match these capabilities to the requirements of jobs, which are stored in the MySQL DB TaskQueueDB.

An example of requirements include the following:

```
JobRequirements =
[
  OwnerDN = "/some/DN/";
  VirtualOrganization = "VO";
```

(continues on next page)

(continued from previous page)

```
  Setup = "VO-Certification";
  CPUTime = 17800;
  OwnerGroup = "user";
  UserPriority = 1;
  Sites = "DIRAC.somewhere.org";
  JobTypes = "User";
  Tags = "MultiProcessor";
];
```

which is what can be visualized in Job JDLs.

The generic Tags mechanism for jobs matching

DIRAC provides a generic mechanism for matching computing capabilities with resource providers, and this is done using generic "Tags". Tags can be used by the users to "mark (tag)" their jobs with requirements, and should be used by DIRAC admins to identify CEs or Queues.

So, as always it's a matter of what's written in the CS:

- Meaning that a CE or a Queue has **Tag=X** means that it's *capable() of running jobs that *require* **Tag=X**.
- Meaning that a CE or a Queue has **RequiredTag=X** means that it will *accept only* jobs that *require* **Tag=X**.

Let's take an example:

```
DIRAC.MySite.org
{
  Name = Test
  CEs
  {
    CE.MySite.org
    {
      CEType = Test
      Queues
      {
        # This queue exposes no Tags. So it will accept (match) all jobs that require␣
↪no tags
        noTagsQueue
        {
          # the following fields are not important
          SI00 = 2400
          maxCPUTime = 200
          MaxTotalJobs = 5
          MaxWaitingJobs = 10
          BundleProxy = True
          RemoveOutput = True
        }
        # This queue has Tag = GPU. So it will accept:
        # - jobs that require Tag = GPU (and no others)
        # - jobs that require no Tags
        GPUTagQueue
        {
          Tag = GPU
          ...
        }
        # This queue has Tag = [GPU, NVidiaGPU]. So it will accept:
```

(continues on next page)

(continued from previous page)

```
        # - jobs that require both the tags above (and no others)
        # - jobs that require Tag = GPU (and no others)
        # - jobs that require Tag = NVidiaGPU (and no others)
        # - jobs that require no Tags
        MultipleGPUTagQueue
        {
          Tag = GPU
          Tag += NVidiaGPU
          ...
        }
        # This queue has Tag = GPU and RequiredTag = GPU. So it will accept:
        # - jobs that require Tag = GPUs (and no others)
        RequiredGPUTagQueue
        {
          Tag = GPU
          RequiredTag = GPU
          ...
        }
        # This queue has Tag = [GPU, NVidiaGPU] and RequiredTag = GPU. So it will␣
↪accept:
        # - jobs that require both the tags above (and no others)
        # - jobs that require Tag = GPU (and no others)
        MultipleGPUTagQueue
        {
          Tag = GPU
          Tag += NVidiaGPU
          RequiredTag = GPU
          ...
        }
      }
    }
    # Tags can also be given to CEs. So, the following CE accepts ALSO GPU jobs.
    # The same examples above, which were done for the queues, apply also to CEs
    GPU-CE.cern.ch
    {
      Tag = GPU
      Queues
      {
        some_queue
        {
           ...
        }
      }
    }
  }
}
```

MultiProcessor Jobs

MultiProcessor (MP) jobs are a typical case of a type of jobs for which a complex matching is normally requested. There are several possible use cases. Starting from the case of the resource providers:

- computing resource providers may give their users the possibility to run on their resources only single processor jobs
- computing resource providers may give their users the possibility to run on their resources only multi processor

jobs

- computing resource providers may give their users the possibility to run both single and multi processor jobs
- computing resource providers may ask their users to distinguish clearly between single and multi processor jobs
- computing resource providers may need to know the exact number of processors a job is requesting

The configuration of the Computing Elements and the Job Queues that a computing resource provider expose will determine all the above. Within DIRAC it's possible to describe CEs and Queues precisely enough to satisfy all use cases above. It should also be remembered that, independently of DIRAC capabilities to accommodate all the cases above, normally, for a correct resource provisioning and accounting, computing resource providers don't allow multi processor payloads to run on single processor queues. And, sometimes they also don't allow single processor payloads to run on multi processor queues.

At the same time, from a users' perspective:

- certain jobs may be able to run only in single multi processor mode
- certain jobs may be able to run only in multi multi processor mode (meaning: need at least 2 processors)
- certain multi processor jobs may need a fixed amount of processors
- certain jobs may be able to run both in single or multi processor mode

Within DIRAC it's possible to describe the jobs precisely enough to satisfy all use cases above. For a description of how to use the DIRAC Job APIs for the use cases above, please refer to the *tutorial on Job Management*. This page explains how to configure the CEs and Queues for satisfying the use cases above, starting from the fact that single processor jobs are, normally, the default.

As of today (release v6r20p25) it's possible to use the tags mechanism (described in *The generic Tags mechanism for jobs matching*) for marking MultiProcessor jobs and queues (or CEs).

2.9.10 Monitoring System

Table of contents

Overview

The Monitoring system is used to monitor various components of DIRAC. Currently, we have two monitoring types:

- WMSHistory: for monitoring the DIRAC WMS
- Component Monitoring: for monitoring DIRAC components such as services, agents, etc.

It is based on Elasticsearch distributed search and analytics NoSQL database. If you want to use it, you have to install the Monitoring service and elasticsearch db. You can use a single node, if you do not have to store lot of data, otherwise you need a cluster (more than one node).

Install Elasticsearch

You can found in https://www.elastic.co official web site. I propose to use standard tools to install for example: yum, rpm, etc. otherwise you encounter some problems. If you are not familiar with managing linux packages, you have to ask your college or read some relevant documents.

Configure the MonitoringSystem

You can run your El cluster without authentication or using User name and password. You have to add the following parameters:

- User
- Password
- Host
- Port

The User name and Password must be added to the local cfg file while the other can be added to the CS using the Configuration web application. You have to handle the EL secret information in a similar way to what is done for the other supported SQL databases, e.g. MySQL

For example:

```
Systems
{
  NoSQLDatabases
  {
    User = test
    Password = password
  }

}
```

Enable WMSHistory monitoring

You have to install the WorkloadManagemet/StatesMonitoringAgent. This agent is used to collect information using the JobDB and send it to the Elasticsearch database. If you install this agent, you can stop the StatesAccounting agent.

Note: You can use RabbitMQ for failover. This is optional as the agent already has a failover mechanism. You can configure RabbitMQ in the local dirac.cfg file where the agent is running:

```
Resources
{
  MQServices
  {
    hostname (for example lbvobox10.cern.ch)
    {
      MQType = Stomp
      Port = 61613
      User = monitoring
```

(continues on next page)

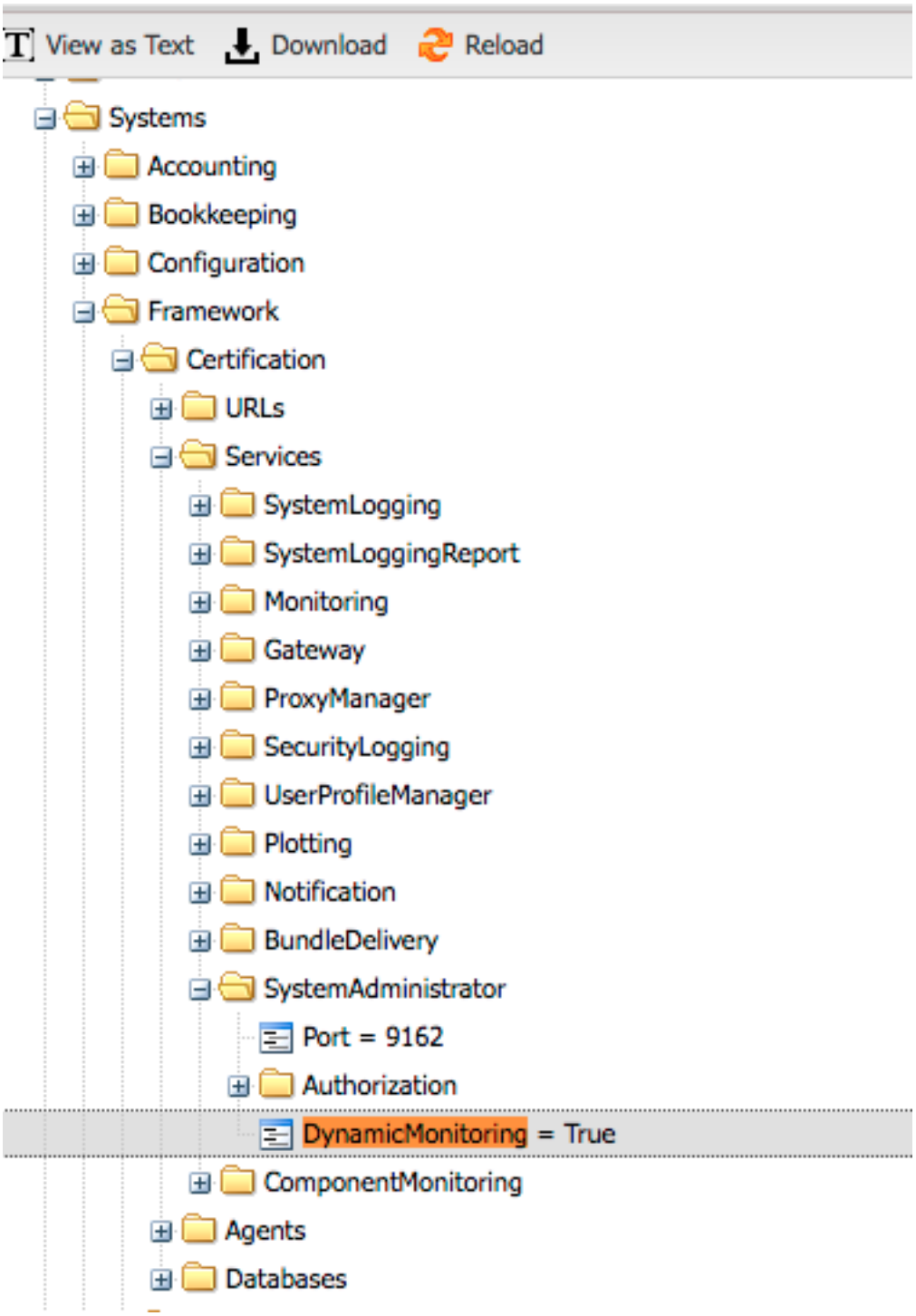
(continued from previous page)

```
      Password = seecret
      Queues
      {
        WMSHistory
        {
          Acknowledgement = True
        }
      }
    }
  }
}
```

Enable Component monitoring

You have to set DynamicMonitoring=True in the CS:

```
Systems
{
   Framework
   {
       SystemAdministrator
        {
           ...
           DynamicMonitoring = True
         }
      }
    }
```



Accessing the Monitoring information

After you installed and configured the Monitoring system, you can use the Monitoring web application.

2.9.11 Workflow

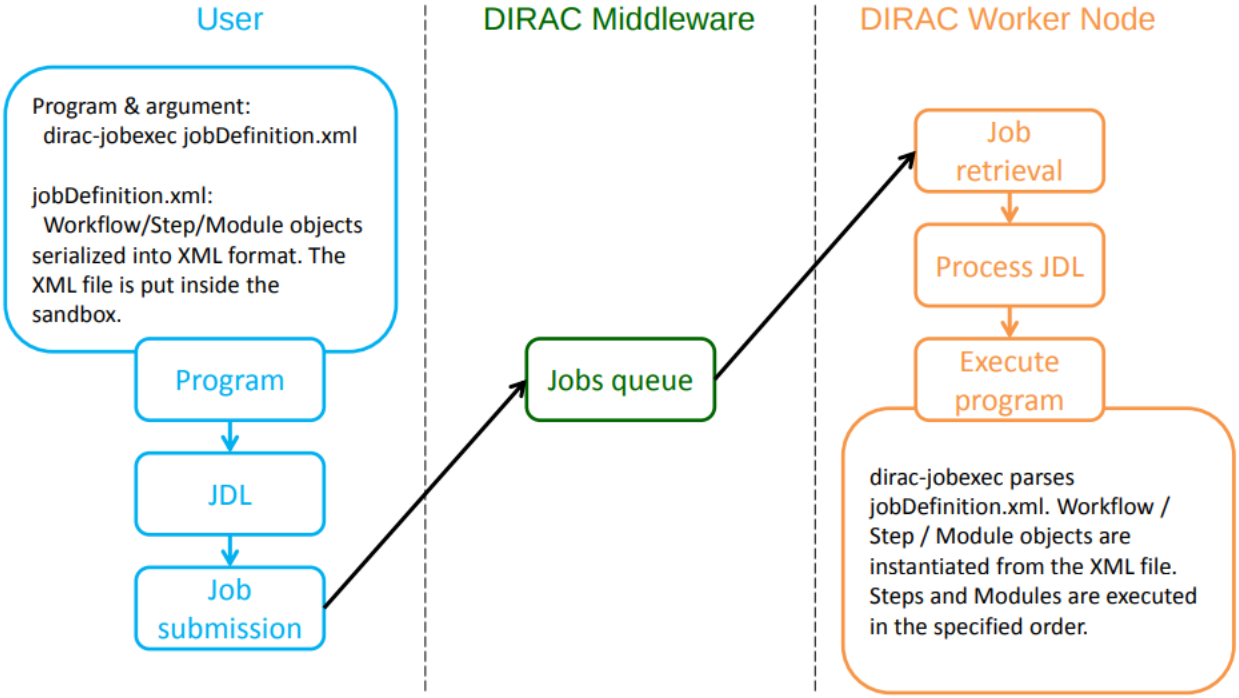
[Please see this presentation for originals of the figures reported here.]

The DIRAC Workflow is not properly a DIRAC system, in the sense that it doesn't implement any DB, nor service, nor agent.

The DIRAC Workflow is instead a way to describe DIRAC jobs. Every job created with DIRAC APIs is, in fact, a DIRAC workflow.

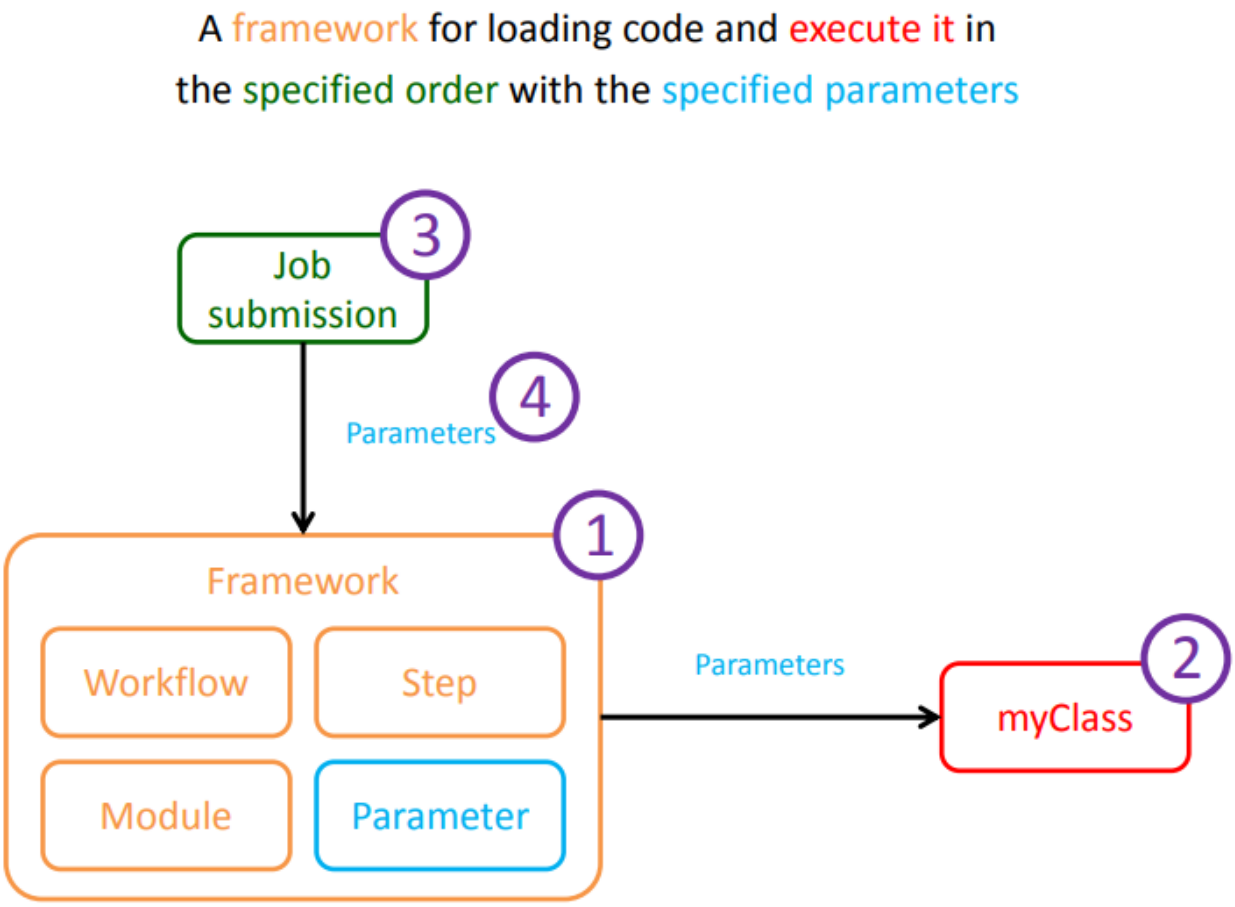
Using DIRAC workflows, users may load code and execute it in a specified order with specified parameters.

A DIRAC workflow is represented in XML format, but also in python format, meaning that DIRAC may access names and variables definded in the workflow directly from python. This may sound quite weird, or unclear, so let's go through a figure:



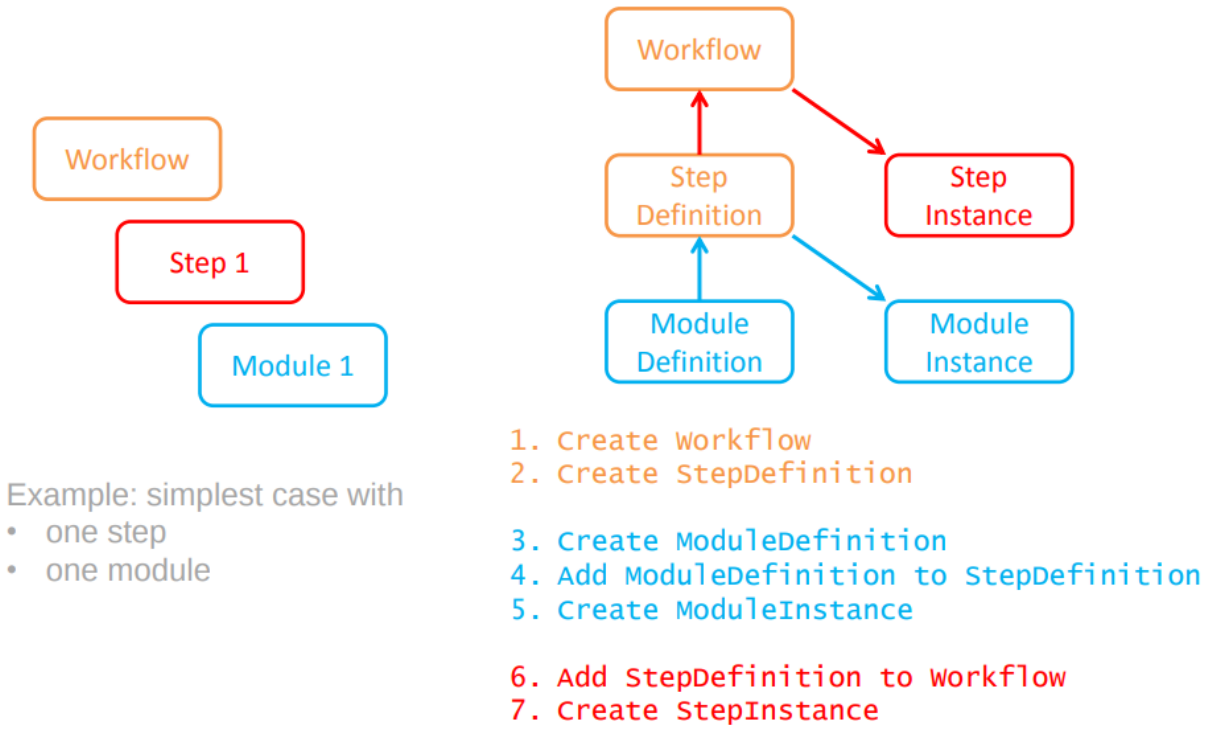
dirac-jobexec is the standard executable of DIRAC jobs. The jobDefinition.xml (usually called jobDescription.xml) is the workflow in XML format.

One more picture looking inside the contect of a workflow:

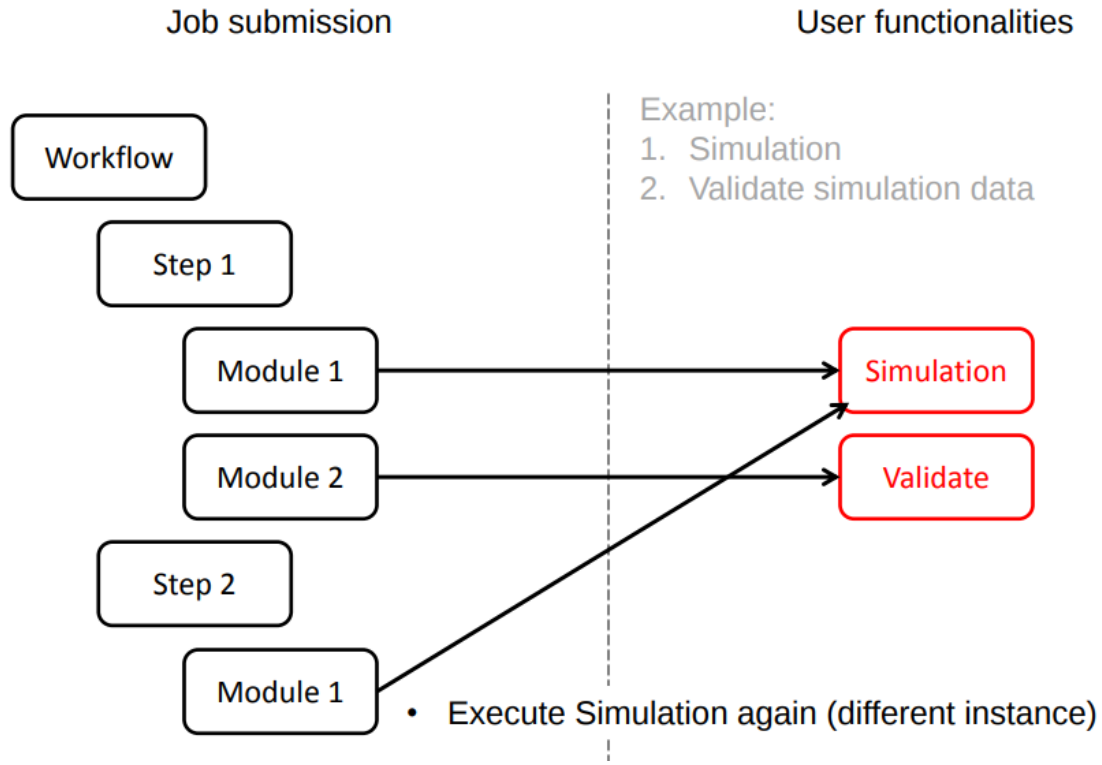


So, a workflow is made of steps, modules, and each of them may have parameters. A workflow is a container of steps. A step is a container for modules. Steps, and modules, are ordered. There's only a linear order possible (no DAGs). Workflows are not limited in the number of steps or modules that they can execute.

The following figure contains a graphical representation of the content of a workflow.



Modules are executed as python modules, when the job runs - i.e. the job executes the content of the workflow via dirac-jobexec, and the content of the workflow (the modules) is executed as standard python modules.



The package DIRAC/Workflow/Modules contain some base modules – the "Script" module can be used to execute any script or command.

Parameters can be added to Workflow, StepDefinition, StepInstance, ModuleDefinition and ModuleInstance. The Workflow framework is the product of a natural evolution from simple jobs to complex jobs.

2.10 Resources

2.10.1 Catalog

Catalogs represent the namespace in DIRAC. They are queried based on the LFN. Even if one is used as a reference (see *Master catalog*), you can use several catalogs in parallel. Every catalog has read and write methods.

The definition of catalogs is shared between two sections:

- */Resources/FileCatalogs*: this describes the catalog, how to access it, and all its options
- */Operations/<vo/setup>/Services/Catalogs/*: this describes how we use the catalog.

Resources

Every catalogs should be defined in the */Resources/FileCatalogs* section. You define one section per catalog. This section is supposed to describe how to access the catalog:

```
<catalogName>
{
  CatalogType = <myCatalogType>
  CatalogURL = <myCatalogURL>
  <anyOption>
}
```

- CatalogType: default *<catalogName>*

used to load the plugin located in *Resources.Catalog.<catalogType>Client*

- CatalogURL default *DataManagement/<CatalogType>*

  passed as *url* argument to the plugin in case it is an RPCClient

- <anyOption>

passed as keyed arguments to the constructor of your plugin.

For example:

```
Resources
{
   FileCatalogs
   {
     FileCatalog
     {
     }
     # This is not in DIRAC, just
     # another catalog
     BookkeepingDB
     {
       CatalogURL = Bookkeeping/BookkeepingManager
     }
   }
}
```

Operations

First of all, */Operations/<vo/setup>/Services/Catalogs/CatalogList* defines which catalogs are eligible for use. If this is not defined, we consider that all the catalogs defined under */Operations/<vo/setup>/Services/Catalogs/* are eligible.

Then, each catalog should have a few (case-sensitive) options defined:

- *Status*: (default *Active*). If anything else than *Active*, the catalog will not be used
- *AccessType*: *Read*/*Write*/*Read-Write*. No default, must be defined. This defines if the catalog is read-only, write only or both.
- *Master*: see *Master catalog*

For example:

```
Catalogs
{

   FileCatalog
   {
     AccessType = Read-Write
     Status = Active
     Master = True
   }
   # This is not in DIRAC, just
   # another catalog
   BookkeepingDB
   {
     AccessType = Write
     Status = Active
   }
}
```

Master catalog

When there are several catalogs, the write operations are not atomic anymore: the master catalog then becomes the reference. Any write operation is first attempted on the master catalog. If it fails, the operation is considered failed, and no attempt is done on the others. If it succedes, the other catalogs will be attempted as well, but a failure in one of the secondary catalogs is not considered as a complete failure. Of course, there should be only one master catalog

Conditional FileCatalogs

The *Status* and *AccessType* flags are global and binary. However it is sometimes a desirable feature to activate a catalog under some conditions only. This is what the conditional FCs are about. Conditions are evaluated for every catalog at every call and for every file. Conditions are defined in a section *Operation/<vo/setup>/Services/Catalogs/<CatalogName>/Conditions/*. They are evaluated by plugins, so it is very modular.

In this section, you can create an CS option for every method of your catalog. The name of the option should be the method name, and the value should be the condition to evaluate. If there are no condition defined for a given method, we check the global *READ*/*WRITE* condition, which are used for all read/write methods. If this does not exist either, we check the global *ALL* condition. If there are no condition at all, everything is allowed.

The conditions are expressed as boolean logic, where the basic bloc has the form *pluginName=whateverThatWillBePassedToThePlugin*. The basic blocs will be evaluated by the respective plugins, and the result can be combined using the standard boolean operators:

```
* ! for not
* & for and
* \| for or
* [ ] for prioritizing the operations
```

All these characters, as well as the '=' symbol cannot be used in any expression to be evaluated by a plugin.

Example of rules are:

```
* Filename=startswith('/lhcb') & Proxy=voms.has(/lhcb/Role->production)
* [Filename=startswith('/lhcb') & !Filename=find('/user/')] | Proxy=group.in(lhcb_mc,␣
↪lhcb_data)
```

The current plugins are:

- Filename: evaluation done on the LFN (`FilenamePlugin`)
- Proxy: evaluation done on the attributes of the proxy (user, group, VOMS role, etc) (`ProxyPlugin`)

2.10.2 RabbitMQ administration tools

RabbitMQ uses a two-step access-control(https://www.rabbitmq.com/access-control.html). Apart from the standard user/password (or ssl-based) authentication, RabbitMQ has an internal database with the list of users and permissions settings. DIRAC provides an interface to the internal RabbitMQ user database via the RabbitMQAdmin class. Internally it uses rabbitmqctl command (https://www.rabbitmq.com/man/rabbitmqctl.1.man.html) Only the user with the granted permissions can execute those commands. The interface provides methods for adding or removing users, setting the permission etc. The interface do not provide the possibilty to e.g. create or destroy queues, because according to the AMPQ and general RabbitMQ philosophy those operations should be done by consumers/producer with given permissions.

2.10.3 Synchronization of RabbitMQ user database

The synchronization between the DIRAC Configuration System and the RabbitMQ internal database is assured by RabbitMQSynchronizer. It checks the current list of users and hosts which are allowed to send messages to RabbitMQ and updates the internal RabbitMQ database accordingly.

2.10.4 StorageElement

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:

```
CERN-USER
{
  OccupancyLFN = /lhcb/spaceReport.json
  ReadAccess = Active
  WriteAccess = Active
  AccessProtocol.1
  {
    # The name of the DIRAC Plugin module to be used for implementation
    # of the access protocol
    PluginName = SRM2
```

(continues on next page)

(continued from previous page)

```
    # Flag specifying the access type (local/remote)
    Access = remote
    # Protocol name
    Protocol = srm
    # Host endpoint
    Host = srm-lhcb.cern.ch
    Port = 8443
    # WSUrl part of the SRM-type PFNs
    WSUrl = /srm/managerv2?SFN=
    # Path to navigate to the VO namespace on the storage
    Path = /castor/cern.ch/grid
    # SRM space token
    SpaceToken = LHCb_USER
    # VO specific path definitions
    VOPath
    {
      biomed = /castor/cern.ch/biomed/grid
    }
  }
}
```

Configuration options are:

- *BackendType*: just used for information. No internal use at the moment
- *SEType*: Can be *T0D1* or *T1D0* or *T1D1*. it is used to asses whether the SE is a tape SE or not. If the digit after *T* is *1*, then it is a tape.
- *UseCatalogURL*: default *False*. If *True*, use the url stored in the catalog instead of regenerating it
- *ChecksumType*: default *ADLER32*. NOT ACTIVE !
- *Alias*: when set to the name of another storage element, it instanciates the other SE instead.
- *ReadAccess*: default *True*. Allowed for Read if no RSS enabled (*Activate RSS*)
- *WriteAccess*: default *True*. Allowed for Write if no RSS enabled
- *CheckAccess*: default *True*. Allowed for Check if no RSS enabled
- *RemoveAccess*: default *True*. Allowed for Remove if no RSS enabled
- *OccupancyLFN*: default (*/<vo>/occupancy.json*). LFN where the json file containing the space reporting is to be found

VO specific paths

Storage Elements supporting multiple VO's can have definitions slightly differing with respect to the *Path* used to navigate to the VO specific namespace in the physical storage. If a generic *Path* can not be suitable for all the allowed VO's a *VOPath* section can be added to the Plugin definition section as shown in the example above. In this section a specific *Path* can be defined for each VO which needs it.

2.10.5 StorageElementBases

Installations tend to have several StorageElements, with very similar configurations (e.g., the same Host and Port). It could be useful to factorize the SEs configuration to avoid repeating it. In order to factorize the configuration, it is possible to use *BaseSE*, which acts just like inheritance in object programming. You define a SE just like any other

but in the *StorageElementBases* section. This SE can then be refered to by another SE. This new SE will inherit all the configuration from its parents, and can override it. For example:

```
StorageElementBases
{
  CERN-EOS
  {
    BackendType = Eos
    SEType = T0D1
    AccessProtocol.1
    {
      Host = srm-eoslhcb.cern.ch
      Port = 8443
      PluginName = GFAL2_SRM2
      Protocol = srm
      Path = /eos/lhcb/grid/prod
      Access = remote
      SpaceToken = LHCb-EOS
      WSUrl = /srm/v2/server?SFN=
    }
  }
}
StorageElements
{
  CERN-DST-EOS
  {
    BaseSE = CERN-EOS
  }
  CERN-USER
  {
    BaseSE = CERN-EOS
    PledgedSpace = 205
    AccessProtocol.1
    {
      PluginName = GFAL2_SRM2
      Path = /eos/lhcb/grid/user
      SpaceToken = LHCb_USER
    }
  }
  GFAL2_XROOT
  {
    Host = eoslhcb.cern.ch
    Port = 8443
    Protocol = root
    Path = /eos/lhcb/grid/user
    Access = remote
    SpaceToken = LHCb-EOS
    WSUrl = /srm/v2/server?SFN=
  }
}
```

This definition would be strictly equivalent to:

```
StorageElementBases
{
  CERN-EOS
  {
    BackendType = Eos
```

(continues on next page)

(continued from previous page)

```
    SEType = T0D1
    AccessProtocol.1
    {
      Host = srm-eoslhcb.cern.ch
      Port = 8443
      PluginName = GFAL2_SRM2
      Protocol = srm
      Path = /eos/lhcb/grid/prod
      Access = remote
      SpaceToken = LHCb-EOS
      WSUrl = /srm/v2/server?SFN=
    }
  }
}
StorageElements
{
  CERN-DST-EOS
  {
    BackendType = Eos
    SEType = T0D1
    AccessProtocol.1
    {
      Host = srm-eoslhcb.cern.ch
      Port = 8443
      PluginName = GFAL2_SRM2
      Protocol = srm
      Path = /eos/lhcb/grid/prod
      Access = remote
      SpaceToken = LHCb-EOS
      WSUrl = /srm/v2/server?SFN=
    }
  }
  CERN-USER
  {
    BackendType = Eos
    SEType = T0D1
    PledgedSpace = 205
    AccessProtocol.1
    {
      Host = srm-eoslhcb.cern.ch
      Port = 8443
      PluginName = GFAL2_SRM2
      Protocol = srm
      Path = /eos/lhcb/grid/user
      Access = remote
      SpaceToken = LHCb_USER
      WSUrl = /srm/v2/server?SFN=
    }
  }
  GFAL2_XROOT
  {
    Host = eoslhcb.cern.ch
    Port = 8443
    PluginName =  GFAL2_XROOT
    Protocol = root
    Path = /eos/lhcb/grid/user
    Access = remote
```

(continues on next page)

(continued from previous page)

```
    SpaceToken = LHCb-EOS
    WSUrl = /srm/v2/server?SFN=
  }
}
```

Note that base SE must be separated from the inherited SE in two different sections. You can also notice that the name of the protocol section can be a plugin name. In this way, you do not need to specify a plugin name inside.

Available protocol plugins

DIRAC comes with a bunch of plugins that you can use to interact with StorageElements. These are the plugins that you should define in the *PluginName* option of your StorageElement definition.

- DIP: used for dips, the DIRAC custom protocol (useful for example for DIRAC SEs).
- File: offers an abstraction of the local access as an SE.
- SRM2 (deprecated): for the srm protocol, using the deprecated gfal libraries.
- RFIO (deprecated): for the rfio protocol.
- Proxy: to be used with the StorageElementProxy.
- XROOT (deprecated): for the xroot protocol, using the python xroot binding (http://xrootd.org/doc/python/xrootd-python-0.1.0/#).

There are also a set of plugins based on the gfal2 libraries (https://dmc.web.cern.ch/projects).

- GFAL2_SRM2: for srm, replaces SRM2
- GFAL2_XROOT: for xroot, replaces XROOT
- GFAL2_HTTPS: for https
- GFAL2_GSIFTP: for gsiftp

Default plugin options:

- *Access*: *Remote* or *Local*. If *Local*, then this protocol can be used only if we are running at the site to which the SE is associated. Typically, if a site mounts the storage as NFS, the *file* protocol can be used.

Space occupancy

Several methods allow to know how much space is left on a storage, depending on the protocol:

- dips: a simple system call returns the space left on the partition
- srm: the srm is able to return space occupancy based on the space token
- any other: a generic implementation has been made in order to retrieve a JSON file containing the necessary information.

A WLCG working group is trying to standardize the space reporting. So a standard will probably emerge soon (before 2053). For the time being, we shall consider that the JSON file will contain a dictionary with keys *Total* and *Free* in Bytes. For example:

```
{
  "Total": 20,
  "Free": 10
}
```

The LFN of this file is by default */<vo>/occupancy.json*, but can be overwritten with the *OccupancyLFN* option of the SE.

Multi Protocol

There are several aspects of multi protocol:

- One SE supports several protocols
- SEs with different protocols need to interact
- We want to use different protocols for different operations

DIRAC supports all of them. The bottom line is that before executing an action on an SE, we check among all the plugins defined for it, which plugins are the most suitable. There are 4 Operation options under the *DataManagement* section used for that:

- *RegistrationProtocols*: used to generate a URL that will be stored in the FileCatalog
- *AccessProtocols*: used to perform the read operations
- *WriteProtocols*: used to perform the write and remove operations
- *ThirdPartyProtocols*: used in case of replications

When performing an action on an SE, the StorageElement class will evaluate, based on these lists, and following this preference order, which StoragePlugins to use.

The behavior is straightforward for simple read or write actions. It is however a bit more tricky when it comes to third party copies.

Each StoragePlugins has a list of protocols that it is able to accept as input and a list that it is able to generate. In most of the cases, for protocol X, the plugin is able to generate URL for the protocol X, and to take as input URL for the protocol X and local files. There are plugins that can do more, like GFAL2_SRM2 plugins that can handle many more (xroot, gsiftp, etc). It may happen that the SE can be writable only by one of the protocol. Suppose the following situation: you want to replicate from storage A to storage B. Both of them have as plugins GFAL2_XROOT and GFAL2_SRM2; AccessProtocols is “root,srm”, WriteProtocols is “srm” and ThirdPartyProtocols is “root,srm”.

The negociation between the storages to find common protocol for third party copy will lead to “root,srm”. Since we follow the order, the sourceURL will be a root url, and it will be generated by GFAL2_XROOT because root is its native protocol (so we avoid asking the srm server for a root turl). The destination will only consider using GFAL2_SRM2 plugins because only srm is allowed as a write plugin, but since this plugins can take root URL as input, the copy will work.

The WriteProtocols and AccessProtocols list can be locally overwritten in the SE definition.

Multi Protocol with FTS

External services like FTS requires pair of URLs to perform third party copy. This is implemented using the same logic as described above. There is however an extra step: once the common protocols between 2 SEs have been filtered, an extra loop filter is done to make sure that the selected protocol can be used as read from the source and as write to the destination. Finally, the URLs which are returned are not necessarily the url of the common protocol, but are the native urls of the plugin that can accept/generate the common protocol. For example, if the common protocol is gsiftp but one of the SE has only an SRM plugin, then you will get an srm URL (which is compatible with gsiftp).

Protocol matrix

In order to make it easier to debug, the script dirac-dms-protocol-matrix will generate a CSV files that allows you to see what would happen if you were to try transfers between SEs

2.10.6 StorageElementGroups

StorageElements can be grouped together in a *StorageElementGroup*. This allows the systems or the users to refer to *any storage within this group*.

2.11 Managing Sites and Resources in DIRAC

A Site, in DIRAC, is the entity that collects access points to resources that are related by locality in a functional sense, i.e. the storage at a given Site is considered local to the CPU at the same Site and this relation will be used by DIRAC. On the other hand, a Site must provide a single entry point responsible for the availability of the resources that it encompasses. In the DIRAC sense, a Site can be from a fraction of physical computer center, to a whole regional grid. It is the responsibility of the DIRAC administrator of the installation to properly define the sites. Not all Sites need to grant access to all VOs supported in the DIRAC installation.

DIRAC can incorporate resources provided by existing Grid infrastructures (e.g. WLCG, OSG) as well as sites not integrated in any grid infrastructure, but still contributing with their computing and storage capacity, available as conventional clusters or file servers.

2.11.1 Site Names

In the *DIRAC configuration Sites* have names resulting from concatenation of the Domain prefix, the name of the Site and the country (or the funding body), according to the ISO 3166 standard with a dot as a separator. The full DIRAC Site Name becomes of the form: [Domain].[Site].[co]. The full site names are used everywhere when the site resources are assigned to the context of a particular Domain: in the accounting, monitoring, configuration of the Operations parameters, etc.

Examples of valid site names are:

- LCG.CERN.ch
- CLOUD.IN2P3.fr
- VAC.Manchester.uk
- DIRAC.farm.cern

The [Domain] may imply a (set of) technologies used for exploiting the resources, even though this is not necessarily true. The use of these Domains is mostly for reporting purposes, and it is the responsibility of the administrator of the DIRAC installation to chose them in such a way that they are meaningful for the communities and for the computing resources served by the installation. In any case, DIRAC will always be a default Domain if nothing else is specified for a given resource.

The Domain, Site and the country must be unique alphanumeric strings, irrespective of case, with a possible use of the following characters: "_".

Sites are providing access to the resources, therefore the /Resources/Sites section is the main place where the resources description is stored. Resource types may include:

- Computing (via Computing Elements, "CE")
- Storage (via Storage Elements, "SE")

- Message Queues

The following sections will focus on other types of resources: Computing Elements (CEs), Storage Elements (SEs), Message Queues (MQs).

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 part of the agents of the Workload Management System, and can't work alone. Please refer to *documentation of the WMS* for info about the other WMS components.

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

For detailed information on the CS configuration of the SiteDirector, please refer to the WMS Code Documentation.

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:

```
Resources
{
  Sites
  {
    # Site administrative domain
    LCG
    {
```

(continues on next page)

(continued from previous page)

```
      # Site section
      LCG.CNAF.it
      {
        # Site name
        Name = CNAF

        # List of valid CEs on the site
        CE = ce01.infn.it, ce02.infn.it

        # Section describing each CE
        CEs
        {
          # Specific CE description section
          ce01.infn.it
          {
            # Type of the CE
            CEType = CREAM

            # 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
            SubmissionMode = direct

            # Section to describe various queue in the CE
            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.

Additional info can be found *here*.

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 describe in the subsections below

CREAM Computing Element

A commented example follows:

```
# 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
      CPUTime = 10000
      # Max total number of jobs in the queue
      MaxTotalJobs = 5
      # Max number of waiting jobs in the queue
      MaxWaitingJobs = 2
    }
  }
}
```

SSH Computing Element

The SSHComputingElement is used to submit pilots through an SSH tunnel to computing clusters with various batch systems. A commented example of its configuration follows

```
# Section placed in the */Resources/Sites/<domain>/<site>/CEs* directory
pc.farm.ch
{
  CEType = SSH
  # Type of the local batch system. Available batch system implementations are:
  # Torque, Condor, GE, LSF, OAR, SLURM
  BatchSystem = Torque
  SubmissionMode = direct
  SSHHost = pc.domain.ch
  # SSH connection details to be defined in the local configuration
  # of the corresponding SiteDirector
  SSHUser = dirac_ssh
  SSHPassword = XXXXXXX
  # Alternatively, the private key location can be specified instead
  # of the SSHPassword
  SSHKey = /path/to/the/key
  # SSH port if not standard one
  SSHPort = 222
  # Sometimes we need an extra tunnel where the batch system is on accessible
  # directly from the site gateway host
  SSHTunnel = ssh pcbatch.domain.ch
  # SSH type: ssh (default) or gsissh
  SSHType = ssh
  # Options to SSH command
  SSHOptions = -o option1=something -o option2=somethingelse
  # Queues section contining queue definitions
  Queues
  {
```

(continues on next page)

(continued from previous page)

```
    # 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 qsub job submission command
      SubmitOptions =
      # 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:

```
# Section placed in the */Resources/Sites/<domain>/<site>/CEs* directory
pc.farm.ch
{
  CEType = SSHBatch
  SubmissionMode = direct

  # Parameters of the SSH conection to the site. The /2 indicates how many cores can␣
↪be used on that host.
  # 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 = XXXXXXXXX
  # 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=somethingelse

  Queues
  {
```

(continues on next page)

(continued from previous page)

```
    # Similar to the corresponding SSHComputingElement section
  }
}
```

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 does 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 explicitely the options to ssh with the SSHOptions option. For example:

```
SSHOptions = -o UserKnownHostsFile=/local/path/to/known_hosts
```

allows to have a local copy of the `known_hosts` file, independent of the HOME directory.

InProcessComputingElement

The InProcessComputingElement is usually invoked by a JobAgent to execute user jobs in the same process as the one of the JobAgent. Its configuration options are usually defined in the local configuration /Resources/Computing/CEDefaults section

```
Resources
{
  Computing
  {
    CEDefaults
    {
      NumberOfProcessors = 2
      Tag = MultiProcessor
      RequiredTag = MultiProcessor
    }
  }
}
```

PoolComputingElement

The Pool Computing Element is used on multi-processor nodes, e.g. cloud VMs and can execute several user payloads in parallel using an internal ProcessPool. Its configuration is also defined by pilots locally in the /Resources/Computing/CEDefaults section

```
Resources
{
  Computing
  {
    CEDefaults
    {
      NumberOfProcessors = 2
      Tag = MultiProcessor
      RequiredTag = MultiProcessor
      # The MultiProcessorStrategy flag defines if the Pool Computing Element
      # will generate several descriptions to present possibly several queries
      # to the Matcher in each cycle trying to select multi-processor jobs first
```

(continues on next page)

(continued from previous page)

```
      # and, if no match found, simple jobs finally
      MultiProcessorStrategy = True
    }
  }
}
```

Storage Elements

A named SE is an SE in the DIRAC sense, in other words, as seen by the DIRAC users.

Different named SEs can point to the same StorageElement server, and make use of different options to upload/retrieve data from different backend storages. For instance a different base path or a different SRM Space Token for different types of data.

In general the SE name is a logical name and not a hostname.

Detailed information about SEs can be found *here*.

Message Queues

Message Queues are services for passing messages between DIRAC components. These services are not part necessarily of the DIRAC software and are provided by third parties. Access to the services is done via logical Queues (or Topics). Queues and Topics are two popular variation of the MQ communication model. In the first case, the messages from the queue are typically delivered to the subscribed consumers one by one. One message will be received by exactly one consumer. If no consumer is available, then the messages are stored in the queue. Many consumers connected to the same queue can be used for the load balancing purposes. The topic architecture can be see as implementation of the publish-subscribe pattern. The messages are typically grouped in categories (e.g. by assigning the label called topic), and consumers subscribe to chosen topics. When the message becomes available, it is send to all subscribed consumers. Detailed implementation of Topic/Queue mechanism can differ dependent e.g. MQ broker used.

The available implementation of the Message Queue uses Stomp protocol. All the Stomp-dependent details are encapsulated in StompMQConnector class, which extends the generic MQConnector class. It is possible to provide a self-defined connector by extending the MQConnector class.

A commented example of the Message Queues configuration is provided below. Each option value is representing its default value:

```
Resources
{
  # General section for all the MessageQueue service. Each subsection is
  # dedicated to a particular MQ server
  MQServices
  {
    # MQ server section. The name of the section is arbitrary, not necessarily
    # the host name
    mardirac3.in2p3.fr
    {
      # The MQ type defines the protocol by which the service is accessed.
      # Currently only Stomp protocol is available. Mandatory option
      MQType = Stomp
      # The MQ server host name
      Host = mardirac3.in2p3.fr
      # The MQ server port number
```

(continues on next page)

(continued from previous page)

```
      Port = 9165
      # Virtual host
      VHost = /
      # User name to access the MQ server (not needed if you are using SSL␣
↪authentication)
      User = guest
      # Password to access the MQ server. (not needed if you are using SSL␣
↪authentication)
      # This option should never be defined
      # in the Global Configuration, only in the local one
      Password = guest
      # if SSLVersion is set, then you are connecting using a certificate host/key␣
↪pair
      # You can also provide a location for the host/key certificates with the options
      # "HostCertificate" and "HostKey" (which take a path as value)
      # and when these options are not set, the standard DIRAC locations will be used
      SSLVersion = TLSv1
      # General section containing subsections per Message Queue. Multiple Message
      # Queues can be defined by MQ server
      Queues
      {
        # Message Queue section. The name of the section is defining the name
        # of the Message Queue
        TestQueue
        {
          # Option defines if messages reception is acknowledged by the listener
          Acknowledgement = True
          # Option defines if the Message Queue is persistent or not
          Persistent = False
        }
      }
    }
  }
}
```

Once Message Queues are defined in the configuration, they can be used in the DIRAC codes like described in *Message Queues*, for example:

```
from DIRAC.Resources.MessageQueue.MQCommunication import createProducer
from DIRAC.Resources.MessageQueue.MQCommunication import createConsumer

result = createProducer( "mardirac3.in2p3.fr::Queues::TestQueue" )
if result['OK']:
   producer = result['Value']

result = createConsumer( "mardirac3.in2p3.fr::Queues::TestQueue" )
if result['OK']:
   consumer = result['Value']

result = producer.put( message )
result = consumer.get( message )
if result['OK']:
  message = result['Value']
```

In order not to spam the logs, the log output of Stomp is always silence, unless the environment variable *DIRAC_DEBUG_STOMP* is set to any value.

Message Queue nomenclature in DIRAC

- MQ - Message Queue System e.g. RabbitMQ
- mqMessenger - processes that send or receive messages to/from the MQ system. We define two types of messengers: consumer (MQConsumer class) and producer (MQProducer class).
- mqDestination is the endpoint of MQ systems. We define two kind of destinations: Queue or Topic. which correspond to two type of communication schemes between MQ and consumers/producers.
- mqService - unique identifier that characterises an MQ resource in the DIRAC CS. mqService can have one or more topics and/or queues assigned.
- mqConnection: authenticated link between an MQ and one or more producers or/and consumers. The link can be characterised by mqService.
- mqURI - pseudo URI identifier that univocally identifies the destination. It has the following format mqService::mqDestinationType::mqDestination name e.g."mardirac3.in2p3.fr::Queues::TestQueue" or "mardirac3.in2p3.fr::Topics::TestTopic".
- mqType - type of the MQ communication protocol e.g. Stomp.
- MQConnector - provides abstract interface to communicate with a given MQ system. It can be specialized e.g. StompMQConnector.

2.12 Multi-VO DIRAC

author Bruno Santeramo <bruno.santeramo at ba.infn.it> - Federico Stagni (fstagni at cern.ch)

date 05/2013 - small update 03/2018

version 1.1

In this chapter a guide to install and configure DIRAC for multi-VO usage.

Table of contents

2.12.1 Before starting 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
- for each VO a <vo_name>_user group is configured to allow normal user operations

Limits to 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.

2.12.2 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:

```
...
#  VO name (not mandatory, useful if DIRAC will be used for a VO)
#VirtualOrganization = superbvo.org
...
```

2.12.3 DIRAC client installation

Second step is to install a dirac client and configure it for new installation.

2.12.4 Configuring first VO (e.g. superbvo.org)

Registry

Add superb_user group

```
Registry
{
  DefaultGroup = superb_user
}
```

Registry/VO

```
Registry
{
  VO
  {
    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
        }
      }
    }
  }
}
```

Registry/Groups

Here define the users part of the "superb_user" group, its DIRAC properties, and its VOMS properties.

```
Registry
{
  Groups
  {
    superb_user
    {
      Users = bsanteramo, anotherUser
      Properties = NormalUser
      VOMSRole = /superbvo.org
      VOMSVO = superbvo.org
      VO = 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
        }
      }
    }
  }
}
```

$HOME/.glite/vomses

DIRAC search for VOMS data in the directory pointed by `$DIRAC_VOMSES` variable. If this is not present, the default directory is `$DIRAC/etc/grid-security/vomses`

For each VO, there should be a file with the same name of VO and filled it the following 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" "8443"
"superbvo.org" "voms-02.pd.infn.it" "15009" "/C=IT/O=INFN/OU=Host/L=Padova/CN=voms-02.
↪pd.infn.it" "superbvo.org" "8443"
```

If your VO is not present, you can add the file by hand.

Operations - Shifter

```
Operations
{
  SuperB-Production
  {
    Shifter
    {
      ProductionManager
```

(continues on next page)

(continued from previous page)

```
      {
        User = bsanteramo
        Group = superb_user
      }
      DataManager
      {
        User = bsanteramo
        Group = superb_user
      }
    }
  }
}
```

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
```

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)

2.12.5 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
    {
      VOAdmin = bsanteramo
      VOMSName = pamela
      VOMSServers
        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
      AutoAddVOMS = True
      AutoUploadProxy = True
      AutoUploadPilotProxy = True
    }
  }
}
```

Registry/VOMS

Add pamela parameters...

```
Registry
{
  VOMS
  {
    Mapping
    {
      pamela_user = /pamela
    }
    Servers
    {
      pamela
      {
        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
        }
      }
    }
  }
}
```

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
    {
      ProductionManager
      {
        User = bsanteramo
        Group = superb_user
      }
      DataManager
      {
        User = bsanteramo
        Group = superb_user
      }
    }
  }
  JobDescription
  {
    AllowedJobTypes = MPI
    AllowedJobTypes += User
    AllowedJobTypes += Test
  }
  pamela
  {
    SuperB-Production
    {
      Shifter
      {
        ProductionManager
        {
          User = bsanteramo
          Group = pamela_user
        }
        DataManager
        {
          User = bsanteramo
          Group = pamela_user
        }
      }
    }
  }
}
```

2.13 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:

2.13.1 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}
```

2.13.2 dirac-platform

The *dirac-platform* script determines the "platform" of a certain node. The platform is a string used to identify the minimal characteristics of the node, enough to determine which version of DIRAC can be installed.

Invoked at any installation, so by the *dirac-install* script, and by the pilots.

On a RHEL 6 node, for example, the determined dirac platform is “Linux_x86_64_glibc-2.5”

Example:

```
$ dirac-platform
Linux_x86_64_glibc-2.5
```

Managing Registry:

2.13.3 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
$
```

2.13.4 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:

```
$ dirac-admin-add-host -H dirac.i2np3.fr -D /O=GRID-FR/C=FR/O=CNRS/OU=CC-IN2P3/
↪CN=dirac.in2p3.fr
```

2.13.5 dirac-admin-add-user

Add or Modify a User info in DIRAC

Usage:

```
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:

```
$ 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
$
```

2.13.6 dirac-admin-delete-user

Remove User from Configuration

Usage:

```
dirac-admin-delete-user [option|cfgfile] ... User ...
```

Arguments:

```
User:     User name
```

Example:

```
$ dirac-admin-delete-user vhamar
```

2.13.7 dirac-admin-list-hosts

Usage:

```
dirac-admin-list-hosts.py (<options>|<cfgFile>)*
```

Options:

```
-e  --extended        : Show extended info
```

Example:

```
$  dirac-admin-list-hosts
dirac.in2p3.fr
host-dirac.in2p3.fr
```

2.13.8 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
```

2.13.9 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
```

2.13.10 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:

```
-t   --test             : Only test. Don't commit changes
```

Example:

```
$ dirac-admin-sync-users-from-file file_users.cfg
```

2.13.11 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
------------------------------
Username        |     Quota (GB)
------------------------------
atsareg         |           None
msapunov        |           None
vhamar          |           None
------------------------------
```

2.13.12 dirac-admin-users-with-proxy

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

```
-v:  --valid=          : Required HH:MM for the users
```

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

```
-v:  --valid=          : Required HH:MM for the users
```

Usage:

```
dirac-admin-users-with-proxy.py (<options>|<cfgFile>)*
```

Options:

```
-v:  --valid=          : Required HH:MM for the users
```

Example:

```
$ dirac-admin-users-with-proxy
* vhamar
DN          : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
group       : dirac_admin
not after   : 2011-06-29 12:04:25
persistent  : False
-
DN          : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
group       : dirac_pilot
not after   : 2011-06-29 12:04:27
persistent  : False
-
DN          : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar
group       : dirac_user
not after   : 2011-06-29 12:04:30
persistent  : True
```

Managing Resources:

2.13.13 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)
```

(continues on next page)

(continued from previous page)

```
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
```

2.13.14 dirac-admin-allow-catalog

Enable usage of the File Catalog mirrors at given sites

Usage:

```
dirac-admin-allow-catalog site1 [site2 ...]
```

2.13.15 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
$
```

2.13.16 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'
```

2.13.17 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
```

2.13.18 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
```

2.13.19 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'
```

2.13.20 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
```

2.13.21 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(ie: ce111.cern.ch)
```

Options:

```
-H:  --host=           : BDII host

-V:  --vo=             : vo
```

Example:

```
$ dirac-admin-bdii-ce-voview LCG.IN2P3.fr
```

2.13.22 dirac-admin-bdii-ce

Check info on BDII for CE

Usage:

```
dirac-admin-bdii-ce [option|cfgfile] ... CE
```

Arguments:

```
CE:       Name of the CE(ie: ce111.cern.ch)
```

Options:

```
-H:  --host=           : BDII host
```

Example:

```
$ dirac-admin-bdii-ce LCG.IN2P3.fr
```

2.13.23 dirac-admin-bdii-cluster

Check info on BDII for Cluster

Usage:

```
dirac-admin-bdii-cluster [option|cfgfile] ... CE
```

Arguments:

```
CE:       Name of the CE(ie: ce111.cern.ch)
```

Options:

```
-H:  --host=           : BDII host
```

Example:

```
$ dirac-admin-bdii-cluster LCG.IN2P3.fr
```

2.13.24 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
```

2.13.25 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
```

2.13.26 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
```

2.13.27 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
```

2.13.28 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
```

2.13.29 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:

```
$ 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
```

2.13.30 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
```

2.13.31 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, sbgce1.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'}
```

2.13.32 dirac-admin-site-mask-logging

Retrieves site mask logging information.

Usage:

```
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 )  ""
```

Workload management commands:

2.13.33 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
```

2.13.34 dirac-admin-get-job-pilots

Retrieve info about pilots that have matched a given Job

Usage:

```
dirac-admin-get-job-pilots [option|cfgfile] ... JobID
```

Arguments:

```
JobID:    DIRAC ID of the Job
```

Example:

```
$ dirac-admin-get-job-pilots 1848
{'https://marlb.in2p3.fr:9000/bqYViq6KrVgGfr6wwgT45Q': {'AccountingSent': 'False',
                                                        'BenchMark': 8.
↪1799999999999997,
                                                        'Broker': 'marwms.in2p3.fr',
                                                        'DestinationSite': 'lpsc-ce.
↪in2p3.fr',
                                                        'GridSite': 'LCG.LPSC.fr',
                                                        'GridType': 'gLite',
                                                        'Jobs': [1848L],
                                                        'LastUpdateTime': datetime.
↪datetime(2011, 2, 21, 12, 39, 10),
                                                        'OutputReady': 'True',
                                                        'OwnerDN': '/O=GRID-FR/C=FR/
↪O=CNRS/OU=LPC/CN=Sebastien Guizard',
                                                        'OwnerGroup': '/biomed',
```

(continues on next page)

(continued from previous page)

```
                                                        'ParentID': 0L,
                                                        'PilotID': 2247L,
                                                        'PilotJobReference': 'https://
↪marlb.in2p3.fr:9000/bqYViq6KrVgGfr6wwgT45Q',
                                                        'PilotStamp': '',
                                                        'Status': 'Done',
                                                        'SubmissionTime': datetime.
↪datetime(2011, 2, 21, 12, 27, 52),
                                                        'TaskQueueID': 399L}}
```

2.13.35 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:9000/26KCLKBFtxXKHF4_ZrQjkw
{'https://marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_ZrQjkw': {'AccountingSent': 'False',
                                                        'BenchMark': 0.0,
                                                        'Broker': 'marwms.in2p3.fr',
                                                        'DestinationSite':
↪'cclcgceli01.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': 2241L,
                                                        'PilotJobReference': 'https://
↪marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_ZrQjkw',
                                                        'PilotStamp': '',
                                                        'Status': 'Done',
                                                        'SubmissionTime': datetime.
↪datetime(2011, 2, 21, 12, 27, 52),
                                                        'TaskQueueID': 399L}}
```

2.13.36 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:

```
$ dirac-admin-get-pilot-logging-info https://marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_
↪ZrQjkw
Pilot Reference: %s https://marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_ZrQjkw
====================== glite-job-logging-info Success ====================

LOGGING INFORMATION:

Printing info for the Job : https://marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_ZrQjkw

    ---
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
- Seqcode                    =
↪UI=000000:NS=0000000001:WM=000000:BH=0000000000:JSS=000000:LM=000000:LRMS=000000:APP=000000:LBS=000
- Source                     =    NetworkServer
```

2.13.37 dirac-admin-get-pilot-output

Usage:

```
dirac-admin-get-pilot-output.py (<options>|<cfgFile>)*
```

Example:

```
$ dirac-admin-get-pilot-output https://marlb.in2p3.fr:9000/26KCLKBFtxXKHF4_ZrQjkw
$ ls -la
drwxr-xr-x  2 hamar marseill     2048 Feb 21 14:13 pilot_26KCLKBFtxXKHF4_ZrQjkw
```

2.13.38 dirac-admin-kill-pilot

Kill the specified pilot

Usage:

```
dirac-admin-kill-pilot <pilot reference>
```

2.13.39 dirac-admin-pilot-summary

Usage:

```
dirac-admin-pilot-summary.py (<options>|<cfgFile>)*
```

Example:

```
$ dirac-admin-pilot-summary
CE                          Status      Count       Status      Count       Status
↪  Count       Status      Count       Status      Count         Status      Count
↪    Status      Count
sbgce1.in2p3.fr             Done        31
lpsc-ce.in2p3.fr            Done        111
lyogrid02.in2p3.fr          Done        81
egee-ce.datagrid.jussieu.fr Aborted     81          Done        18
cclcgceli03.in2p3.fr        Done        275
marce01.in2p3.fr            Done        156
node07.datagrid.cea.fr      Done        75
cclcgceli01.in2p3.fr        Aborted     1           Done        235
ce0.m3pec.u-bordeaux1.fr    Done        63
grive11.ibcp.fr             Aborted     3           Done        90
lptace01.msfg.fr            Aborted     3           Aborted_Day 3           Done
↪  90
ipnls2001.in2p3.fr          Done        87
Total                       Aborted     89          Done        1423        Ready
↪  0           Running     0           Scheduled   0             Submitted   0
↪    Waiting     0
lapp-ce01.in2p3.fr          Aborted     1           Done        111
```

2.13.40 dirac-admin-reoptimize-jobs

Usage:

```
dirac-admin-reoptimize-jobs.py (<options>|<cfgFile>)*
```

Example:

```
$ dirac-admin-reoptimize-jobs
```

2.13.41 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:

```
$ dirac-admin-reset-job 1848
Reset Job 1848
```

2.13.42 dirac-admin-show-task-queues

Usage:

```
dirac-admin-show-task-queues.py (<options>|<cfgFile>)*
```

Example:

```
$ dirac-admin-show-task-queues
Getting TQs..
* TQ 401
        CPUTime: 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
```

2.13.43 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
```

2.13.44 dirac-jobexec

Usage:

```
dirac-jobexec.py (<options>|<cfgFile>)*
```

Options:

```
-p:  --parameter=      : Parameters that are passed directly to the workflow
```

Transformation management commands:

2.13.45 dirac-transformation-archive

Usage:

```
dirac-transformation-archive.py (<options>|<cfgFile>)*
```

2.13.46 dirac-transformation-clean

Usage:

```
dirac-transformation-clean.py (<options>|<cfgFile>)*
```

2.13.47 dirac-transformation-cli

Launch the Transformation shell

Usage:

```
dirac-transformation-cli [option]
```

2.13.48 dirac-transformation-remove-output

Usage:

```
dirac-transformation-remove-output.py (<options>|<cfgFile>)*
```

2.13.49 dirac-transformation-resolve-problematics

Resolve problematic files for the specified transformations

Usage:

```
dirac-transformation-resolve-problematics [options] TransID [TransID]
```

2.13.50 dirac-transformation-verify-outputdata

Usage:

```
dirac-transformation-verify-outputdata.py (<options>|<cfgFile>)*
```

2.13.51 dirac-transformation-replication

Create one replication transformation for each MetaValue given

Is running in dry-run mode, unless enabled with -x

MetaValue and TargetSEs can be comma separated lists:

```
dirac-transformation-replication <MetaValue1[,val2,val3]> <TargetSEs> [-G<Files>] [-S
↪<SourceSEs>][-N<ExtraName>] [-T<Type>] [-M<Key>] [-K...] -x
```

Options:

```
-G  --GroupSize <value>       : Number of Files per transformation task
-S  --SourceSEs <value>       : SourceSE(s) to use, comma separated list
-N  --Extraname <value>       : String to append to transformation name
-P  --Plugin <value>          : Plugin to use for transformation
-T  --Flavour <value>         : Flavour to create: Replication or Moving
-K  --MetaKey <value>         : Meta Key to use: TransformationID
-M  --MetaData <value>        : MetaData to use Key/Value Pairs: 'DataType:REC,'
-x  --Enable                  : Enable the transformation creation, otherwise dry-run
```

Managing DIRAC installation:

2.13.52 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': 54.859999999999999,
                         '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',
             ())}
```

2.13.53 dirac-install-agent

2013-02-06 12:30:28 UTC Framework NOTICE: DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7

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
```

2.13.54 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)
```

2.13.55 dirac-install-service

2013-02-06 13:06:05 UTC Framework NOTICE: DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7

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
```

2.13.56 dirac-install-web-portal

2013-02-06 13:06:07 UTC Framework NOTICE: DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7

Do the initial installation of a DIRAC Web portal

Usage:

```
dirac-install-web-portal [option|cfgfile] ...
```

2.13.57 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:  externals=        : Externals 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

2.13.58 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)
```

2.13.59 dirac-restart-mysql

Restart DIRAC MySQL server

Usage:

```
dirac-restart-mysql [option|cfgfile] ...
```

2.13.60 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)
```

2.13.61 dirac-start-mysql

Start DIRAC MySQL server

Usage:

```
dirac-start-mysql [option|cfgfile] ...
```

2.13.62 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
                                        Name : Runit    Uptime    PID
          WorkloadManagement_PilotStatusAgent : Run        4029     1697
           WorkloadManagement_JobHistoryAgent : Run        4029     1679
                     Framework_CAUpdateAgent : Run        4029     1658
                  Framework_SecurityLogging : Run        4025     2111
                 WorkloadManagement_Matcher : Run        4029     1692
           WorkloadManagement_StalledJobAgent : Run        4029     1704
          WorkloadManagement_JobCleaningAgent : Run        4029     1676
                               Web_paster : Run        4029     1683
           WorkloadManagement_MightyOptimizer : Run        4029     1695
             WorkloadManagement_JobMonitoring : Run        4025     2133
     WorkloadManagement_StatesAccountingAgent : Run        4029     1691
             RequestManagement_RequestManager : Run        4025     2141
                   DataManagement_FileCatalog : Run        4024     2236
              WorkloadManagement_JobManager : Run        4024     2245
         WorkloadManagement_TaskQueueDirector : Run        4029     1693
                      Framework_Notification : Run        4026     2101
                                  Web_httpd : Run        4029     1681
                      Framework_ProxyManager : Run        4024     2260
                        Framework_Monitoring : Run        4027     1948
          WorkloadManagement_WMSAdministrator : Run        4027     1926
            WorkloadManagement_InputDataAgent : Run        4029     1687
                     Framework_SystemLogging : Run        4025     2129
                        Accounting_DataStore : Run        4025     2162
                Framework_SystemAdministrator : Run        4026     2053
                   Accounting_ReportGenerator : Run        4026     2048
             Framework_SystemLoggingDBCleaner : Run        4029     1667
           DataManagement_StorageElementProxy : Run        4024     2217
                          Framework_Plotting : Run        4025     2208
                      Configuration_Server : Run        4029     1653
              WorkloadManagement_SandboxStore : Run        4025     2186
```

(continues on next page)

(continued from previous page)

```
                Framework_UserProfileManager : Run        4025     2182
             DataManagement_StorageElement : Run        4024     2227
       Framework_TopErrorMessagesReporter : Run        4029     1672
            WorkloadManagement_MPIService : Run        4024     2226
                 Configuration_CE2CSAgent : Run           1    32461
        WorkloadManagement_JobStateUpdate : Run        4025     2117
            Framework_SystemLoggingReport : Run        4024     2220
                 Framework_BundleDelivery : Run        4025     2157
```

2.13.63 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)
```

2.13.64 dirac-stop-mysql

Stop DIRAC MySQL server

Usage:

```
dirac-stop-mysql [option|cfgfile] ...
```

2.13.65 dirac-monitoring-get-components-status

Usage:

```
dirac-monitoring-get-components-status.py (<options>|<cfgFile>)*
```

2.13.66 dirac-service

2013-02-06 13:06:27 UTC Framework FATAL: You must specify which server to run!

2.13.67 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)
```

2.13.68 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 <cename>

-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>

-L:  --LocalSE=         : Configure LocalSite/LocalSE=<localse>

-F   --ForceUpdate      : Force Update of dirac.cfg (otherwise nothing happens if␣
↪dirac.cfg already exists)
```

2.13.69 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-get-CAs
```

2.13.70 dirac-info

Report info about local DIRAC installation

Usage:

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

Example:

```
$ dirac-info
       DIRAC version : v5r12
               Setup : Dirac-Production
 ConfigurationServer : ['dips://dirac.in2p3.fr:9135/Configuration/Server']
 VirtualOrganization : vo.formation.idgrilles.fr
```

2.13.71 dirac-version

Return the current dirac version used by the client.

Example:

```
$ dirac-version
v5r12-pre9
```

Managing DIRAC software:

2.13.72 dirac-deploy-scripts

Scripts will be deployed at /afs/in2p3.fr/home/h/hamar/DIRAC-v6r7/scripts

Inspecting DIRAC module

Example:

```
$ dirac-deploy-scripts
Scripts will be deployed at /afs/in2p3.fr/home/h/hamar/DIRAC-v5r12/scripts
Inspecting DIRAC module
Inspecting EELADIRAC module
```

2.13.73 dirac-distribution

Create tarballs for a given DIRAC release

Usage:

```
dirac-distribution [option|cfgfile] ...
```

Options:

```
-r:  --releases=       : releases to build (mandatory, comma separated)

-l:  --project=        : Project to build the release for (DIRAC by default)

-D:  --destination     : Destination where to build the tar files
```

(continues on next page)

(continued from previous page)

```
-i:  --pythonVersion    : Python version to use (25/26)

-P   --ignorePackages   : Do not make tars of python packages

-C:  --relcfg=          : Use <file> as the releases.cfg

-b   --buildExternals   : Force externals compilation even if already compiled

-B   --ignoreExternals  : Skip externals compilation

-t:  --buildType=       : External type to build (client/server)

-x:  --externalsLocation= : Use externals location instead of downloading them

-j:  --makeJobs=        : Make jobs (default is 1)

-M:  --defaultsURL=     : Where to retrieve the global defaults from
```

2.13.74 dirac-externals-requirements

Usage:

```
dirac-externals-requirements.py (<options>|<cfgFile>)*
```

Options:

```
-t:  --type=            : Installation type. 'server' by default.
```

2.13.75 dirac-fix-ld-library-path

Usage:

```
dirac-fix-ld-library-path.py (<options>|<cfgFile>)*
```

2.13.76 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
```

2.13.77 dirac-install-mysql

Install MySQL. The clever way to do this is to use the dirac-admin-sysadmin-cli.

User convenience:

2.13.78 dirac-accounting-report-cli

Command line interface to DIRAC Accounting ReportGenerator Service

Usage:

```
dirac-accounting-report-cli [option|cfgfile] ...
```

2.13.79 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
```

2.13.80 dirac-cert-convert.sh

> From a p12 file, obtain the pem files with the right access rights. Needed to obain 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.
```

2.13.81 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
```

2.13.82 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
```

2.13.83 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:

2.13.84 dirac-admin-accounting-cli

Command line administrative interface to DIRAC Accounting DataStore Service

Usage:

```
dirac-admin-accounting-cli [option|cfgfile] ...
```

2.13.85 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:

```
$ dirac-admin-get-proxy vhamar dirac_user
Proxy downloaded to /afs/in2p3.fr/home/h/hamar/proxy.vhamar.dirac_user
```

2.13.86 dirac-admin-proxy-upload

Usage:

```
dirac-admin-proxy-upload.py (<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
```

2.13.87 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
```

2.13.88 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
-------------------------------------------------------------------------------------------
↪------------------
| UserDN                                          | UserGroup   | ExpirationTime      ␣
↪| PersistentFlag |
-------------------------------------------------------------------------------------------
↪------------------
| /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar | dirac_user  | 2011-06-29 12:04:25␣
↪| True           |
-------------------------------------------------------------------------------------------
↪------------------
```

2.13.89 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
```

(continues on next page)

(continued from previous page)

```
-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']
```

2.13.90 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

-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
```

(continues on next page)

(continued from previous page)

```
-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

-r   --rfc              : Create an RFC proxy (https://www.ietf.org/rfc/rfc3820.txt)
```

Example:

```
$ dirac-proxy-init -g dirac_user -t --rfc
Enter Certificate password:
$
```

2.13.91 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}}
```

2.13.92 dirac-admin-select-requests

Select requests from the request management system

Usage:

```
dirac-admin-select-requests [option|cfgfile] ...
```

Options:

```
-    --JobID=           : WMS JobID for the request (if applicable)

-    --RequestID=       : ID assigned during submission of the request

-    --RequestName=     : XML request file name

-    --RequestType=     : Type of the request e.g. 'transfer'

-    --Status=          : Request status

-    --Operation=       : Request operation e.g. 'replicateAndRegister'
```

(continues on next page)

(continued from previous page)

```
-    --RequestStart=    : First request to consider (start from 0 by default)

-    --Limit=           : Selection limit (default 100)

-    --OwnerDN=         : DN of owner (in double quotes)

-    --OwnerGroup=      : Owner group
```

Example:

```
$ dirac-admin-select-requests
9 request(s) selected with conditions  and limit 100
['RequestID', 'RequestName', 'JobID', 'OwnerDN', 'OwnerGroup', 'RequestType', 'Status
↪', 'Operation', 'Error', 'CreationTime', 'LastUpdateTime']
['1', 'LFNInputData_44.xml', '44', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
↪'dirac_user', 'diset', 'Waiting', 'setJobStatusBulk', 'None',   '2010-12-08 22:27:07
↪', '2010-12-08 22:27:08']
['1', 'LFNInputData_44.xml', '44', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
↪'dirac_user', 'diset', 'Waiting', 'setJobParameters', 'None', '2010-12-08 22:27:07',
↪ '2010-12-08 22:27:08']
['2', 'API_2_23.xml', '23', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar', 'dirac_
↪user', 'diset', 'Waiting', 'setJobParameters', 'None', '2010-12-08 22:27:07', '2010-
↪12-08 22:27:09']
['3', 'API_19_42.xml', '42', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
↪'dirac_user', 'diset', 'Waiting', 'setJobStatusBulk', 'None', '2010-12-08 22:27:07',
↪ '2010-12-08 22:27:09']
['3', 'API_19_42.xml', '42', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar',
↪'dirac_user', 'diset', 'Waiting', 'setJobParameters', 'None', '2010-12-08 22:27:07',
↪ '2010-12-08 22:27:09']
['4', 'Accounting.DataStore.1293829522.01.0.145174243188', 'None', 'Unknown', 'Unknown
↪', 'diset', 'Waiting', 'commitRegisters', 'None', '2010-12-31 21:05:22', '2010-12-
↪31 21:56:49']
['5', 'Accounting.DataStore.1293840021.45.0.74714473302', 'None', 'Unknown', 'Unknown
↪', 'diset', 'Waiting', 'commitRegisters', 'None', '2011-01-01 00:00:21', '2011-01-
↪01 00:05:39']
['6', '1057.xml', '1057', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar', 'dirac_
↪user', 'register', 'Waiting', 'registerFile', 'None', '2011-01-31 13:31:46', '2011-
↪01-31 13:31:53']
['7', '1060.xml', '1060', '/O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Vanessa Hamar', 'dirac_
↪user', 'register', 'Waiting', 'registerFile', 'None', '2011-01-31 13:42:33', '2011-
↪01-31 13:42:36']
```

2.13.93 dirac-admin-sysadmin-cli

Usage:

```
dirac-admin-sysadmin-cli.py (<options>|<cfgFile>)*
```

Options:

```
-H:  --host=           :    Target host
```

Example:

```
$ dirac-admin-sysadmin-cli --host dirac.in2p3.fr
DIRAC Root Path = /afs/in2p3.fr/home/h/hamar/DIRAC-v5r12
dirac.in2p3.fr >
```

2.13.94 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
```

2.13.95 dirac-configuration-cli

Command line interface to DIRAC Configuration Server

Usage:

```
dirac-configuration-cli [option|cfgfile] ...
```

2.13.96 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
```

2.13.97 dirac-configuration-shell

Usage:

```
dirac-configuration-shell.py (<options>|<cfgFile>)*
```

2.13.98 dirac-repo-monitor

Monitor the jobs present in the repository

Usage:

```
dirac-repo-monitor [option|cfgfile] ... RepoDir
```

Arguments:

```
RepoDir:  Location of Job Repository
```

2.13.99 dirac-rss-reassign-token

Re-assign a token: if it was assigned to a human, assign it to 'RS_SVC' and viceversa.

Usage:

```
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
```

2.13.100 dirac-rss-renew-token

Extend the duration of given token

Usage:

```
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)
```

2.13.101 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:

```
--element=            Element family to be Synchronized ( Site, Resource or Node )

--elementType=        ElementType narrows the search; None if default

--elementName=        ElementName; None if default

--tokenOwner=         Owner of the token; None if default

--statusType=         StatusType; None if default

--status=             Status; None if default

Verbosity:

-o LogLevel=LEVEL     NOTICE by default, levels available: INFO, DEBUG, VERBOSE..
```

2.13.102 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
```

2.13.103 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`
```

2.13.104 dirac-rss-setup

What is this doing??

2.13.105 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
```

2.13.106 dirac-stager-monitor-request

Report the summary of the stage task from the DB.

Usage:

```
dirac-stager-monitor-request [option|cfgfile] ... Request ...
```

Arguments:

```
Request:  ID of the Stage request in the StorageManager
```

2.13.107 dirac-stager-stage-files

Submit Stage Request for Files at given SE

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)
```

2.13.108 install_site.sh

Usage:

```
install_site.sh [Options] ... CFG_file"
```

Options::

-v, --version for a specific version”

-d, --debug debug mode”

-h, --help print this”

CFG_file - is the name of the installation configuration file which contains” all the instructions for the DIRAC installation. See DIRAC Administrator ” Guide for the details”

2.13.109 dirac-agent

Script running a dirac agent. Mostly internal.

2.13.110 dirac-executor

2013-02-06 12:30:09 UTC Framework FATAL: You must specify which executor to run!

2.13.111 dirac-compile-externals

Compile DIRAC externals (does not require DIRAC code)

Usage:

```
dirac-compile-externals [options]...
```

Options:

```
-D:  --destination=    : Destination where to build the externals

-t:  --type=           : Type of compilation (default: client)

-e:  --externalsPath=  : Path to the externals sources

-v:  --version=        : Version of the externals to compile (default will be the␣
↪latest commit)

-i:  --pythonVersion=  : Python version to compile (default 26)

-f   --fixLinksOnly    : Only fix absolute soft links

-j:  --makeJobs=       : Number of make jobs, by default is 1
```

2.13.112 dirac-fix-mysql-script

Fixes the mysql.server script, it requires a proper /LocalInstallation section

Usage:

```
dirac-fix-mysql-script [option] ... [cfgfile]
```

2.14 Limitations

2.14.1 DataManagement

LFN length

Because they are stored in a database, the LFNs are limited in size. The standard size is 255 characters. It is enforced in the following database:

- JobDB
- TransformationTB
- StorageManagementDB
- DataIntegrityDB
- FTSDB
- RequestDB

Shall you want to have longer LFN, then you would need to update your database manually.

A special case is the DFC. The limitations depend on the Directory and File managers you use.

In the DirectoryLevelTree and FileManager (defaults one) managers, the LFNs are split by '/', yielding other limitations:

- 128 char for the filename
- 255 char for each directory level

In case of the Managers WithPkAndPs (LHCb):

- 128 char for the filename
- 255 for the base path

2.15 Scaling

2.15.1 Servers

When you servers are heavily loaded, you may want to tune some kernel parameters. Internet is full or resources to explain you what you should do, but a few parameters of interests certainly are the number of file descriptors allowed, as well as a few kernel tcp parameters that should be increased (https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt):

```
net.nf_conntrack_max
net.ipv4.tcp_max_syn_backlog
net.core.somaxconn
net.core.netdev_max_backlog
```

Finally, the parameter *SocketBacklog* for a service can be increased (*man listen* is your friend).

2.15.2 Duplications

In some cases, it is possible to run several instances of the same agent/service in order to scale.

2.15.3 Services

System	Component	Duplicate	Remarque
Accounting	DataStore	PARTIAL	One master and helpers (http://dirac.readthedocs.io/en/latest/Admir
	ReportGenerator		
Configuration	Configuration		
	Server	PARTIAL	
DataManagement	HttpStorageAccess		
	DataIntegrity	YES	
	FileCatalog	YES	
	FileCatalogProxy		
	FTSManager	YES	
	IRODSStorageElement		
	StorageElement		
	StorageElementProxy		
Framework	BundleDelivery		
	ComponentMonitoring		
	Monitoring		
	Notification		
	Plotting		
	ProxyManager	YES	
	RabbitMQSync		

Table 1 – continued from previous page

	SecurityLogging	YES	
	SystemAdministrator	YES	In principle there should be one on each and every machine
	SystemLogging		
	SystemLoggingReport		
	UserProfileManager		
Monitoring	Monitoring		
RequestManagement	ReqManager	YES	
	ReqProxy	YES	
ResourcesStatus	Publisher		
	ResourceManagement		
	ResourceStatus		
StorageManager	StorageManager		
Transformation	TransformationManager		
WorkloadManagement	JobManager		
	JobMonitoring		
	JobStateSync		
	JobStateUpdate		
	Matcher		
	OptimizationMind		
	SandboxStore		
	WMSAdministrator		

2.15.4 Agents

System	Component	Duplicate	Remarque
Accounting	NetworkAgent		
	Test_NetworkAgent		
Configuration	Bdii2CSAgent		
	GOCDB2CSAgent		
	VOMS2CSAgent		
DataManagement	CleanFTSDBAgent	NO	
	FTSAgent	PARTIAL	See bellow
Framework	CAUpdateAgent		
	MyProxyRenewalAgent		
RequestManagement	CleanReqDBAgent	NO	
	RequestExecutingAgent	YES	
ResourceStatus	CacheFeederAgent		
	ElementInspectorAgent		
	EmailAgent		
	SiteInspectorAgent		
	SummarizeLogsAgent		
	Test_EmailActionAgent		
	TokenAgent		
StorageManagement	RequestFinalizationAgent	NO	
	RequestPreparationAgent	NO	
	StageMonitorAgent	NO	
	StageRequestAgent	NO	
Transformation	InputDataAgent		
	MCExtensionAgent		
	RequestTaskAgent		
	TransformationAgent		

Continued on next page

Table 2 – continued from previous page

	TransformationCleaningAgent		
	ValidateOutputDataAgent		
	WorkflowTaskAgent		
WorkloadManagement	DiracSiteAgent		
	JobAgent		
	JobCleaningAgent		
	PilotMonitorAgent		
	PilotStatusAgent		
	StalledJobAgent		
	StatesAccountingAgent		
	StatesMonitoringAgent		

FTSAgent

This agent can be split in two: one agent for the failover transfers, and one for the others (coming from transformations and so on). For this you need to define two agents using both the FTSAgent module, and use the *ProcessJobRequests* flag: once to True, once to False.

2.16 DIRAC Administrator tutorials

Each of this tutorial is a step by step guide.

2.16.1 Basic Tutorial setup

Tutorial goal

The aim of the tutorial is to have a self contained DIRAC setup. You will be guided through the whole installation process both of the server part and the client part. By the end of the tutorial, you will have:

- a Configuration service, to serve other servers and clients
- a ComponentMonitoring service to keep track of other services and agents installed
- a SystemAdministrator service to manage the DIRAC installation in the future
- the WebApp, to allow for web interface access

The setup you will have at the end is the base for all the other tutorials.

Basic requirements

We assume that you have at your disposition a fresh SLC6 64bit installation. If you don't, we recommend installing a virtual machine. Instructions for installing SLC6 can be found here

In this tutorial, we will use a freshly installed SLC6 x86_64 virtual machine, with all the default options, except the hostname being `dirac-tuto`.

Machine setup

This section is to be executed as `root` user.

Make sure that the machine can address itself using the `dirac-tuto` alias. Modify the `/etc/host` file as such:

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4 dirac-
↪tuto
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6 dirac-
↪tuto
```

Install runit

The next step is to install `runit`, which is responsible for supervising DIRAC processes

First, install the RPM:

```
yum install -y http://diracproject.web.cern.ch/diracproject/rpm/runit-2.1.2-1.el6.x86_
↪64.rpm
```

Next, edit the `/etc/init/runsvdir.conf` file to point to the future DIRAC installation as such:

```
# for runit - manage /usr/sbin/runsvdir-start
start on runlevel [2345]
stop on runlevel [^2345]
normal exit 0 111
respawn
exec /opt/dirac/sbin/runsvdir-start
```

Finally, create the directory `/opt/dirac/sbin`:

```
mkdir -p /opt/dirac/sbin
```

and the file `/opt/dirac/sbin/runsvdir-start` with the following content:

```
cd /opt/dirac
RUNSVCTRL='/sbin/runsvctrl'
chpst -u dirac $RUNSVCTRL d /opt/dirac/startup/*
killall runsv svlogd
RUNSVDIR='/sbin/runsvdir'
exec chpst -u dirac $RUNSVDIR -P /opt/dirac/startup 'log:  DIRAC runsv'
```

make it executable:

```
chmod +x /opt/dirac/sbin/runsvdir-start
```

and restart `runsvdir`:

```
restart runsvdir
```

Install MySQL

First of all, remove the existing (outdated) installation:

```
yum remove -y $(rpm -qa | grep -i mysql | paste -sd ' ')
```

Install all the necessary RPMs for MySQL 5.7:

```
yum install -y https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-community-devel-5.
↪7.25-1.el6.x86_64.rpm https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-community-
↪server-5.7.25-1.el6.x86_64.rpm https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-
↪community-client-5.7.25-1.el6.x86_64.rpm  https://dev.mysql.com/get/Downloads/MySQL-
↪5.7/mysql-community-libs-5.7.25-1.el6.x86_64.rpm https://dev.mysql.com/get/
↪Downloads/MySQL-5.7/mysql-community-common-5.7.25-1.el6.x86_64.rpm
```

(continues on next page)

(continued from previous page)

Setup the root password:

```
[root@dirac-tuto ~]# mysqld_safe --skip-grant-tables &
[1] 8840
[root@dirac-tuto ~]# 190410 16:11:21 mysqld_safe Logging to '/var/lib/mysql/dirac-
↪tuto.err'.
190410 16:11:21 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql

[root@dirac-tuto ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.43 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)


mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('password');
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye

[root@dirac-tuto ~]# service mysqld stop
Shutting down MySQL..190410 16:12:52 mysqld_safe mysqld from pid file /var/lib/mysql/
↪dirac-tuto.pid ended
                                                           [  OK  ]
[1]+  Done                    mysqld_safe --skip-grant-tables
[root@dirac-tuto ~]# service mysqld start
Starting MySQL.
```

Create the `dirac` user

The user that will run the server will be `dirac`. You can set a password for that user:

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

All files bellow `/opt/dirac/` should belong to this user:

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

Server installation

This section is to be executed as `dirac` user

CA and certificate

DIRAC relies on TLS for securing its connections and for authorization and authentication. Since we are using a self contained installation, we will be using our own CA. There are a bunch of utilities that we will be using to generate the necessary files.

First of all, download the utilities from the DIRAC repository:

```
mkdir ~/caUtilities/ && cd ~/caUtilities/
curl -O -L https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/tests/
↪Jenkins/utilities.sh
curl -O -L https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/tests/
↪Jenkins/config/ci/openssl_config_ca.cnf
curl -O -L https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/tests/
↪Jenkins/config/ci/openssl_config_host.cnf
curl -O -L https://raw.githubusercontent.com/DIRACGrid/DIRAC/integration/tests/
↪Jenkins/config/ci/openssl_config_user.cnf
```

We then will generate the CA, the host certificate, and the client certificate that will be used by our client later. First, we create a subshell, and source the tools to be able to call the functions:

```
bash
export SERVERINSTALLDIR=/opt/dirac
export CI_CONFIG=~/caUtilities/
source utilities.sh
```

Then we generate the CA:

```
[dirac@dirac-tuto caUtilities]$ generateCA
==> [generateCA]
Generating RSA private key, 2048 bit long modulus
.............+++
...............+++
e is 65537 (0x10001)
```

Now generate a host certificate, valid for 1 year:

```
[dirac@dirac-tuto ca]$ generateCertificates 365
==> [generateCertificates]
Using configuration from /opt/dirac/etc/grid-security/ca/openssl_config_ca.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Apr 10 14:47:38 2019 GMT
            Not After : Apr  9 14:47:38 2020 GMT
        Subject:
            countryName               = ch
            organizationName          = DIRAC
            organizationalUnitName    = DIRAC CI
            commonName                = dirac-tuto
```

(continues on next page)

(continued from previous page)

```
            emailAddress              = lhcb-dirac-ci@cern.ch
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Server Certificate
            X509v3 Subject Key Identifier:
                85:90:F4:7D:6E:31:50:F7:3E:53:7E:0B:B3:22:D5:5C:37:D4:D0:5A
            X509v3 Authority Key Identifier:
                keyid:33:F0:C8:60:6D:6B:52:BD:E9:A7:FA:57:27:72:5A:5D:7E:43:12:ED
                DirName:/O=DIRAC CI/CN=DIRAC CI Signing Certification Authority
                serial:88:B1:7A:54:17:8C:00:13

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                DNS:dirac-tuto, DNS:localhost
Certificate is to be certified until Apr  9 14:47:38 2020 GMT (365 days)

Write out database with 1 new entries
Data Base Updated
```

Finally, generate the client certificate for later, also valid one year:

```
[dirac@dirac-tuto grid-security]$ generateUserCredentials 365
==> [generateUserCredentials]
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.....................................................................
↪.....................................................+++
e is 65537 (0x10001)
Using configuration from /opt/dirac/etc/grid-security/ca/openssl_config_ca.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Apr 10 14:48:31 2019 GMT
            Not After : Apr  9 14:48:31 2020 GMT
        Subject:
            countryName               = ch
            organizationName          = DIRAC
            organizationalUnitName    = DIRAC CI
            commonName                = ciuser
            emailAddress              = lhcb-dirac-ci@cern.ch
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                98:BB:F0:A8:96:4F:80:C8:3E:21:60:5E:FD:17:4E:34:97:EF:31:17
            X509v3 Authority Key Identifier:
                keyid:33:F0:C8:60:6D:6B:52:BD:E9:A7:FA:57:27:72:5A:5D:7E:43:12:ED

            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment
```

(continues on next page)

(continued from previous page)

```
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            Netscape Comment:
                OpenSSL Generated Client Certificate
Certificate is to be certified until Apr  9 14:48:31 2020 GMT (365 days)

Write out database with 1 new entries
Data Base Updated
```

To finish, time to exit the subshell:

```
exit
```

At this point, you should find:

- The CA in `/opt/dirac/etc/grid-security/certificates`:

  ```
  [dirac@dirac-tuto caUtilities]$ ls /opt/dirac/etc/grid-security/certificates/
  ```

  855f710d.0 ca.cert.pem

- The host certificate (`hostcert.pem`) and key (`hostkey.pem`) in `/opt/dirac/etc/grid-security`:

  ```
  [dirac@dirac-tuto caUtilities]$ ls /opt/dirac/etc/grid-security/
  ```

  ca certificates hostcert.pem hostkey.pem openssl_config_host.cnf request.csr.pem

- The user credentials for later in `/opt/dirac/user/`:

  ```
  [dirac@dirac-tuto caUtilities]$ ls /opt/dirac/user/
  ```

  client.key client.pem client.req openssl_config_user.cnf

Install DIRAC Server

This section is to be run as dirac user.

We will install DIRAC v6r21 with DIRACOS.

First, download the installer, and make it executable:

```
mkdir ~/DiracInstallation && cd ~/DiracInstallation
curl -O -L https://github.com/DIRACGrid/DIRAC/raw/integration/Core/scripts/install_
↪site.sh
chmod +x install_site.sh
```

`install_site.sh` requires a configuration file to tell it what and how to install. Create a file called `installation.cfg` with the following content:

```
LocalInstallation
{
  #  DIRAC release version to install
  Release = v6r21p3
  #  Installation type
  InstallType = server
```

(continues on next page)

(continued from previous page)

```
  #  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
  #  Install the WebApp extension
  Extensions = WebApp

  # Name of the VO we will use
  VirtualOrganization = tutoVO
  # Name of the site or host
  SiteName = dirac-tuto
  # Setup name
  Setup = MyDIRAC-Production
  #  Default name of system instances
  InstanceName = Production
  #  Flag to skip download of CAs
  SkipCADownload = yes
  #  Flag to use the server certificates
  UseServerCertificate = yes

  # Name of the Admin user (from the user certificate we created )
  AdminUserName = ciuser
  # DN of the Admin user certificate (from the user certificate we created)
  AdminUserDN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
  AdminUserEmail= adminUser@cern.ch
  # Name of the Admin group
  AdminGroupName = dirac_admin

  # DN of the host certificate (from the host certificate we created)
  HostDN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=dirac-tuto/emailAddress=lhcb-dirac-ci@cern.ch
  # Define the Configuration Server as Master
  ConfigurationMaster = yes

  # List of DataBases to be installed (what's here is a list for a basic installation)
  Databases = InstalledComponentsDB
  Databases += ResourceStatusDB

  #  List of Services to be installed (what's here is a list for a basic installation)
  Services  = Configuration/Server
  Services += Framework/ComponentMonitoring
  Services += Framework/SystemAdministrator
  #  Flag determining whether the Web Portal will be installed
  WebPortal = yes
  WebApp = yes

  Database
  {
    #  User name used to connect the DB server
    User = Dirac
    #  Password for database user access
    Password = Dirac
    #  Password for root DB user
    RootPwd = password
    #  location of DB server
    Host = localhost
  }
```

(continues on next page)

(continued from previous page)

```
}
```

And then run it:

```
[dirac@dirac-tuto DIRAC]$ ./install_site.sh --dirac-os install.cfg
--2019-04-11 08:51:21--  https://github.com/DIRACGrid/DIRAC/raw/integration/Core/
↪scripts/dirac-install.py
Resolving github.com... 140.82.118.4, 140.82.118.3
Connecting to github.com|140.82.118.4|:443... connected.
HTTP request sent, awaiting response... 302 Found


[...]


Status of installed components:

  Name                            Runit Uptime PID
=================================================
1 Web_WebApp                      Run        4 24338
2 Configuration_Server            Run       53 24142
3 Framework_ComponentMonitoring   Run       36 24207
4 Framework_SystemAdministrator   Run       20 24247
```

You can verify that the components are running:

```
[dirac@dirac-tuto DIRAC]$ runsvstat /opt/dirac/startup/*
/opt/dirac/startup/Configuration_Server: run (pid 24142) 288 seconds
/opt/dirac/startup/Framework_ComponentMonitoring: run (pid 24207) 271 seconds
/opt/dirac/startup/Framework_SystemAdministrator: run (pid 24247) 255 seconds
/opt/dirac/startup/Web_WebApp: run (pid 24338) 239 seconds
```

The logs are to be found in `/opt/dirac/runit/`, grouped by component.

The installation created the file `/opt/dirac/etc/dirac.cfg`. The content is the same as the `installation.cfg`, with the addition of the following:

```
DIRAC
{
  Setup = MyDIRAC-Production
  VirtualOrganization = tutoVO
  Extensions = WebApp
  Security
  {
  }
  Setups
  {
    MyDIRAC-Production
    {
      Configuration = Production
      Framework = Production
    }
  }
  Configuration
  {
    Master = yes
    Name = MyDIRAC-Production
```

(continues on next page)

(continued from previous page)

```
    Servers = dips://dirac-tuto:9135/Configuration/Server
  }
}
LocalSite
{
  Site = dirac-tuto
}
Systems
{
  Databases
  {
    User = Dirac
    Password = Dirac
    Host = localhost
    Port = 3306
  }
  NoSQLDatabases
  {
    Host = dirac-tuto
    Port = 9200
  }
}
```

This part is used as configuration for all your services and agents that you will run. It contains two important information:

- The database credentials
- The address of the configuration server: `Servers = dips://dirac-tuto:9135/Configuration/Server`

The Configuration service will serve the content of the file `/opt/dirac/etc/MyDIRAC-Production.cfg` to every client, be it a service, an agent, a job, or an interactive client. The content looks like such:

```
DIRAC
{
  Extensions = WebApp
  VirtualOrganization = tutoVO
  Configuration
  {
    Name = MyDIRAC-Production
    Version = 2019-04-11 06:52:18.414086
    MasterServer = dips://dirac-tuto:9135/Configuration/Server
  }
  Setups
  {
    MyDIRAC-Production
    {
      Configuration = Production
      Framework = Production
    }
  }
}
Registry
{
  Users
  {
```

(continues on next page)

(continued from previous page)

```
    ciuser
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
      Email = adminUser@cern.ch
    }
  }
  Groups
  {
    dirac_user
    {
      Users = ciuser
      Properties = NormalUser
    }
    dirac_admin
    {
      Users = ciuser
      Properties = AlarmsManagement
      Properties += ServiceAdministrator
      Properties += CSAdministrator
      Properties += JobAdministrator
      Properties += FullDelegation
      Properties += ProxyManagement
      Properties += Operator
    }
  }
  Hosts
  {
    dirac-tuto
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=dirac-tuto/emailAddress=lhcb-dirac-ci@cern.ch
      Properties = TrustedHost
      Properties += CSAdministrator
      Properties += JobAdministrator
      Properties += FullDelegation
      Properties += ProxyManagement
      Properties += Operator
    }
  }
  DefaultGroup = dirac_user
}
Operations
{
  Defaults
  {
    EMail
    {
      Production = adminUser@cern.ch
      Logging = adminUser@cern.ch
    }
  }
}
WebApp
{
  Access
  {
    upload = TrustedHost
  }
```

(continues on next page)

(continued from previous page)

```
}
Systems
{
  Framework
  {
    Production
    {
      Services
      {
        ComponentMonitoring
        {
          Port = 9190
          Authorization
          {
            Default = ServiceAdministrator
            componentExists = authenticated
            getComponents = authenticated
            hostExists = authenticated
            getHosts = authenticated
            installationExists = authenticated
            getInstallations = authenticated
            updateLog = Operator
          }
        }
        SystemAdministrator
        {
          Port = 9162
          Authorization
          {
            Default = ServiceAdministrator
            storeHostInfo = Operator
          }
        }
      }
      URLs
      {
        ComponentMonitoring = dips://dirac-tuto:9190/Framework/ComponentMonitoring
        SystemAdministrator = dips://dirac-tuto:9162/Framework/SystemAdministrator
      }
      FailoverURLs
      {
      }
      Databases
      {
        InstalledComponentsDB
        {
          DBName = InstalledComponentsDB
          Host = localhost
          Port = 3306
        }
      }
    }
  }
}
```

This configuration will be used for example by Services in order to:

- know their configuration (for example the `ComponentMonitoring` Service will use everything under

`Systems/Framework/Production/Services/ComponentMonitoring`)

- Identify host and persons (`Registry` section)

Or by clients to get the URLs of given services (for example `ComponentMonitoring = dips://dirac-tuto:9190/Framework/ComponentMonitoring`)

Since this configuration is given as a whole to every client, you understand why no database credentials are in this file. Services and Agents running on the machine will have their configuration as a merge of what is served by the Configuration service and the `/opt/dirac/etc/dirac.cfg`, and thus have access to these private information.

The file `/opt/dirac/bashrc` is to be sourced whenever you want to use the server installation.

Client installation

Now we will create another linux account `diracuser` and another installation to be used as client

Setup client session

This section has to be ran as `root`

First, create an account, and add in its `~/.globus/` directory the user certificate you created earlier:

```
adduser -s /bin/bash -d /home/diracuser diracuser
passwd diracuser
mkdir ~diracuser/.globus/
cp /opt/dirac/user/client.pem ~diracuser/.globus/usercert.pem
cp /opt/dirac/user/client.key ~diracuser/.globus/userkey.pem
chown -R diracuser:diracuser ~diracuser/.globus/
```

Install DIRAC client

This section has to be ran as `diracuser`

We will do the installation in the `~/DiracInstallation` directory. For a client, the configuration is really minimal, so we will just install the code and its dependencies. First, create the structure, and download the installer:

```
mkdir ~/DiracInstallation && cd ~/DiracInstallation
curl -O -L https://github.com/DIRACGrid/DIRAC/raw/integration/Core/scripts/dirac-
↪install.py
chmod +x dirac-install.py
```

Now we trigger the installation, with the same version as the server:

```
[diracuser@dirac-tuto DIRAC]$ ./dirac-install.py -r v6r21 --dirac-os
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Processing installation requirements
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Destination path for installation is /
↪home/diracuser/DIRAC
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Discovering modules to install
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Installing modules...
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Installing DIRAC:v6r21
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Retrieving http://diracproject.web.
↪cern.ch/diracproject/tars/DIRAC-v6r21.tar.gz
2019-04-11 14:46:41 UTC dirac-install [NOTICE]  Retrieving http://diracproject.web.
↪cern.ch/diracproject/tars/DIRAC-v6r21.md5
```

(continues on next page)

(continued from previous page)

```
2019-04-11 14:46:42 UTC dirac-install [NOTICE]  Deploying scripts...
Scripts will be deployed at /home/diracuser/DIRAC/scripts
Inspecting DIRAC module
2019-04-11 14:46:42 UTC dirac-install [NOTICE]  Installing DIRAC OS ...
2019-04-11 14:46:42 UTC dirac-install [NOTICE]  Retrieving https://diracos.web.cern.
↪ch/diracos/releases/diracos-1.0.0.tar.gz
................................................................................
↪................................................................................
↪................................................................................
↪................................................................................
↪................................................................................
↪................................................................................
↪................................................................................
↪................................................................................
↪.............................2019-04-11 14:46:46 UTC dirac-install [NOTICE] ␣
↪Retrieving https://diracos.web.cern.ch/diracos/releases/diracos-1.0.0.md5
2019-04-11 14:47:02 UTC dirac-install [NOTICE]  Fixing externals paths...
2019-04-11 14:47:02 UTC dirac-install [NOTICE]  Running externals post install...
2019-04-11 14:47:02 UTC dirac-install [NOTICE]  Creating /home/diracuser/DIRAC/bashrc
2019-04-11 14:47:02 UTC dirac-install [NOTICE]  Defaults written to defaults-DIRAC.cfg
2019-04-11 14:47:02 UTC dirac-install [NOTICE]  Executing /home/diracuser/DIRAC/
↪scripts/dirac-externals-requirements...
2019-04-11 14:47:03 UTC dirac-install [NOTICE]  DIRAC properly installed
```

You will notice that among other things, the installation created a `~/DiracInstallation/bashrc` file. This file must be sourced whenever you want to use dirac client.

In principle, your system administrator will have managed the CA for you. In this specific case, since we have our own CA, we will just link the client installation CA with the server one:

```
mkdir -p ~/DiracInstallation/etc/grid-security/
ln -s /opt/dirac/etc/grid-security/certificates/ ~/DiracInstallation/etc/grid-
↪security/certificates
```

The last step is to configure the client to talk to the proper configuration service. This is easily done by creating a `~/DiracInstallation/etc/dirac.cfg` file with the following content:

```
DIRAC
{
  Setup = MyDIRAC-Production
  Configuration
  {
    Servers = dips://dirac-tuto:9135/Configuration/Server
  }
}
```

You should now be able to get a proxy:

```
[diracuser@dirac-tuto DIRAC]$ source ~/DiracInstallation/bashrc
[diracuser@dirac-tuto DIRAC]$ dirac-proxy-init
Generating proxy...
Proxy generated:
subject      : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch/
↪CN=460648814
issuer       : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
```

(continues on next page)

(continued from previous page)

```
identity     : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
timeleft     : 23:59:59
DIRAC group  : dirac_user
rfc          : True
path         : /tmp/x509up_u501
username     : ciuser
properties   : NormalUser
```

And you can observe that the Configuration Service has served the client:

```
[diracuser@dirac-tuto DIRAC]$ grep ciuser /opt/dirac/runit/Configuration/Server/log/
↪current
2019-04-11 14:54:10 UTC Configuration/Server NOTICE: Executing action␣
↪([::1]:33394)[dirac_user:ciuser] RPC/getCompressedDataIfNewer(<masked>)
2019-04-11 14:54:10 UTC Configuration/Server NOTICE: Returning response␣
↪([::1]:33394)[dirac_user:ciuser] (0.00 secs) OK
```

Use the WebApp

This section is to be executed as `diracuser`.

First you need to convert your user certificate into a `p12` format (you will be prompt for a password, you can leave it empty):

```
cd ~/.globus/
openssl pkcs12 -export -out certificate.p12 -inkey userkey.pem -in usercert.pem
```

This will create the file `~/.globus/certificate.p12`.

Use your favorite browser, and add this certificate.

You should be able to access the WebApp using the following address `https://localhost:8443/DIRAC/`

Conclusion

We have seen how to install a DIRAC server and client using a personal CA, and how to access the WebApp. Starting from here, you will be able to extend on further tutorials.

2.16.2 Managing identities

Pre-requisite

You should:

- have a machine setup as described in *Basic Tutorial setup*
- be able to install dirac components

Tutorial goal

Very quickly when using DIRAC, you will need to manage identities of people and their proxies. This is done with the `ProxyManager` service and with several configuration options. In this tutorial, we will install the `ProxyManager`, create a new group, and define some `Shifter`.

Further reading

- *Components authentication and authorization*
- *Manage authentification and authorizations*

Installing the `ProxyManager`

This section is to be performed as `diracuser` with `dirac_admin` group proxy.

The `ProxyManager` will host delegated proxies of the users. As any other service, it is very easy to install:

```
[dirac-tuto]> install db ProxyDB
MySQL root password:
Adding to CS Framework/ProxyDB
Database ProxyDB from DIRAC/FrameworkSystem installed successfully
[dirac-tuto]> install service Framework ProxyManager
Loading configuration template /home/diracuser/DiracInstallation/DIRAC/
↪FrameworkSystem/ConfigTemplate.cfg
Adding to CS service Framework/ProxyManager
service Framework_ProxyManager is installed, runit status: Run
```

Note: The ProxyDB contains sensitive information. For production environment, it is recommended that you keep this in a separate database with different credentials and strict access control.

Testing the `ProxyManager`

The simplest way to test it is to upload your user proxy:

```
[diracuser@dirac-tuto ~]$ dirac-proxy-init -U
Generating proxy...
Uploading proxy for dirac_user...
Proxy generated:
subject      : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch/
↪CN=6045995638
issuer       : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
identity     : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
timeleft     : 23:59:59
DIRAC group  : dirac_user
rfc          : True
path         : /tmp/x509up_u501
username     : ciuser
properties   : NormalUser

Proxies uploaded:
DN                                                                | Group      |␣
↪Until (GMT)
/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch | dirac_user |␣
↪2020/04/09 14:43
```

As you can see, the proxyDB now contains a delegated proxy for the `ciuser` with the group `dirac_user`.

If you use a proxy with the `ProxyManagement` permission, like the `dirac_admin` group has, you can retrieve proxies stored in the DB:

```
[diracuser@dirac-tuto ~]$ dirac-proxy-init -g dirac_admin
Generating proxy...
Proxy generated:
subject      : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch/
↪CN=5472309786
issuer       : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
identity     : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
timeleft     : 23:59:59
DIRAC group  : dirac_admin
rfc          : True
path         : /tmp/x509up_u501
username     : ciuser
properties   : AlarmsManagement, ServiceAdministrator, CSAdministrator,␣
↪JobAdministrator, FullDelegation, ProxyManagement, Operator
[diracuser@dirac-tuto ~]$ dirac-admin-get-proxy ciuser dirac_user
Proxy downloaded to /home/diracuser/proxy.ciuser.dirac_user
```

Adding a new group

Groups are useful to manage permissions and separate activities. For example, we will create a new group `dirac_data`, and decide to use that group for all the data centrally managed.

Using the `Configuration Manager` application in the WebApp, create a new section `dirac_data` in `/Registry/Groups`:

```
Users = ciuser
Properties = NormalUser
AutoUploadProxy = True
```

You should now be able to get a proxy belonging to the *dirac_data* group that will be automatically uploaded:

```
[diracuser@dirac-tuto ~]$ dirac-proxy-init -g dirac_data
Generating proxy...
Uploading proxy for dirac_data...
Proxy generated:
subject      : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch/
↪CN=6009266000
issuer       : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
identity     : /C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch
timeleft     : 23:59:59
DIRAC group  : dirac_data
rfc          : True
path         : /tmp/x509up_u501
username     : ciuser
properties   : NormalUser

Proxies uploaded:
DN                                                                | Group      |␣
↪Until (GMT)
/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch | dirac_data |␣
↪2020/04/09 14:43
/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch | dirac_user |␣
↪2020/04/09 14:43
```

Note: if you get `Unauthorized query ( 1111 : Unauthorized query)`, it means the ProxyManager

has not yet updated its internal configuration. Just restart it to save time, or wait.

Adding a Shifter

`Shifter` is basically a role, to which you associate a given proxy, for example `DataManager` (it could be anything). You can then tell your Components to use the `DataManager` identity to perform certain operations (at random: data management operations ? :-)).

Using the `Configuration Manager` application in the WebApp, create a new section `Shifter` in `/Operations/Defaults`:

```
DataManager
{
  User = ciuser
  Group = dirac_data
}
```

You can now force any agent (don't, unless you know what you are doing) to use a proxy instead of the host certificate by specifying the `shifterProxy` option.

2.16.3 Install a DIRAC Storage Element

Pre-requisite

You should have a machine setup as described in *Basic Tutorial setup*, and be able to install dirac components. For simple interaction with the StorageElement using `dirac-dms-*` commands, you should also have a working FileCatalog.

Tutorial goal

The aim of the tutorial is to do a step by step guide to install a DIRAC StorageElement. By the end of the tutorial, you will be able to have a fully functional storage element that can be addressed like any other storage.

Machine setup

This section is to be executed as `dirac` user.

We will simply create a folder where the files will be stored:

```
mkdir /opt/dirac/storageElementOne/
```

Installing the service

This section is to be executed as `diracuser` user, with `dirac_admin` proxy (reminder: `dirac-proxy-init -g dirac_admin`).

Install the StorageElement service using `dirac-admin-sysadmin-cli`:

```
[diracuser@dirac-tuto ~]$ dirac-admin-sysadmin-cli --host dirac-tuto
[dirac-tuto]> add instance DataManagement Production
Adding DataManagement system as Production self.instance for MyDIRAC-Production self.
↪setup to dirac.cfg and CS
DataManagement system instance Production added successfully
[dirac-tuto]> install service DataManagement StorageElement
Loading configuration template /home/diracuser/DIRAC/DIRAC/DataManagementSystem/
↪ConfigTemplate.cfg
Adding to CS service DataManagement/StorageElement
service DataManagement_StorageElement is installed, runit status: Run
[dirac-tuto]> quit
```

From the web interface, change the configuration of the StorageElement you just installed to point to the folder you created earlier:

```
Systems/DataManagement/Production/StorageElement/BasePath = /opt/dirac/
↪storageElementOne/
```

And restart the service:

```
[diracuser@dirac-tuto ~]$ dirac-admin-sysadmin-cli --host dirac-tuto
[dirac-tuto]> restart DataManagement StorageElement

DataManagement_StorageElement started successfully, runit status:

('   DataManagement_StorageElement', ':', 'Run')
```

You now have a Service offering grid like storage. However, you still need to declare a StorageElement for it to be usable within DIRAC.

Adding the StorageElement

You need to add a StorageElement in the `Resources/StorageElements` section. Using the WebApp, just add the following:

```
StorageElementOne
{
  BackendType = DISET
  DIP
  {
    Host = dirac-tuto
    Port = 9148
    Protocol = dips
    Path = /DataManagement/StorageElement
    Access = remote
  }
}
```

You now have a storage element that you can address as `StorageElementOne` in all the dirac commands or in your code.

Test it

Create a dummy file:

```
echo "dummyFile" > /tmp/dummy.txt
```

Now create a file called `/tmp/testSE.py`, with the following content:

```
from DIRAC.Core.Base.Script import parseCommandLine
parseCommandLine()

localFile = '/tmp/dummy.txt'
lfn = '/tutoVO/myFirstFile.txt'

from DIRAC.Resources.Storage.StorageElement import StorageElement


se = StorageElement('StorageElementOne')

print "Putting file"
print se.putFile({lfn: localFile})

print "Listing directory"
print se.listDirectory('/tutoVO')

print "Getting file"
print se.getFile(lfn, '/tmp/donwloaded.txt')

print "Removing file"
print se.removeFile(lfn)

print "Listing directory"
print se.listDirectory('/tutoVO')
```

This file uploads `/tmp/dummy.txt` on the StorageElement, list the directory and removes it. The output should be something like that:

```
[diracuser@dirac-tuto ~]$ python /tmp/testSE.py
Putting file
{'OK': True, 'Value': {'Successful': {'/tutoVO/myFirstFile.txt': 10}, 'Failed': {}}}
Listing directory
{'OK': True, 'Value': {'Successful': {'/tutoVO': {'Files': {'myFirstFile.txt': {
↪'Accessible': True, 'Migrated': 0, 'Unavailable': 0, 'Lost': 0, 'Exists': True,
↪'Cached': 1, 'Checksum': '166203b7', 'Mode': 420, 'File': True, 'Directory': True,
↪'TimeStamps': (1555342476, 1555342476, 1555342476), 'Type': 'File', 'Size': 10}},
↪'SubDirs': {}}}, 'Failed': {}}}
Getting file
{'OK': True, 'Value': {'Successful': {'/tutoVO/myFirstFile.txt': 10}, 'Failed': {}}}
Removing file
{'OK': True, 'Value': {'Successful': {'/tutoVO/myFirstFile.txt': True}, 'Failed': {}}}
Listing directory
{'OK': True, 'Value': {'Successful': {'/tutoVO': {'Files': {}, 'SubDirs': {}}},
↪'Failed': {}}}
```

Note: you might be getting the following message if you have no Accounting system. you can safely ignore it:

Error sending accounting record Cannot get URL for Accounting/DataStore in setup MyDIRAC-Production: RuntimeError('Option /DIRAC/Setups/MyDIRAC-Production/Accounting is not defined',)

Adding a second DIRAC SE

It is often interesting to have a second SE.

As `dirac` user, create a new directory:

```
mkdir /opt/dirac/storageElementTwo/
```

Now the rest is to be installed with `diracuser` and a proxy with `dirac_admin` group.

We need another StorageElement service. However, it has to have a different name than the first one, so we will just call this service `StorageElementTwo`:

```
[diracuser@dirac-tuto ~]$ dirac-admin-sysadmin-cli --host dirac-tuto
Pinging dirac-tuto...
[dirac-tuto]> install service DataManagement StorageElementTwo -m StorageElement -p␣
↪Port=9147
Loading configuration template /home/diracuser/DIRAC/DIRAC/DataManagementSystem/
↪ConfigTemplate.cfg
Adding to CS service DataManagement/StorageElementTwo
service DataManagement_StorageElementTwo is installed, runit status: Run
```

Using the WebApp, add the new StorageElement definition in the `/Resources/StorageElements` section:

```
StorageElementTwo
{
  BackendType = DISET
  DIP
  {
    Host = dirac-tuto
    Port = 9147
    Protocol = dips
    Path = /DataManagement/StorageElementTwo
    Access = remote
  }
}
```

In order to test it, just re-use `/tmp/testSE.py`, replacing `StorageElementOne` with `StorageElementTwo`

2.16.4 Installing the DIRAC File Catalog

Pre-requisite

You should:

- have a machine setup as described in *Basic Tutorial setup*
- be able to install dirac components
- have installed a DIRAC SE using the tutorial (*Install a DIRAC Storage Element*).

Tutorial goal

The aim of the tutorial is to install the DIRAC FileCatalog (DFC) By the end of the tutorial, you will be able to do all sort of simple Data Management operations.

More links

More information can be found at the following places:

- Introduction to DataManagement: *Data Management System*
- Catalog resource definition *Catalog*
- How-to datamanagement for user *DataManagement*

Installing the DFC

This section is to be executed as `diracuser` with a proxy with `dirac_admin` group.

The DFC is no different than any other DIRAC service with a database. The installation step are thus very simple:

```
[diracuser@dirac-tuto ~]$ dirac-admin-sysadmin-cli --host dirac-tuto
Pinging dirac-tuto...
[dirac-tuto]> install db FileCatalogDB
Adding to CS DataManagement/FileCatalogDB
Database FileCatalogDB from DIRAC/DataManagementSystem installed successfully
[dirac-tuto]> install service DataManagement FileCatalog
Loading configuration template /home/diracuser/DIRAC/DIRAC/DataManagementSystem/
↪ConfigTemplate.cfg
Adding to CS service DataManagement/FileCatalog
service DataManagement_FileCatalog is installed, runit status: Run
```

Adding the FileCatalog resource

In order to be used as a FileCatalog by clients, the DFC needs to be declared. This happens in two places:

- `/Resources/FileCatalogs/`: in this section, you define how to access the catalog
- `/Operations/Defaults/Services/Catalogs/`: in this section, you define how to use the catalog (for example read/write)

Since we have only one catalog, we will use it as `Read-Write` and as `Master`.

Using the WebApp, add the following in `/Resources/FileCatalogs/` (all options to defaults):

```
FileCatalog
{
}
```

Using the WebApp, add the following in `/Operations/Defaults/Services/Catalogs`:

```
FileCatalog
{
  AccessType = Read-Write
  Status = Active
  Master = True
}
```

From this moment onward, the catalog is totally usable.

Test the catalog

Since we have a StorageElement at our disposal, we can use the standard `dirac-dms-*` script.

First, let us create a file and then "put it on the grid":

```
[diracuser@dirac-tuto ~]$ echo "Hello" > /tmp/world.txt
[diracuser@dirac-tuto ~]$ dirac-dms-add-file /tutoVO/user/c/ciuser/world.txt /tmp/
↪world.txt StorageElementOne

Uploading /tutoVO/user/c/ciuser/world.txt
Successfully uploaded file to StorageElementOne
```

Now, let's check its replicas and metadata:

```
[diracuser@dirac-tuto ~]$ dirac-dms-lfn-replicas /tutoVO/user/c/ciuser/world.txt
LFN                              StorageElement    URL
=================================================
/tutoVO/user/c/ciuser/world.txt StorageElementOne dips://dirac-tuto:9148/
↪DataManagement/StorageElement/tutoVO/user/c/ciuser/world.txt

[diracuser@dirac-tuto ~]$ dirac-dms-lfn-metadata /tutoVO/user/c/ciuser/world.txt
{'Failed': {},
'Successful': {'/tutoVO/user/c/ciuser/world.txt': {'Checksum': '078b01ff',
                                                   'ChecksumType': 'Adler32',
                                                   'CreationDate': datetime.
↪datetime(2019, 4, 16, 9, 5, 58),
                                                   'FileID': 1L,
                                                   'GID': 1,
                                                   'GUID': '09F7E02F-1290-BE21-1DA7-
↪07A266F153B3',
                                                   'Mode': 509,
                                                   'ModificationDate': datetime.
↪datetime(2019, 4, 16, 9, 5, 58),
                                                   'Owner': 'ciuser',
                                                   'OwnerGroup': 'dirac_admin',
                                                   'Size': 6L,
                                                   'Status': 'AprioriGood',
                                                   'UID': 1}}}
```

Note that these metadata are those registered in the catalog (which hopefully should match the physical one !)

We can also check all the user files that belong to us on the grid:

```
[diracuser@dirac-tuto ~]$ dirac-dms-user-lfns
Will search for files in /tutoVO/user/c/ciuser
/tutoVO/user/c/ciuser: 1 files, 0 sub-directories
1 matched files have been put in tutoVO-user-c-ciuser.lfns
[diracuser@dirac-tuto ~]$ cat tutoVO-user-c-ciuser.lfns
/tutoVO/user/c/ciuser/world.txt
```

Finally, let's remove the file:

```
[diracuser@dirac-tuto ~]$ dirac-dms-remove-files /tutoVO/user/c/ciuser/world.txt
Successfully removed 1 files
```

2.16.5 Installing the RequestManagement System

Pre-requisite

You should:

- have a machine setup as described in *Basic Tutorial setup*
- be able to install dirac components
- have installed two DIRAC SE using the tutorial (*Install a DIRAC Storage Element*).
- have installed the DFC (*Installing the DIRAC File Catalog*)
- have followed the tutorial on identity management (*Managing identities*)

Tutorial goal

The aim of the tutorial is to install the RequestManagement system components and to use it to perform a simple replication of file.

More links

More information can be found at the following places:

- *Data Management System*
- *Request Management System*

Installing the RMS

This section is to be executed as `diracuser` with a proxy with `dirac_admin` group.

The RMS needs the `ReqManager` service and the `RequestExecutingAgent` to work (you may want to add the `CleanReqDBAgent` if you scale...).

The RMS is no different than any other DIRAC system. The installation step are thus very simple:

```
[diracuser@dirac-tuto ~]$ dirac-admin-sysadmin-cli --host dirac-tuto
Pinging dirac-tuto...
[dirac-tuto]> add instance RequestManagement Production
Adding RequestManagement system as Production self.instance for MyDIRAC-Production␣
↪self.setup to dirac.cfg and CS
RequestManagement system instance Production added successfully
[dirac-tuto]> install db ReqDB
MySQL root password:
Adding to CS RequestManagement/ReqDB
Database ReqDB from DIRAC/RequestManagementSystem installed successfully
[dirac-tuto]> install service RequestManagement ReqManager
Loading configuration template /home/diracuser/DiracInstallation/DIRAC/
↪RequestManagementSystem/ConfigTemplate.cfg
Adding to CS service RequestManagement/ReqManager
service RequestManagement_ReqManager is installed, runit status: Run
[dirac-tuto]> install agent RequestManagement RequestExecutingAgent
Loading configuration template /home/diracuser/DiracInstallation/DIRAC/
↪RequestManagementSystem/ConfigTemplate.cfg
Adding to CS agent RequestManagement/RequestExecutingAgent
agent RequestManagement_RequestExecutingAgent is installed, runit status: Run
[dirac-tuto]> quit
```

By default, the installation of the `RequestExecutingAgent` will configure it with a whole bunch of default Operations possible. You can see that in the Agent configuration in `/Systems/RequestManagement/Production/Agents/RequestExecutingAgent/OperationHandlers`

Testing the RMS

This section is to be executed with a proxy with *dirac_user* group.

The test we are going to do consists in transferring a file from one storage element to another, using the RequestExecutingAgent.

First, let's add a file:

```
[diracuser@dirac-tuto ~]$ echo "My Test File" > /tmp/myTestFile.txt
[diracuser@dirac-tuto ~]$ dirac-dms-add-file /tutoVO/user/c/ciuser/myTestFile.txt /
↪tmp/myTestFile.txt StorageElementOne

Uploading /tutoVO/user/c/ciuser/myTestFile.txt
Successfully uploaded file to StorageElementOne
```

We can see that our file is indeed in the `StorageElementOne`:

```
[diracuser@dirac-tuto ~]$ dirac-dms-lfn-replicas /tutoVO/user/c/ciuser/myTestFile.txt
LFN                                 StorageElement    URL
=============================================================
/tutoVO/user/c/ciuser/myTestFile.txt StorageElementOne dips://dirac-tuto:9148/
↪DataManagement/StorageElement/tutoVO/user/c/ciuser/myTestFile.txt
```

Let's replicate it to `StorageElementTwo` using the RMS:

```
[diracuser@dirac-tuto ~]$ dirac-dms-replicate-and-register-request myFirstRequest /
↪tutoVO/user/c/ciuser/myTestFile.txt StorageElementTwo
Request 'myFirstRequest' has been put to ReqDB for execution.
RequestID(s): 8
You can monitor requests' status using command: 'dirac-rms-request <requestName/ID>'
```

The Request has a name (`myFirstRequest`) that we chose, but also an ID, returned by the system (here "8"). The ID is guaranteed to be unique, while the name is not, so it is recommended to use the ID when you interact with the RMS. You can see the status of your Request, using its name or ID:

```
[diracuser@dirac-tuto ~]$ dirac-rms-request myFirstRequest
Request name='myFirstRequest' ID=8 Status='Waiting'
Created 2019-04-23 14:37:05, Updated 2019-04-23 14:37:05, NotBefore 2019-04-23␣
↪14:37:05
Owner: '/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch',␣
↪Group: dirac_data
  [0] Operation Type='ReplicateAndRegister' ID=8 Order=0 Status='Waiting'
      TargetSE: StorageElementTwo - Created 2019-04-23 14:37:05, Updated 2019-04-23␣
↪14:37:05
    [01] ID=2 LFN='/tutoVO/user/c/ciuser/myTestFile.txt' Status='Waiting' Checksum=
↪'1e750431'

[diracuser@dirac-tuto ~]$ dirac-rms-request 8
Request name='myFirstRequest' ID=8 Status='Waiting'
Created 2019-04-23 14:37:05, Updated 2019-04-23 14:37:05, NotBefore 2019-04-23␣
↪14:37:05
Owner: '/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch',␣
↪Group: dirac_data
```

(continues on next page)

(continued from previous page)

```
  [0] Operation Type='ReplicateAndRegister' ID=8 Order=0 Status='Waiting'
      TargetSE: StorageElementTwo - Created 2019-04-23 14:37:05, Updated 2019-04-23␣
↪14:37:05
    [01] ID=2 LFN='/tutoVO/user/c/ciuser/myTestFile.txt' Status='Waiting' Checksum=
↪'1e750431'
```

You can here clearly see that the Request consists of one `ReplicateAndRegister` operation (which does what it says) targeting the LFN `/tutoVO/user/c/ciuser/myTestFile.txt`. The RequestExecutingAgent will pick up the request and execute it. And shortly you should be able to see it done:

```
[diracuser@dirac-tuto ~]$ dirac-rms-request 8
Request name='myFirstRequest' ID=8 Status='Done'
Created 2019-04-23 14:37:05, Updated 2019-04-23 14:37:29, NotBefore 2019-04-23␣
↪14:37:05
Owner: '/C=ch/O=DIRAC/OU=DIRAC CI/CN=ciuser/emailAddress=lhcb-dirac-ci@cern.ch',␣
↪Group: dirac_data
  [0] Operation Type='ReplicateAndRegister' ID=8 Order=0 Status='Done'
      TargetSE: StorageElementTwo - Created 2019-04-23 14:37:05, Updated 2019-04-23␣
↪14:37:29
    [01] ID=2 LFN='/tutoVO/user/c/ciuser/myTestFile.txt' Status='Done' Checksum=
↪'1e750431'

[diracuser@dirac-tuto ~]$ dirac-dms-lfn-replicas /tutoVO/user/c/ciuser/myTestFile.txt
LFN                                    StorageElement    URL
=============================================================
/tutoVO/user/c/ciuser/myTestFile.txt StorageElementTwo dips://dirac-tuto:9147/
↪DataManagement/StorageElementTwo/tutoVO/user/c/ciuser/myTestFile.txt
                                     StorageElementOne dips://dirac-tuto:9148/
↪DataManagement/StorageElement/tutoVO/user/c/ciuser/myTestFile.txt
```

Conclusion

You now have an RMS in place, which is the base for all the asynchronous operations in DIRAC. This is used for big scale operations, failover, or even more !

2.16.6 Doing large scale DataManagement with the Transformation System

Pre-requisite

Pre-requisite

You should:

- have a machine setup as described in *Basic Tutorial setup*
- have installed two DIRAC SE using the tutorial (*Install a DIRAC Storage Element*).
- have installed the DFC using the tutorial (*Installing the DIRAC File Catalog*).
- have followed the tutorial on identity management (*Managing identities*)
- have installed the RMS using the tutorial (*Installing the RequestManagement System*)

Tutorial goal

The aim of the tutorial is to demonstrate how large scale data management operations (removals, replications, etc) can be achieved using the Transformation System. By the end of the tutorial, you will be able to:

- Submit simple transformation for manipulating a given list of files
- Have transformations automatically fed thanks to metadata
- Write your own plugin for TransformationSystem

More links

- *Transformation System*

Installing the RequestManagementSystem

This section is to be performed as `diracuser` with a proxy in `dirac_admin` group.

In order to have asynchronous operations handled in DIRAC, you need to have the RequestManagementSystem installed. For it to be functional, you need at least:

- The ReqManager: the service interfacing this system
- The RequestExecutingAgent: the agent performing the operations

CHAPTER 3

Developer Guide

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 *Developing in DIRAC: the 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 detailes 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 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.

3.1.1 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
}

Releases
{
  v1r2p3
  {
    depends = DIRAC:v5r12
  }

  v1r2p2
  {
    depends = DIRAC:v5r12p1
  }
}
RequiredExternals
{
  Server = tornado>=4.4.2, apache-libcloud==2.2.1
  Client = apache-libcloud==2.2.1
}
```

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).

The *RequiredExternals* section contains lists of extra python modules that can be installed with a *pip* installer for different installation types. Each module in the lists is specified in a format suitable to pass to the *pip* command.

An example with more than one module follows:

```
DefaultModules = MyExt
RequiredExtraModules = WebApp

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
  {
```

(continues on next page)

(continued from previous page)

```
    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 especified contains the required module version of the module.

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
BaseURL = http://diracproject.web.cern.ch/diracproject/tars/
#How to upload the release tarballs to the BaseURL
UploadCommand = ( cd %OUTLOCATION% ; tar -cf - %OUTFILENAMES% ) | ssh webuser@webhost
↪'cd /diracproject/tars &&  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.

3.1.2 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

```
#List of modules to be installed by default for the project
DefaultModules = MyExt
#Extra modules to be installed
RequiredExtraModules = WebApp

#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
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
```

(continues on next page)

(continued from previous page)

```
  ModulesToInstall = MyExt
  #Type of externals to install (client, client-full, server)
  ExternalsType = client
  #Version of lcg bundle to install
  LcgVer = v14r2
  #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
}
```

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
```

(continues on next page)

(continued from previous page)

```
  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*

3.2 Making DIRAC releases

This section is describing the procedure to follow by release managers when preparing new DIRAC releases (including patches).

3.2.1 Prerequisites

The release manager needs to:

- be aware of the DIRAC repository structure and branching.
- have push access to the "release" repository, so the one on GitHub (being part of the project "owners")

The release manager of LHCbDIRAC has the triple role of:

1. WebAppDIRAC release
2. creating the release
3. making basic verifications
4. deploying DIRAC tarballs

3.2.2 1. WebAppDIRAC release

Before you start releasing DIRAC, you have to install sencha cmd and you have to download extjs sdk.

Sencha Cmd

You can download from https://www.sencha.com/products/extjs/cmd-download/ Note: you have to add sencha to the system path. Please make sure, if you type sencha in the terminal it will work.

ExtJs SDK

If you are using DIRAC v6r20 series or later, You can download from the following link:

curl -O http://cdn.sencha.com/ext/gpl/ext-4.2.1-gpl.zip

otherwise:

https://www.sencha.com/legal/GPL/

Note: You have to provide a valid email address and you will receive a link where the sdk can be downloaded.

3.2.3 2. Creating the release(s)

The procedure consists of several steps:

- 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. First, just a note on *Pull Requests* on GitHub:

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). The PRs are also reviewed by autimated tools, like Travis (not limited to). 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.

Release notes

Release notes are contained in the *release.notes* file. Each release version has a dedicated section in this file, for example:

```
[v6r19p7]

*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.

The release notes for a given branch can be obtained with the *docs/Tools/GetReleaseNotes.py* script:

```
python docs/Tools/GetReleaseNotes.py --branches <branch> [<branch2>...] --date
↪<dateTheLastTagWasMade> [--openPRs]
```

Working with code and tags

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
> cd DIRAC
> git remote rename origin release
```

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-v6r19* while the new release branch *rel-v6r20* is already in production. This can be accomplished by the following sequence of commands, which will bring all the changes from the central repository including all the *release* branches. We now 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-v6r19 release/rel-v6r19
> vim release.notes
```

We can now start merging PRs, directly from GitHub. At the same time we edit the release notes to reflect what has been merged (please see the note below about how to edit this file). Once finished, save the file. Then, modify the __init__.py file of the root directory and define the version also there. Then we commit the changes (those done to release.notes and __init__.py) and update the current repository:

```
> git commit -a #this will commit the changes we made to the release notes in rel-
↪v6r19 local branch
> git fetch release #this will bring in the updated release/rel-v6r19 branch from the␣
↪github repository
> git rebase --no-ff release/rel-v6r19 #this will rebase the current rel-v6r19 branch␣
↪to the content of release/rel-v6r19
```

You can now proceed with tagging, pushing, and uploading:

```
> git tag v6r19p7 #this will create a tag, from the current branch, in the local␣
↪repository
> git push --tags release rel-v6r19 #we push to the *release* repository (so to␣
↪GitHub-hosted one) the tag just created, and the rel-v6r19 branch.
```

From the previous command, note that due to the fact that we are pushing a branch named *rel-v6r19* to the *release* repository, where it already exists a branch named *rel-v6r19*, the local branch will override the remote one.

All the patches must now be also propagated to the *upper* branches. In this example we are going through, we are supposing that it exists rel-v6r20 branch, from which we already derived production tags. We then have to propagate the changes done to rel-v6r19 to rel-v6r20. 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 v6r19p7 section.

```
> git checkout -b rel-v6r20 release/rel-v6r20 # We start by checking out the rel-
↪v6r20 branch
> git merge rel-v6r19 # Merge to rel-v6r20 what we have advanced in rel-v6r19
```

The last command may result in merge conflicts, which should be resolved "by hand". One typical conflict is about the content of the release.notes file.

From now on, the process will look very similar to what we have already done for creating tag v6r19p7. We should then repeat the process for v6r20:

```
> vim release.notes
> vim __init__.py
```

Merge PRs (if any), then save the files above. Then:

```
> git commit -a #this will commit the changes we made to the release notes in rel-
↪v6r20 local branch
> git fetch release #this will bring in the updated release/rel-v6r20 branch from the␣
↪github repository
> git rebase --no-ff release/rel-v6r20 #this will rebase the current rel-v6r20 branch␣
↪to the content of release/rel-v6r20
> git tag v6r20p2 #this will create a tag, from the current branch, in the local␣
↪repository
> git push --tags release rel-v6r20 #we push to the *release* repository (so to␣
↪GitHub-hosted one) the tag just created, and the rel-v6r20 branch.
```

The *master* branch of DIRAC always contains the latest stable release. If this corresponds to rel-v6r20, we should make sure that this is updated:

> git push release rel-v6r20:master

Repeat the process for every "upper" release branch.

The *integration* branch is also receiving new features to go into the next release. The *integration* branch also contains the *releases.cfg* file, which holds all the versions of DIRAC together with the dependencies among the different packages.

From the *integration* branch we also do all the tags of *pre-release* versions, that can be then installed with standard tools on test DIRAC servers.

The procedure for creating pre-releases is very similar to creating releases:

```
> git checkout -b integration release/integration
> git merge rel-v6r20 #replace with the "last" branch
> vim release.notes
> vim __init__.py
> vim releases.cfg #add the created tags (all of them, releases and pre-releases)
```

Merge all the PRs targeting integration that have been approved (if any), then save the files above. Then:

```
> git commit -a
> git fetch release #this will bring in the updated release/integration branch from␣
↪the github repository
> git rebase --no-ff release/integration #this will rebase the current integration␣
↪branch to the content of release/integration
> git tag v6r21-pre3 #this will create a tag, from the current branch, in the local␣
↪repository
> git push release integration
```

3.2.4 3. Making basic verifications

There are a set of basic tests that can be done on releases. The first test can be done even before creating a release tarball.

A first test is done automatically by Travis: https://travis-ci.org/DIRACGrid/DIRAC/branches

Travis also runs on all the Pull Requests, so if for all the PRs merged travis didn't show any problem, there's a good chance (but NOT the certainty) that the created tags are also sane.

A second test is represented by pylint, for which you may find some more info in section *Code quality*. Within Travis, we run also a "pylint –errors-only" test, which should be strictly equal to 0.

3.2.5 4. Deploying DIRAC tarballs

Once the release and integration branches are tagged and pushed, the new release and pre-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. To do this, just execute *dirac-distribution* command with the appropriate flags. For instance:

```
> dirac-distribution -r v6r19p7 -l DIRAC --extjspath=<extjs library path> for␣
↪example: /home/diracCertif/extjs/ext-4.2.1.883/
> dirac-distribution -r v6r20p2 -l DIRAC --extjspath=<extjs library path> for␣
↪example: /home/diracCertif/extjs/ext-4.2.1.883/
> dirac-distribution -r v6r21-pre3 -l DIRAC --extjspath=<extjs library path> for␣
↪example: /home/diracCertif/extjs/ext-4.2.1.883/
```

Note: if the sencha or extjs library is missing, the web will be not compiled.

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*).

It's now time to advertise that new releases have been created. Use the DIRAC google forum.

3.3 Development Model

The DIRACGrid Project is a pure open source project, advanced collectively by a distributed team of developers working in parallel on the core software as well as on various extensions. Everybody is welcome to participate.

The DIRACGrid project includes several repositories, all hosted in Github:

- DIRAC is the main repository: contains the client and server code
- WebAppDIRAC is the repository for the web portal
- Pilot is *not* a DIRAC extension, but a new version of the DIRAC pilots (dubbed Pilots 3.0)
- DIRACOS is the repository for the DIRAC dependencies
- Externals is the OLD repository for the DIRAC dependencies, going to be superseded by DIRACOS
- VMDIRAC is a DIRAC extension for using cloud sites
- COMDIRAC is a DIRAC extension of its CLI
- RESTDIRAC is a DIRAC extension that creates a REST layer for DIRAC services
- DB12 is *not* a DIRAC extension, but a self-contained quick benchmark

The content of the other repositories at *https://github.com/DIRACGrid* have either been included in those above, or became obsolete.

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.

3.3.1 Managing Code with Git

DIRAC uses Git to manage it's source code. Git is a Distributed Version Control System (DVCS). That means that there's no central repository like the one 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 Subversion do. Each developer has it's 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.

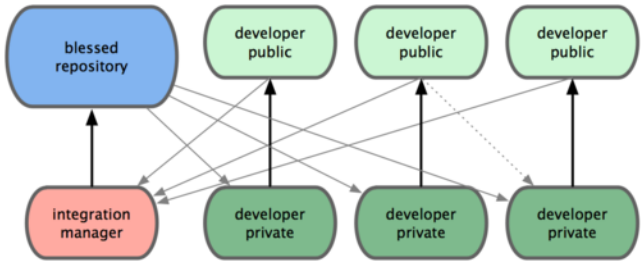


Fig. 1: How to publish and retrieve changes to DIRAC (see also Pro Git Book)

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.

3.3.2 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 interger 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-preZ`, 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-preZ`.

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* .

3.3.3 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::

```
git remote add release git@github.com:DIRACGrid/DIRAC.git
```

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 use 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:

```
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-v6r19 ::

```
git fetch release
git checkout -b fix-v6r19 release/rel-v6r19
```

Make the necessary changes to the code of the branch and then push them to the developer's fork::

```
git push origin fix-v6r19
```

Do the *PR* with the rel-v6r19 as a target branch. Once the *PR* is merged, scrap the patch branch from the forked repository::

```
git push origin :fix-v6r19
```

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*::

```
$ 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*.

3.4 Developing in DIRAC: the Development Environment

This chapter describes three ways to set up an environment for developing DIRAC software. Depending on what you need to do, different cases apply. You should anyway read this chapter from the beginning, even if you think that the

cases presented at the beginning do not apply to you.

3.4.1 Editing DIRAC code

What is this for?

Here we describe the suggested method for editing and unit-testing DIRAC code, and direct extensions of it.

What is this NOT for?

- This method is NOT specific for the WebAppDIRAC or Pilot code development, although several things that are described here can be applied.
- This method will NOT work out of box if you need 3rd party python libraries that are not pip installable.

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: as already explained, DIRAC's GIT repositories are hosted on GitHub. for which you need to have an account before continuing.

Stuff you need to have installed

python: make sure python 2.7.9+ (possibly 2.7.13) is installed and set as default (beware: the latest versions of Ubuntu use python 3.X as default, SLC6 use python 2.6 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-dependent, distribution dependent, and version dependent, so you'll need to figure it out by yourself how to install them. Some equivalent packages for Fedora/CentOS: python-devel, openssl-devel, mysql, ncurses-libs freetype, libjpeg-devel, MySQL-python. If you are using a OSX machine, then you may end up in more problems than using a linux box.

editor: get your favorite one. Examples include IDE like Eclipse or PyCharm, or whatever you prefer (vim, sublime, atom...) - anyway you'll need some plug-ins! I think atom and especially sublime (with the *anaconda* plugin) are excellent choices.

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*, just for our own convenience

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 (online!), then:

   git clone https://github.com/YOUR_GITHUB_USERNAME/DIRAC.git

Obviously, you must replace 'YOUR_GITHUB_USERNAME' with the username that we have registered on github. 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*:

```
cd DIRAC
git remote add release https://github.com/DIRACGrid/DIRAC.git
git fetch release
```

Repository structure

Just looking at the root directory:

```
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, one called Core, one Worfklow... maybe something else depending on the version you are looking at
- there is an __init__.py file
- there are some base files (README, LICENCE, etc.) and some dotfiles, which will become useful reading further.

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 one, or something slightly different. The only important thing is what you are going to put in the $PYTHONPATH.

Installing the dependencies

First, make sure that *setuptools* and *pip* are at the latest versions:

```
[sudo] pip install --upgrade setuptools
[sudo] pip install --upgrade pip
```

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 DIRAC # this creates the "DIRAC"
pip install -r $DEVROOT/DIRAC/requirements.txt
```

This will create a virtual python environment in which we can install all python packages that DIRAC use (this may take a while, and you might need to manually install some package from your distribution).

Some useful commands:

```
"pip install -r requirements.txt --upgrade" will upgrade the packages
"deactivate" will exit from a virtualenv
"workon DIRAC" will get you back in DIRAC virtualenv
```

NOTE: A (maybe better) *alternative* to virtualenv is conda, and specifically miniconda. Use it if you like, if you understand virtualenv you can understand conda environments.

Adding to the PYTHONPATH

You may either add the PATH to the global PYTHONPATH, as following:

```
export PYTHONPATH=$PYTHONPATH:$DEVROOT
```

And repeat for the extension development root, or use virtualenv for managing the path, using *add2virtualenv <http://virtualenvwrapper.readthedocs.io/en/latest/command_ref.html#add2virtualenv>*

Ready!

You're ready for DIRAC development! (or at least, good part of it). What can you do with what you have just done?

1. Editing code (this is the obvious!)
2. Running unit tests: please refer to *Testing (VO)DIRAC* for more info.
3. Running linters: please refer to *Code quality* for more info

So, this is "pure code"! And what you CAN'T do (yet)?

- You can't get a proxy
- you can't interact with configuration files nor with the Configuration System
- you can't run services, nor agents (no DIRAC components)

Next?

This depends from you: if you are a casual developer, you can stop here, and look into sections *Check your installation* and the following *Your first DIRAC code*

Alternatively, if you want to do more, you may proceed to the section *Developing stuff that runs*.

3.4.2 Developing *stuff that runs*

Which means developing for databases, services, agents, and executors. But also for the configuration service.

We'll guide you through using what we made in section *Editing DIRAC code* for developing and testing for databases, services, and agents. To do that, we'll create 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 those part of DIRAC that "run", e.g. databases, services, and agents. You need this type of installation for running so-called integration tests.

Do I need this?

Maybe. It depends from what you want to develop.

If you only need to develop "clients" and "utilities" code, you won't need this. But if you are going to change databases and DIRAC components, and if you want to run integration tests, you better keep reading.

Notes before continuing, on top of what is in section Editing DIRAC code

OS: a DIRAC server can be installed, as of today, only on SLC6 (Scientific Linux Cern 6) or CC7 (Cern CentOS 7).

The reason is that some binaries are proved to work only there (and TBH, support for CC7 is still partial), and this includes several WMS (Workload Management) and DMS (Data Management), like *arc* or *gfal2*. If you have to do many DMS (and partly WMS) developments, you should consider using SLC6 or CC7. Or, using a Virtual Machine, or a docker instance. We'll go through this.

Stuff you need to have installed, on top of what is in section Editing DIRAC code

If your development machine is a SLC6 or a CC7, probably nothing of what follows.

If your development machine is a Scientific Linux 6 or a CentOS 7, probably nothing of what follows, but I wouldn't be too sure about it.

If your development machine is a CentOS 6 or a RedHat "equivalent", maybe nothing of what follows, but I am even less sure about it.

If you are not in any of the above cases, you still have a chance: that, while developing for services or agents, none of them will need any "externals" library. If this is your case, then you can still run locally on your development machine, which can be for example Ubuntu, or Debian, or also macOS.

Do you need to develop using external, compiled libraries like *arc*, *cream*, *gfal2*, *fts3*?

Then you probably need a SLC6 or CC7 image. If your development box is not one of them, then you are presented with the alternatives of either using Virtual Machines, or Containers, and so in this case you'll need to install something:

docker: docker is as of today a "standard" for applications' containerization. The following examples use a DIRAC's base docker image for running DIRAC components.

a hypervisor, like *virtualbox*: if you don't want (or can't) use *docker*, you can use a virtual machine.

Whatever you need/decide, we will keep referring to your desktop as ''the host'', opposed to ''running image" (which, as just explained, may coincide with the host).

General principles while using a virtual machine or a container

- You keep editing the code on your host
- $DEVROOT should be mounted from the host to the running image
- The host and the running image should share the same configuration (dirac.cfg file)
- The DIRAC components are going to run on the running image
- The clients that contact the running components can be started on the host
- The running image should have a ''host certificate" for TLS verification and for running the components
- The host should have a ''user certificate''
- The user proxy should be created on the host for identifying the client

You can implement all the principles above in more than one way.

Using a Docker container [to expand]

The following steps will try to guide you on setting up a development environment for DIRAC (or its extensions) that combines what you have learned in *Editing DIRAC code* with a docker image with which you will run code that you develop.

Please see the Dockerfile that DIRAC provides at https://github.com/DIRACGrid/DIRAC/tree/integration/container and Docker hub []

[to expand]

What's in this image?

An dirac-install installed version of DIRAC (server).

[to expand]

Using a virtual machine [to expand]

Alternatively to docker...

Configuring DIRAC for running in an isolated environment

We'll configure DIRAC to work in isolation. At this point, the key becomes understanding how the DIRAC *Configuration Service (CS) :ref:'dirac-cs-structure* 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:

```
DIRAC
{
  Setup = DeveloperSetup
  Setups
  {
    DeveloperSetup
    {
      Framework = DevInstance
      Test = DevInstance
    }
  }
}
Systems
{
  Framework
  {
    DevInstance
    {
      URLs
      {
      }
      Services
      {
      }
    }
  }
  Test
  {
    DevInstance
    {
      URLs
      {
      }
      Services
      {
      }
    }
  }
}
Registry
{
  Users
```

(continues on next page)

(continued from previous page)

```
  {
    yourusername
    {
      DN = /your/dn/goes/here
      Email = youremail@yourprovider.com
    }
  }
  Groups
  {
    devGroup
    {
      Users = yourusername
      Properties = CSAdministrator, JobAdministrator, ServiceAdministrator,␣
↪ProxyDelegation, FullDelegation
    }
  }
  Hosts
  {
    mydevbox
    {
      DN = /your/box/dn/goes/here
      Properties = CSAdministrator, JobAdministrator, ServiceAdministrator,␣
↪ProxyDelegation, FullDelegation
    }
  }
}
```

Within the code we also provide a pre-filled example of dirac.cfg. You can get it simply doing (on the host):

```
cp $DEVROOT/DIRAC/docs/source/DeveloperGuide/AddingNewComponents/dirac.cfg.basic.
↪example $DEVROOT/etc/dirac.cfg
```

Scripts (DIRAC commands)

DIRAC scripts can be found in (almost) every DIRAC package. For example in DIRAC.WorkloadManagementSystem.scripts. You can invoke them directly, or you can run the command:

```
dirac-deploy-scripts
```

which will inspect all these directories (including possible DIRAC extensions) and deploy the found scripts in $DEVROOT/scripts. Developers can then persist this directory in the $PATH.

Certificates

By default, all connections to/from DIRAC services are secured, by with TLS/SSL security, so X.509 certificates need to be used. This sub-section explains how to create (with few openSSL commands) a Certification Authority (CA), and with that sign user and host certificates. This CA would be a in-house CA, so its certificates won't be trusted by anyone.

Still, you CAN run DIRAC services without any certificate. The reason is that, while the use of TLS/SSL and certificates is the default, you can still go away without it, simply disabling TLS/SSL. You'll see how later. So, if you find difficulties with this subsection, the good news is that you don't strictly need it.

Anyway: DIRAC understands certificates in *pem* format. That means that a 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
generateCA
generateCertificates 365
generateUserCredentials 365
mkdir -p ~/.globus/
cp $DEVROOT/user/*.{pem,key} ~/.globus/
mv ~/.globus/client.key ~/.globus/userkey.pem
mv ~/.globus/client.pem ~/.globus/usercert.pem
```

Now we need to register those certificates in DIRAC. To do so you must modify *$DEVROOT/etc/dirac.cfg* file and set the correct certificate DNs for you and your development box. To register the host, replace "/your/box/dn/goes/here" (/Registry/Hosts/mydevbox/DN option) with the result of:

```
openssl x509 -noout -subject -in $DEVROOT/etc/grid-security/hostcert.pem | sed 's:^
↪subject= ::g'
```

Same process to register yourself, replace "/your/dn/goes/here" (/Registry/Users/yourusername/DN option) with the result of:

```
openssl x509 -noout -subject -in ~/.globus/usercert.pem | sed 's:^subject= ::g'
```

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 python shell (iPython is included in the requirements.txt file).

From the host:

```
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, still from the host:

```
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.

The next test, also executed from the host, will verify if you will be able to produce a proxy starting from the user certificates that you have created above:

```
X509_CERT_DIR=$DEVROOT/etc/grid-security/certificates ./FrameworkSystem/scripts/dirac-
↪proxy-init.py -ddd
```

Should return you a user proxy. You can verify the content and location of the proxy with:

```
X509_CERT_DIR=$DEVROOT/etc/grid-security/certificates ./FrameworkSystem/scripts/dirac-
↪proxy-info.py
```

Then, you can login on your running image (or your local installation) and try running a service, using the dips protocol.

Do not think about you just typed right now. It will become more clear later. Please, look into *Check your installation* section for further checks.

Ready!

You're (even more) ready for DIRAC development! What can you do with what you have just done? Everything that was in the previous section, and on top:

1. Developing and testing code that "run"
2. Developing and testing code that requires integration between different components, like services and databases, but also agents
3. Running integration tests: please refer to *Testing (VO)DIRAC* (towards the end) for more info.

And what you CAN'T do (yet)?

- you can't interact with a ''production" setup, unless you use valid certificates
- you can't develop for web portal pages, because browsers won't accept self-signed certificates

3.4.3 Interacting with the production environment

Which means developing, while interacting with an existing production environment.

In the end, it's a matter of being correctly authenticated and authorized. So, the only real thing that you need to have is:

- a DIRAC developer installation
- a (real) certificate, that is recognized by your server installation
- a dirac.cfg that include the (real) setup of the production environment that you want to connect to (in DIRAC/Setup section)
- a dirac.cfg that include the (real) URL of the production Configuration server.

The last 2 bullets can be achieved with the following command:

```
dirac-configure -S MyProductionSetup -C dips://some.whe.re:9135/Configuration/Server -
↪-SkipCAChecks
```

Or simply by manual editing the dirac.cfg file.

From now on, you need to be extremely careful with whatever you do, because your development installation ends up not being anymore a "close" installation.

3.5 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.

3.5.1 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.

3.5.2 DIRAC Architecture

DIRAC follows the paradigm of a Services Oriented Architecture (SOA).

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.

3.5.3 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:

- DISET (DIRAC Secure Transport) secure communication protocol
- Web Portal framework
- Configuration System
- Logging System
- Monitoring System

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.

3.6 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.

3.6.1 Pep8, Pycodestyle and autopep8

In order to ensure consistent formatting between developers, it was decided to stick to the Pep8 style guide (https://www.python.org/dev/peps/pep-0008/), with two differences: * we use 2 space indentation instead of 4 * we use a line length of 120 instead of 80

This is managed by the setup.cfg at the root of the DIRAC repository.

In order to ensure that the formatting preference of the developer's editor does not play trick, there are two files under *tests/formatting*: *pep8_bad.py* and *pep8_good.py*. The first one contains generic rules and examples of dos and donts. The developer should pass this file through the autoformat of his/her editor. The output should be exactly *pep8_good.py*. We recommand the use of autopep8 for the autoformatting:

```
[chaen@pclhcb31 formatting]$ pycodestyle pep8_bad.py
pep8_bad.py:11:121: E501 line too long (153 > 120 characters)
pep8_bad.py:15:121: E501 line too long (124 > 120 characters)
pep8_bad.py:26:1: E303 too many blank lines (3)
pep8_bad.py:28:23: E401 multiple imports on one line
pep8_bad.py:73:3: E741 ambiguous variable name 'l'
pep8_bad.py:74:3: E741 ambiguous variable name 'O'
pep8_bad.py:75:3: E741 ambiguous variable name 'I'
pep8_bad.py:79:42: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:79:44: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:79:62: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:79:64: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:79:82: E231 missing whitespace after ','
pep8_bad.py:79:89: E231 missing whitespace after ','
pep8_bad.py:79:99: E231 missing whitespace after ','
pep8_bad.py:79:106: E231 missing whitespace after ','
pep8_bad.py:79:117: E231 missing whitespace after ','
pep8_bad.py:79:121: E501 line too long (153 > 120 characters)
pep8_bad.py:79:126: E231 missing whitespace after ','
pep8_bad.py:79:148: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:79:150: E251 unexpected spaces around keyword / parameter equals
pep8_bad.py:108:1: E303 too many blank lines (3)


[chaen@pclhcb31 formatting]$ autopep8 pep8_bad.py > myAutoFormat.py

[chaen@pclhcb31 formatting]$ pycodestyle myAutoFormat.py
myAutoFormat.py:11:121: E501 line too long (153 > 120 characters)
myAutoFormat.py:74:3: E741 ambiguous variable name 'l'
myAutoFormat.py:75:3: E741 ambiguous variable name 'O'
myAutoFormat.py:76:3: E741 ambiguous variable name 'I'
```

(continues on next page)

(continued from previous page)

```
[chaen@pclhcb31 formatting]$ diff myAutoFormat.py pep8_good.py
[chaen@pclhcb31 formatting]$
```

Note that pycodestyle will still complain about the ambiguous variable in the good file, since autopep8 will not remove them. Also, autopep8 will not modify comment inside docstrings, hence the first warning on the good file.

3.6.2 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 git 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*.

3.6.3 Module Coding Conventions

R3 Each module should define the following variables in its global scope:

```
__RCSID__ = "$Id$"
```

this is the SVN macro substituted by the module revision number.

```
__docformat__ = "restructedtext 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 ReStructedText mark-up language.

Importing modules

R5 Standard python modules are imported using:

```
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.

3.6.4 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 120 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 120 column limit.

Comments and doc strings

Comments should be abundant, and must follow the rules of automatic documentation by the sphinx tool using ReStructedText 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.

3.7 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.

3.7.1 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 read about *Developing in DIRAC: the Development Environment*, at least until the *Editing DIRAC code* part.

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.

Make sure that you are running these commands inside the python virtual environment that you have created with *virtualenv* as explained in *Editing DIRAC code*.

```
In [1]: import GSI
In [2]: import pyparsing
In [3]: import MySQLdb
In [4]: import DIRAC
```

Were these imports OK? If not, then you should probably hit the "previous" button of this guide, or check the *pip install* log.

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]: False
```

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. You will find more details on gLogger in the *gLogger* documentation.

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'}

In [13]: S_ERROR('Damn it')
Out[13]:  {'Errno': 0, 'Message': 'Damn it', 'OK': False, 'CallStack': ['  File "
↪<stdin>", line 1, in <module>\n']}

In [14]: S_ERROR( errno.EPERM, 'But I want to!')
Out[14]:  {'Errno': 1, 'Message': 'Operation not permitted ( 1 : But I want to!)', 'OK
↪': False, 'CallStack': ['  File "<stdin>", line 1, in <module>\n']}
```

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'])
  return result
else:
  returnedValue = result['Value']
```

Playing with the Configuration Service

Note: please, read and complete *Developing stuff that runs* before continuing.

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 confortable, 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/lib'
```

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
  {
```

(continues on next page)

(continued from previous page)

```
    Users = fstagni
  }
}
```

You can keep playing with it (e.g. adding some properties), but for the moment this is enough.

3.7.2 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:

```
> 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:

```
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:

```
# 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
```

(continues on next page)

(continued from previous page)

```
class TestcheckCAOfUserSuccess(TestcheckCAOfUser):

  def test_success( self ):
    self.gConfigMock.getValue.return_value = 'attendedValue'
    res = checkCAOfUser( 'aUser', 'attendedValue' )
    self.assertTrue(res['OK'])

  def test_failure( self ):
    self.gConfigMock.getValue.return_value = 'unAttendedValue'
    res = checkCAOfUser( 'aUser', 'attendedValue' )
    self.assertFalse( res['OK'] )


if __name__ == '__main__':
  suite = unittest.defaultTestLoader.loadTestsFromTestCase( TestcheckCAOfUser )
  suite.addTest( unittest.defaultTestLoader.loadTestsFromTestCase(␣
↪TestcheckCAOfUserSuccess ) )
  testResult = unittest.TextTestRunner( verbosity = 2 ).run( suite )
```

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 wih info or error mode depending on the result

Remember to start the script with:

```
#!/usr/bin/env python
""" Some doc: what does this script should do?
"""
from DIRAC.Core.Base import Script
Script.parseCommandLine()
```

Then run it.

3.7.3 Developing Services

Service Handler

All the DIRAC Services are built in the same framework. 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. This means that developers should be aware that 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:

```
""" Hello Service is an example of how to build services in the DIRAC framework
"""

__RCSID__ = "$Id$"

from DIRAC import gLogger, S_OK, S_ERROR
from DIRAC.Core.DISET.RequestHandler import RequestHandler

class HelloHandler(RequestHandler):

  @classmethod
  def initializeHandler(cls, serviceInfo):
    """ Handler initialization
    """
    cls.defaultWhom = "World"
    return S_OK()

  def initialize(self):
    """ Response initialization
    """
    self.requestDefaultWhom = self.srv_getCSOption("DefaultWhom", HelloHandler.
↪defaultWhom)

  auth_sayHello = ['all']
  types_sayHello = [basestring]
  def export_sayHello(self, whom):
    """ Say hello to somebody
    """
    gLogger.notice("Called sayHello of HelloHandler with whom = %s" % whom)
    if not whom:
      whom = self.requestDefaultWhom
    if whom.lower() == 'nobody':
      return S_ERROR("Not greeting anybody!")
    return S_OK("Hello " + whom)
```

Let us walk through this code to see which elements should be provided.

The first lines shows 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. The ''__RCSID__" is only used for keeping the last committer and the timestamp of the last commit.

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. Within this method a developer 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.

No ''__init__" method is specified, and, by construction, it should not be.

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, but please use this possibility only for doing local tests (see later). 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. There is also the possibility to define authentication rules in the Configuration Service.

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 can also be imported from the ''types" standard python module.

Default Service Configuration parameters

The Hello Handler is written. There's not even the need to copy/paste, because you can do:

```
cp $DEVROOT/DIRAC/docs/source/DeveloperGuide/AddingNewComponents/DevelopingServices/
↪HelloHandler.py $DEVROOT/DIRAC/FrameworkSystem/Service/
```

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 service will need 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".

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 following file can be used as dirac.cfg file,

```
LocalSite
{
  Site = DIRAC.DevBox.org
}
DIRAC
{
  Setup = DeveloperSetup
  Setups
  {
    DeveloperSetup
    {
      Accounting = DevInstance
      Configuration = DevInstance
      DataManagement = DevInstance
      Framework = DevInstance
      Monitoring = DevInstance
      RequestManagement = DevInstance
      ResourceStatus = DevInstance
      StorageManagement = DevInstance
      Transformation = DevInstance
      WorkloadManagement = DevInstance
    }
  }
}
Systems
{
  Database
  {
```

(continues on next page)

(continued from previous page)

```
    User = Dirac
    Password = Dirac
    RootPwd = Dirac
    Host = localhost
    RootUser = root
  }
  NoSQLDatabase
  {
    User = Dirac
    Password = Dirac
    Host = localhost
    Port = 9203
  }
  Accounting
  {
    DevInstance
    {
      Agents
      {
      }
      URLs
      {
      }
      Services
      {
      }
      Databases
      {
      }
    }
  }
  Framework
  {
    DevInstance
    {
      URLs
      {
        Hello = dips://localhost:3424/Framework/Hello
      }
      Services
      {
        Hello
        {
          Port = 3424
          DisableMonitoring = yes
          Authorization
          {
            Default = all
          }
        }
      }
    }
  }
  ResourceStatus
  {
    DevInstance
    {
```

(continues on next page)

(continued from previous page)

```
    Agents
    {
    }
    URLs
    {
    }
    Services
    {
    }
    Databases
    {
    }
  }
}
WorkloadManagement
{
  DevInstance
  {
    Agents
    {
    }
    URLs
    {
    }
    Services
    {
    }
    Databases
    {
    }
    Executors
    {
    }
  }
}
Transformation
{
  DevInstance
  {
    Agents
    {
    }
    URLs
    {
    }
    Services
    {
    }
    Databases
    {
    }
  }
}
RequestManagement
{
  DevInstance
  {
```

(continues on next page)

(continued from previous page)

```
      Agents
      {
      }
      URLs
      {
      }
      Services
      {
      }
      Databases
      {
      }
    }
  }
  DataManagement
  {
    DevInstance
    {
      Agents
      {
      }
      URLs
      {
      }
      Services
      {
      }
      Databases
      {
      }
    }
  }
}
Registry
{
  DefaultGroup = users
  Users
  {
    diracuser
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=diracuser/emailAddress=diracuser@diracgrid.org
      Email = diracuser@diracgrid.org
    }
  }
  Groups
  {
    users
    {
      Users = diracuser
      Properties = NormalUser
    }
    dirac_admin
    {
      Users = diracuser
      Properties = AlarmsManagement
      Properties += ServiceAdministrator
      Properties += CSAdministrator
```

(continues on next page)

(continued from previous page)

```
      Properties += JobAdministrator
      Properties += FullDelegation
      Properties += ProxyManagement
      Properties += Operator
    }
    prod
    {
      Users = diracuser
      Properties = Operator
      Properties += FullDelegation
      Properties += ProxyManagement
      Properties += ServiceAdministrator
      Properties += JobAdministrator
      Properties += CSAdministrator
      Properties += AlarmsManagement
      Properties += FileCatalogManagement
      Properties += SiteManager
      Properties += NormalUser
    }
  }
  Hosts
  {
    DIRACDockerDevBox
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=DIRACDockerDevBox/
↪emailAddress=DIRACDockerDevBox@diracgrid.org
      Properties = JobAdministrator
      Properties += FullDelegation
      Properties += Operator
      Properties += CSAdministrator
      Properties += ProductionManagement
      Properties += AlarmsManagement
      Properties += TrustedHost
      Properties += SiteManager
    }
    DIRACVMDevBox
    {
      DN = /C=ch/O=DIRAC/OU=DIRAC CI/CN=DIRACVMDevBox/
↪emailAddress=DIRACVMDevBox@diracgrid.org
      Properties = JobAdministrator
      Properties += FullDelegation
      Properties += Operator
      Properties += CSAdministrator
      Properties += ProductionManagement
      Properties += AlarmsManagement
      Properties += TrustedHost
      Properties += SiteManager
    }
  }
  DefaultGroup = users
}
```

Again, there's no need to copy/paste, because you can do:

```
cp $DEVROOT/docs/source/DeveloperGuide/AddingNewComponents/DevelopingServices/dirac.
↪cfg.service.example $DEVROOT/etc/dirac.cfg
```

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.

If everything goes well, you should see something like:

```
2014-05-23 13:58:04 UTC Framework/Hello[MppQ] ALWAYS: Listening at dips://
↪localhost:3234/Framework/Hello
```

The URL displayed should be added to the local *dirac.cfg* in the URLs section (for this example, it already is).

Just a quick note on the URL displayed: it starts with "dips://". "dip" stands for *DISET protocol* and the "s" is for "secure", which for DIRAC means using X509 based authentication.

While "secure" is the default, it is also possible to run, for testing purpose, in unsecure way, which translates into using a "dip://" URL. For pure testing purpose this is often a convenience (no need for proxies nor certificates). If you want to run your services using the "dip" protocol, use the following configuration:

```
Services
{
  Hello
  {
    Port = 3424
    DefaultWhom = Universe
    Protocol = dip
  }
}
```

which is the same configuration used above with the difference of the "Protocol = dip" line.

Now, going back for a second on the service calls authorizations: in the example above we have used *auth_<method_name>* to define the service authorization properties. What we have done above can be achieved using the following CS structure:

```
Services
{
  Hello
  {
    Port = 3424
    DefaultWhom = Universe
    Authorization
```

(continues on next page)

(continued from previous page)

```
    {
      sayHello = all
    }
  }
}
```

and removing the *auth_<method_name>* from the code. This is a better "production" level coding.

You can also specify which default authorizations a service call should have at deploy time by editing the "ConfigTemplate.cfg" file present in every system. An example can be found in https://github.com/DIRACGrid/DIRAC/blob/integration/WorkloadManagementSystem/ConfigTemplate.cfg

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:

```
# Needed for stand alone tests
from DIRAC.Core.Base.Script import parseCommandLine
parseCommandLine(ignoreErrors=False)

from DIRAC.Core.Base.Client import Client

simpleMessageService = Client()
simpleMessageService.serverURL = 'Framework/Hello'
result = simpleMessageService.sayHello('you')
if not result['OK']:
  print "Error while calling the service:", result['Message'] #Here, in DIRAC, you␣
↪better use the gLogger
else:
  print result[ 'Value' ] #Here, in DIRAC, you better use the gLogger
```

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.

3.7.4 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:

```
# 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 ):
    pass

class TestHelloHandlerSuccess( TestHelloHandler ):

  def test_success( self ):


class TestHelloHandlerFailure( TestHelloHandler ):

  def test_failure( self ):



if __name__ == '__main__':
  suite = unittest.defaultTestLoader.loadTestsFromTestCase( TestHelloHandler )
  suite.addTest( unittest.defaultTestLoader.loadTestsFromTestCase(␣
↪TestHelloHandlerSuccess ) )
  suite.addTest( unittest.defaultTestLoader.loadTestsFromTestCase(␣
↪TestHelloHandlerFailure ) )
  testResult = unittest.TextTestRunner( verbosity = 2 ).run( suite )
```

As said, examples can be found in the DIRAC/tests package.

3.7.5 Developing Databases

This is a quick guide about developing classes interacting with MySQL databases. DIRAC supports also Oracle SQL DB, and also the NoSQL database and ElasticSearch, but they are not part of this document.

Before starting developing databases, you have to make sure that MySQL is installed, as well as python-mysql, as explained in *Editing DIRAC code*, and make sure that MySQL service is on.

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.

The `DIRAC.Core.Base.DB` module provides a base class for defining and interacting with MySQL DBs. The example that follows make use of it. There is also the possibility to define the DB using *sqlalchemy* python package, (and some DIRAC DBs do indeed that) but this is not covered in this document.

A simple example of the python class of a database follows:

```
""" A test DB in DIRAC, using MySQL as backend
"""

from DIRAC.Core.Base.DB import DB


class AtomDB(DB):

  def __init__(self):
    DB.__init__(self, 'AtomDB', 'Test/AtomDB')
    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']
                                 }

    return self._createTables(tablesD)

  def addStuff(self, something):
    return self.insertFields('atom_mytable', ['stuff'], [something])
```

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 2 or 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. Boolean for the debug flag

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:

```
Systems
{
  Test
  {
    <instance name>
    {
      Databases
      {
        AtomDB
        {
          Host = localhost
          User = yourusername
          Password = yourpasswd
          DBName = yourdbname
        }
      }
    }
}
```

In a production environment, the "Password" should be defined in a non-accessible file, while the rest of the configuration can go in the central Configuration Service.

If you encounter any problem with sockets, you should replace "localhost" (DIRAC/Systems/Test/<instance name>/AtomDB/Host) by 127.0.0.1.

Keep in mind that <instance name> is the name of the instance defined under */DIRAC/Setups/<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:

```
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
```

3.7.6 Testing a DB while developing it

For testing a DB code, we suggests 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.

3.7.7 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:

```
""" :mod: SimplestAgent

    Simplest Agent send a simple log message
"""

# # imports
from DIRAC import S_OK, S_ERROR
from DIRAC.Core.Base.AgentModule import AgentModule
from DIRAC.Core.DISET.RPCClient import RPCClient


__RCSID__ = "Id$"

class SimplestAgent(AgentModule):
  """
  .. class:: SimplestAgent
```

(continues on next page)

(continued from previous page)

```
  Simplest agent
  print a message on log
  """

  def initialize(self):
    """ agent's initalisation

    :param self: self reference
    """
    self.message = self.am_getOption('Message', "SimplestAgent is working...")
    self.log.info("message = %s" % self.message)
    return S_OK()

  def execute(self):
    """ execution in one agent's cycle

    :param self: self reference
    """
    self.log.info("message is: %s" % self.message)
    simpleMessageService = RPCClient('Framework/Hello')
    result = simpleMessageService.sayHello(self.message)
    if not result['OK']:
      self.log.error("Error while calling the service: %s" % result['Message'])
      return result
    self.log.info("Result of the request is %s" % result['Value'])
    return S_OK()
```

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 not that the __init__ method cannot be used when developing an Agent. It is used to intialize 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.

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:

```
Agents
{
  ##BEGIN SimplestAgent
  SimplestAgent
  {
    LogLevel = INFO
```

(continues on next page)

(continued from previous page)

```
    LogBackends = stdout
    PollingTime = 60
    Message = still working...
  }
  ##END
}
```

'PollingTime' defines the time between cycles, 'Message' is this agent specific option. ##BEGIN SimplestAgent and ##END are used to automagically include the agent's documentation into the docstring of the agents' module, by placing this snippet there, see *Component Options documentation*

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:

```
> 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:

```
> 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

In case you are running a SystemAdministrator service, you'll be able to log in to the machine using (as administrator) *dirac-admin-sysadmin-cli* and show the log of SimplestAgent using:

```
> show log Framework SimplestAgent
```

An info message will appear in log:

```
Framework/SimplestAgent  INFO: message: still working...
```

Note that the service is always returning the result in the form of S_OK/S_ERROR structure.

3.7.8 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 DIRAC/tests 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:

```
.. code-block:: 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

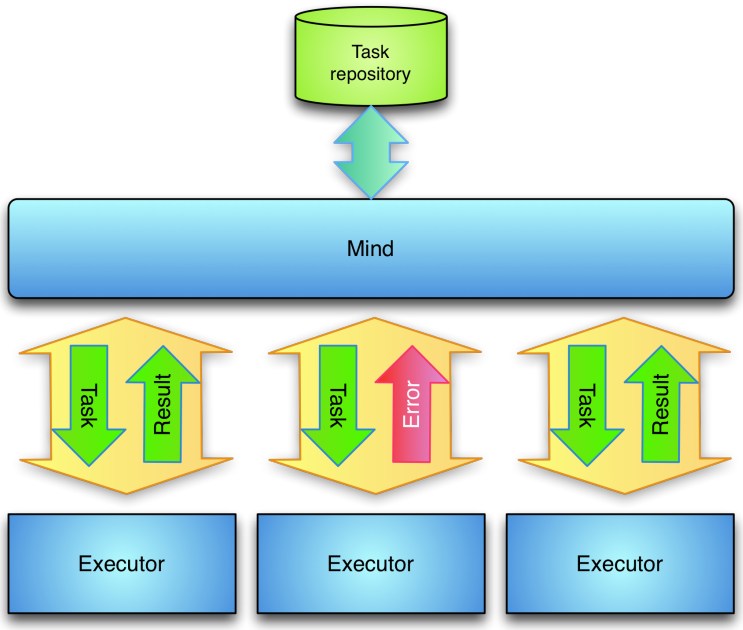
def test_myTest(self): bla

if __name__ == '__main__': suite = unittest.defaultTestLoader.loadTestsFromTestCase(MyAgentTestCase) testResult = unittest.TextTestResult(verbosity = 2).run(suite)

3.7.9 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:

```
import threading
from DIRAC import S_OK
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.
    """
```

(continues on next page)

(continued from previous page)

```
    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))
```

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*.

Funciton *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. Jusk 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:

```
""" Example of ExecutorMindHandler implementation
"""

import time
import random
from DIRAC import S_OK, gLogger
from DIRAC.Core.Utilities import DEncode
from DIRAC.Core.Base.ExecutorMindHandler import ExecutorMindHandler

random.seed()


class PingPongMindHandler(ExecutorMindHandler):

  MSG_DEFINITIONS = {'StartReaction': {'numBounces': (int, long)}}

  auth_msg_StartReaction = ['all']
```

(continues on next page)

(continued from previous page)

```
18
19    def msg_StartReaction(self, msgObj):
20      bouncesLeft = msgObj.numBounces
21      taskid = time.time() + random.random()
22      taskData = {'bouncesLeft': bouncesLeft}
23      return self.executeTask(time.time() + random.random(), taskData)
24
25    auth_startPingOfDeath = ['all']
26    types_startPingOfDeath = [int]
27
28    def export_startPingOfDeath(self, numBounces):
29      taskData = {'bouncesLeft': numBounces}
30      gLogger.info("START TASK = %s" % taskData)
31      return self.executeTask(int(time.time() + random.random()), taskData)
32
33    @classmethod
34    def exec_executorConnected(cls, trid, eTypes):
35      """
36      This function will be called any time an executor reactor connects
37
38      eTypes is a list of executor modules the reactor runs
39      """
40      gLogger.info("EXECUTOR CONNECTED OF TYPE %s" % eTypes)
41      return S_OK()
42
43    @classmethod
44    def exec_executorDisconnected(cls, trid):
45      """
46      This function will be called any time an executor disconnects
47      """
48      return S_OK()
49
50    @classmethod
51    def exec_dispatch(cls, taskid, taskData, pathExecuted):
52      """
53      Before a task can be executed, the mind has to know which executor module can␣
    ↪process it
54      """
55      gLogger.info("IN DISPATCH %s" % taskData)
56      if taskData['bouncesLeft'] > 0:
57        gLogger.info("SEND TO PLACE")
58        return S_OK("Test/PingPongExecutor")
59      return S_OK()
60
61    @classmethod
62    def exec_prepareToSend(cls, taskId, taskData, trid):
63      """
64      """
65      return S_OK()
66
67    @classmethod
68    def exec_serializeTask(cls, taskData):
69      gLogger.info("SERIALIZE %s" % taskData)
70      return S_OK(DEncode.encode(taskData))
71
72    @classmethod
73    def exec_deserializeTask(cls, taskStub):
```

(continues on next page)

(continued from previous page)

```
74      gLogger.info("DESERIALIZE %s" % taskStub)
75      return S_OK(DEncode.decode(taskStub)[0])
76
77    @classmethod
78    def exec_taskProcessed(cls, taskid, taskData, eType):
79      """
80      This function will be called when a task has been processed and by which executor␣
    ↪module
81      """
82      gLogger.info("PROCESSED %s" % taskData)
83      taskData['bouncesLeft'] -= 1
84      return cls.executeTask(taskid, taskData)
85
86    @classmethod
87    def exec_taskError(cls, taskid, taskData, errorMsg):
88      print "OOOOOO THERE WAS AN ERROR!!", errorMsg
89      return S_OK()
90
91    @classmethod
92    def exec_taskFreeze(cls, taskid, taskData, eType):
93      """
94      A task can be frozen either because there are no executors connected that can␣
    ↪handle it
95       or becase an executor has requested it.
96      """
97      print "OOOOOO THERE WAS A TASK FROZEN"
98      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** wil 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 tast for **at least** that amount of time. It can keep it for more time if there aren't free *Executors* to handle it.

3.7.10 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:

```
DIRAC/WorkloadManagementSystem/scripts/dirac-wms-job-submit.py
```

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:

```
#!/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

```
#!/usr/bin/env python

"""  Ping a list of services and show the result
"""

__RCSID__ = "$Id$"

import sys
from DIRAC import exit as DIRACExit
from DIRAC import S_OK, S_ERROR
from DIRAC.Core.Base import Script
```

(continues on next page)

(continued from previous page)

```
# Define a simple class to hold the script parameters


class Params(object):

  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()

# Register accepted switches and their callbacks
Script.registerSwitch("r", "showRaw", "show raw result from the query",␣
↪cliParams.setRawResult)
Script.registerSwitch("p:", "numPings=", "Number of pings to do (by default␣
↪1)", cliParams.setNumOfPingsToDo)

# Define a help message
Script.setUsageMessage('\n'.join([__doc__,
                                  'Usage:',
                                  '  %s [option|cfgfile] <system name to␣
↪ping>+' % Script.scriptName,
                                  '  Specifying a system is mandatory']))

# Parse the command line and initialize DIRAC
Script.parseCommandLine(ignoreErrors=False)

# Get the list of services
servicesList = Script.getPositionalArgs()

# Check and process the command line switches and options
if not servicesList:
  Script.showHelp()
  DIRACExit(1)
```

Let's follow the example step by step. First, we import the required modules from DIRAC. *S_OK* and *S_ERROR* are the default way DIRAC modules return values or errors. The *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 *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 *registerSwtch()* method of the 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:

- *-h* and *–help* show the script help
- *-d* and *–debug* enables debug level for the script. Note that the forms *-dd* and *-ddd* are accepted resulting in increasingly higher verbosity level
- *-s* and *–section* changes the default section in the configuration for the script
- *-o* and *–option* set the value of an option in the configuration
- *-c* and *–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 *getPositionalArguments()* function.

After defining the switches, the *parseCommandLine()* function has to be called. This method not only parses the command line options but also initializes DIRAC collecting all the configuration data. **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:

```
#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:

```
#!/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> ...]
"""

__RCSID__ = '$Id$'

from DIRAC import S_OK, S_ERROR, gLogger, exit as DIRACExit
from DIRAC.Core.Base import Script

cliParams = None
switchDict = None


class Params(object):
  '''
    Class holding the parameters raw and pingsToDo, and callbacks for their
    respective switches.
  '''

  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 = [
```

(continues on next page)

(continued from previous page)

```
      ('', '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():
  '''
    This is the script main method, which will hold all the logic.
  '''

  # 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()
```

(continues on next page)

(continued from previous page)

```
  # Import the required DIRAC modules
  from DIRAC.Interfaces.API.Dirac import Dirac

  # Run the script
  main()

  # Bye
  DIRACExit(0)
```

3.7.11 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.

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:

```
from DIRAC.ConfigurationSystem.Client.Helpers.Operations import Operations

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:

```
from DIRAC.ConfigurationSystem.Client.Helpers.Operations import Operations

ops = Operations( group = 'dirac_user' )
```

Helper for accessing /Resources

Utilities for parallel programming

Thread Pool

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:

```
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:

```
threadPool.generateJobAndQueueIt( <functionToExecute>,
                                  args = ( arg1, arg2, ... ),
                                  oCallback = <resultCallbackFunction> )
```

or:

```
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:

```
pool = ProcessPool( minSize, maxSize, maxQueuedRequests, strictLimits=True,␣
↪poolCallback=None, poolExceptionCallback=None  )
```

where parameters are:

param int minSize at least <minSize> workers will be alive all the time

param int maxSize no more than <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:

```
pool.createAndQueueTask( funcDef,
                         args = ( arg1, arg2, ... ),
                         kwargs = { "kwarg1" : value1, "kwarg2" : value2 },
                         taskID = taskID,
                         callback = callbackDef,
                         exceptionCallback = exceptionCallBackDef,
                         usePoolCallbacks = False,
                         timeOut = 0,
                         blocking = True )
```

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:

```
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:

```
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:

```
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:

  ```
  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*.

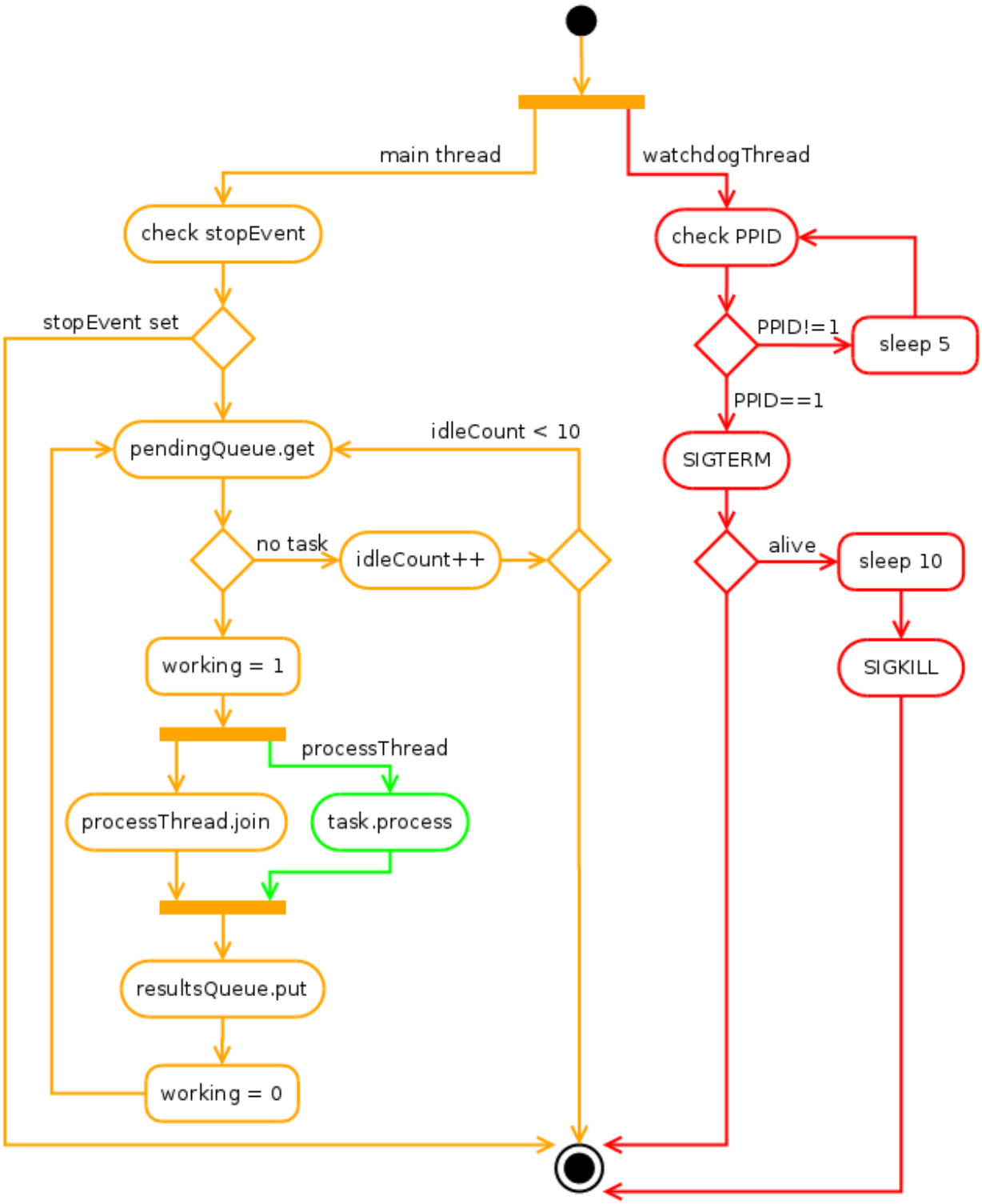
To use this procedure one has to execute:

```
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 spawing a watchdog thread checking on every 5 seconds PPID of worker. If parent process executing *ProcessPool* instance is dead for some reason (an 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 commiting 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 timout 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 basicaly a dictionary with the *'OK'* key to *False*, and a key *'Message'* which contains the actual error message.

```
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

```
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 = callAFunction()

  if not res['OK']:
    if 'No such file' in res['Message']:
      # Handle the error properly
      # Unfortunately not for func2, eventhough 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.

```
def doSomething(listOfItems):
  successful = {}
  failed = {}
```

(continues on next page)

(continued from previous page)

```
  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'*.

```
from DIRAC.Core.Utilities import DErrno
import errno

res = func1()
if not res['OK']:
  # This works whether res is an S_ERROR or a DError object
  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

```
# 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:

```
def func1():
  [...]
  return S_ERROR("File does not exist")

def main():
  res = func1()
  if not res['OK']:
    if res['Message'] == "File does not exist":
      # Handle the error properly
```

You happen to modify *func1* and decide to return the appropriate DError object, but do not change the *main* function:

```
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 *compatErrorString* which would state that "File does not exist" is *compatible* with errno.ENOENT.

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

- extra_dErrName: keys are the error name, values the number of it
- extra_dErrorCode: same as dErrorCode. 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)
- extra_dStrError: same as dStrError, Keys are the error code, values the error description
- extra_compatErrorString: same as compatErrorString. The compatible error strings are added to the existing one, and not replacing them.

Example of extension file :

```
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 adds yet another compatible error message
                                                                                ␣
↪         # for an error defined in the DIRAC DErrno
```

DIRAC gLogger

gLogger

gLogger is the logging solution within DIRAC. Based on the python *logging* library, it represents an interface to create and send informational, warn or error messages from the middleware to different outputs. In this documentation, we will focus on the functionalities proposed by *gLogger*.

Basics

Get a child *Logging* object

Logging presentation

gLogger is an instance of a *Logging* object. The purpose of these objects is to create log records. Moreover, they are part of a tree, which means that each *Logging* has a parent and can have a list of children. *gLogger* is considered as the root *Logging*, on the top of this tree.

Initialize a child *Logging*

Since *Logging* objects are part of a tree, it is possible to get children from each *Logging* object. For a simple use, we will simply get one child *Logging* from *gLogger*, the root *Logging*, via the command:

```
logger = gLogger.getSubLogger("logger")
```

This child can be used like *gLogger* in the middleware. In this way, we recommend you to avoid to use directly *gLogger* and to create at least one child from it for each component in *DIRAC* with a correct name.

Otherwise, note that the created child is identified by its name, *logger* in our case, and can be retrieve via the *getSubLogger()* method. For instance :

```
logger = gLogger.getSubLogger("logger")
newLogger = gLogger.getSubLogger("logger")
# Here, logger and newlogger are a same and unique object
```

Get its sub name

We can obtain the name of a child *Logging* via the *getSubName* method. Here is an example of use:

```
logger = gLogger.getSubLogger("logger")
logger.getSubName()
# > logger
```

Get its system and component names

Each *Logging* object belongs to one component from one system, the one which is running. Thus, we can get these names thanks to the *getName* method. They will appear as a *system/component* path like this:

```
logger = gLogger.getSubLogger("logger")
logger.getName()
# > Framework/Atom
```

Send a log record

Log record presentation

A log record is composed by a date, a system and a component name, a *Logging* name, a level and a message. This information represents its identity.

```
[Date] UTC [System]/[Component]/[Log] [Level]: [Message]
2017-04-25 15:51:01 UTC Framework/Atom/log ALWAYS: message
```

Levels and context of use

The level of a log record represents a major characteristic in its identity. Indeed, it constitutes its nature and defines if it will be displayed or not. *gLogger* puts 10 different levels at our disposal in DIRAC and here is a table describing them and their context of use.

Level name	Context of use
Fatal	Must be used before an error forcing the program exit and only in this case.
Always	Used with moderation, only for message that must appears all the time.
Error	Used when an error occur but do not need to force the program exit.
Exception	Actually a specification of the Error level which must be used when an exception is trapped.
Notice	Used to provide an important information.
Warn	Used when a potentially undesired behaviour can occur.
Info	Used to provide information.
Verbose	Used to provide extra information.
Debug	Must be used with moderation to debug the program.

These levels have a priority order from *debug* to *fatal*. In this way, *fatal* and *always* log records appear almost all the time whereas *debug* log records rarely appears. Actually, their appearance depends on the level of the *Logging* object which sends the log records.

Log record creation

10 methods are at our disposal to create log records from a *Logging* object. These methods carry the name of the different levels and they are all the same signature. They take a message which has to be fixed and a variable message in parameters and return a boolean value indicating if the log will appear or not. Here is an example of the *error* method to create error log records:

```
boolean error(sMsg, sVarMsg='')
```

For instance, we create *notice* log records via the following commands:

```
logger = gLogger.getSubLogger("logger")
logger.notice("message")
# > 2017-04-25 15:51:01 UTC Framework/logger NOTICE: message
logger.notice("mes", "sage")
# > 2017-04-25 15:51:01 UTC Framework/logger NOTICE: mes sage
```

Another interesting point is the use of the *exception* method which gives a stack trace with the message. Here is a use of the *exception* method:

```
logger = gLogger.getSubLogger("logger")
try:
    badIdea = 1/0
    print badIdea
except:
    logger.exception("bad idea")
# > 2017-04-25 15:51:01 UTC Framework/logger ERROR: message
#Traceback (most recent call last):
#File "....py", line 32, in <module>
#a = 1/0
#ZeroDivisionError: integer division or modulo by zero
```

Log records with variable data

gLogger use the old *%-style* to include variable data. Thus, you can include variable data like this:

```
logger = gLogger.getSubLogger("logger")
arg = "argument"
logger.notice("message with %s" % arg)
#> 2017-04-25 15:51:01 UTC Framework/logger NOTICE: message with argument
```

Control the *Logging* level

Logging level presentation

As we said before, each *Logging* has a level which is set at *notice* by default. According to this level, the log records are displayed or not. To be displayed, the level of the log record has to be equal or higher than the *Logging* level. Here is an example:

```
# logger level: NOTICE
logger = gLogger.getSubLogger("logger")
logger.error("appears")
logger.notice("appears")
logger.verbose("not appears")
# > 2017-04-25 15:51:01 UTC Framework/logger ERROR: appears
# > 2017-04-25 15:51:01 UTC Framework/logger NOTICE: appears
```

As we can see, the *verbose* log record is not displayed because its level is inferior to *notice*. Moreover, we will see in the advanced part that the level is propagate to the *Logging* children. Thus, for a basic use, you do not need to set the level of a child *Logging*.

Set a level via the command line

The more used and recommended method to set the level of *gLogger* is to use the command line arguments. It works with any *DIRAC* component but we can not define a specific level. Here is a table of these different arguments:

Argument	Level associated to the root *Logging*
default	notice
-d	verbose
-dd	verbose
-ddd	debug

We can find a complete table containing all the effects of the command line arguments in the *Summary of the command line argument configuration* part.

Set a level via the configuration

We can also set the *gLogger* level in the configuration via the *LogLevel* line. We can define a specific level with this method, but it does not work for scripts. Here is an example of an agent with the root *Logging*level set to *always*:

```
Agents
{
  SimplestAgent
  {
    LogLevel = ALWAYS
    ...
  }
}
```

Set a level via the *setLevel* method

Here is a last method to set any *Logging* level. We just have to give it a string representing a level like this:

```
logger = gLogger.getSubLogger("logger")
logger.setLevel("info")
```

In this example, the level of *logger* is set to *info*. By the way, we recommend you to not use this method for a basic use.

Get the level attaching to a specific *Logging*

We can obviously get a level associate to a *Logging* via the *getLevel* method. This method returns a string representing a level. Here is an example of use:

```
logger = gLogger.getSubLogger("logger")
logger.getLevel()
# > "NOTICE"
```

Get all the existing levels

In the same way, we can get all the existing level names thanks to the *getAllPossibleLevels* method. This method returns a list of string representing the different levels. Here is an example of use:

```
# 'level' comes from a user
def method(level):
    if level in self.logger.getAllPossibleLevels():
     # ...
```

Test the *Logging* level superiority

In some cases, it can be interesting to test the *Logging* level before creating a log record. For instance, we need to send a *verbose* log record using an expensive function and we do not need to make it if it can not be send to an output.

To avoid such an operation, we can use the *shown* method which controls if the *Logging* level is superior to a specific level. If it is the case, the method returns *True*, else returns *False*. Here is an example of this use:

```
# logger level: ERROR
logger = gLogger.getSubLogger("logger")
if logger.shown('verbose'):
    logger.verbose("Expensive message: %s" % expensiveFunc())
# > False
```

Modify the log record display

Default display

As we saw before, the basic display for a log record is:

```
[Date] UTC [System]/[Component]/[Log] [Level]: [Message]
2017-04-25 15:51:01 UTC Framework/Atom/log ALWAYS: message
```

The date is UTC formatted and the system and the component names come from the configuration. By default, the system name is *Framework* while the component name does not exist. This display can vary according to different option parameters.

Remove the prefix of the log record

In the scripts, we can observe log record without any prefix, only a message like this:

```
[Message]
message
```

This behaviour is explained by the *parseCommandLine* function, that we can find in every scripts, which set the boolean *headerIsShown* from *Logging* to *False*. To do a such operation, it used the *showHeaders* method from *Logging*. Here is the signature of the method:

```
showHeaders(yesno=True)
```

To summarize, the default value of *headerIsShown* is *True*, which means that the prefix is displayed, and we can set it at False to hide it.

There are two ways to modify it, the *showHeaders* method as we saw, and the command line argument *-d*. Here is a table presenting the changes according to the argument value:

Argument	Level associated to the root *Logging*
Default(Executors/Agents/Services)	True
Default(Scripts)	False
-d	default value
-dd	True
-ddd	True

We can find a complete table containing all the effects of the command line arguments in the *Summary of the command line argument configuration* part.

Add the thread ID in the log record

It is possible to add a thread ID in our log records thanks to the *showThreadIDs* method which modify the boolean *threadIDIsShown* value. As the *showHeaders* method, it takes a boolean in parameter to set *threadIDIsShown*. This attribute is set at *False* by default. Here is an example with the boolean at *True*:

```
[Date] UTC [System]/[Component]/[Log][Thread] [Level]: [Message]
2017-04-25 15:51:01 UTC Framework/Atom/log[140218144]ALWAYS: message
```

We can see the thread ID between the *Logging* name and the level: [140218144]. Nevertheless, set the boolean value is not the only requirement. Indeed, *headerIsShown* must be set at *True* to effect the change. In this way, it is impossible to have the thread ID without the prefix.

A second way to set the boolean is to use the command line argument *-d*. Here is a table presenting the changes according to the argument:

Argument	Level associated to the root *Logging*
Default(Executors/Agents/Services)	False
Default(Scripts)	False
-d	default value
-dd	default value
-ddd	True

We can find a complete table containing all the effects of the command line arguments in the *Summary of the command line argument configuration* part.

Remove colors on the log records

LogColor option is only available from the configuration, and only for the *stdout* and the *stderr* with agents, services and executors. By default, the *LogColor* option is set a *True* and adds colors on the log records according to their levels. You can remove colors setting the option at *False* in the configuration:

```
LogColor = False
```

We can find a configuration example containing different options in the *Configuration example* part.

Get the option values

It is possible to obtain the names and the values associated of all these options with the *getDisplayOptions* method. This method returns the dictionary used by the *Logging* object itself and not a copy, so we have to be careful with its use. Here is an example:

```
logger = gLogger.getSubLogger("logger")
logger.getDisplayOptions()
# > {'Color': False, 'Path': False,
#    'headerIsShown': True, 'threadIsShown': False}
```

Send a log record in different outputs

Backend presentation

Backend objects are used to receive the log record created before, format it according to the choice of the client, and send it in the right output. We can find an exhaustive list of the existing *Backend* types in the *Backends* part.

Backend resources

A *Backend resource* is the representation of a *Backend* object in the configuration. It is represented by one or two elements depending on its nature. The first is an identifier, which can be a default identifier or a custom:

- Default identifiers take the name of a *Backend* class name, <*backendID*> will refer to the <*BackendID*>*Backend* class, *stdout* and *StdoutBackend* for instance.
- Custom identifiers can take any name like *f015* or *Jwr8*, there is no construction rule.

The second element is a set of parameters according to the *Backend* represented. Custom identifiers absolutely need to complete the *Plugin* option to indicate which *Backend* type they represent using a default identifier. This section can also be empty if the *Backend* do not need parameter and if the identifier is a default identifier. Here is a generic example of a *Backend resource*:

```
<backendDefaultID1>
{
    <param1> = <value1>
    <param2> = <value2>
}

<backendCustomID>
{
    Plugin = <backendDefaultID2>
    <param1> = <value1>
}
```

Declare the *Backend* resources

Before using them, *Backend resources* have to be declared in the configuration. They can be configured in a global way or in a local way. To declare them in the global way, we must put them in the */Resources/LogBackends* section of the configuration, like this:

```
Resources
{
    LogBackends
    {
        <backendID1>
        {
            Plugin = <backendClass1>
            <param1> = <value1>
        }
        <backendID2>
        {
            Plugin = <bakendClass2>
            <param2> = <value2>
        }
        <backendID3>
        {
            <param3> = <value3>
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

Here is an example of a concrete configuration:

```
Resources
{
    LogBackends
    {
        f01
        {
            Plugin = file
            FileName = /path/to/file.log
        }
        es2
        {
            Plugin = elasticSearch
            Host = lhcb
            Port = 9540
        }
        file
        {
            FileName = /path/to/anotherfile.log
        }
    }
}
```

In this case, we have 3 *Backend* identifiers, namely *f01* and *es2* which are custom identifiers respectively related on *FileBackend* and *ElasticSearchBackend*, and *file* which is a default identifier based on *FileBackend*.

This configuration allows a *Backend resource* use in any component of the configuration, but we can also create some specific *Backend resources* inside a local component. To create local resources, you have to follow the same process in a *LogBackendsConfig* section like this:

```
<Agent>
{
    ...
    LogBackendsConfig
    {
        <backendID4>
        {
            Plugin = <backendClass4>
            <param4> = <value4>
        }
        <backendID5>
        {
            Plugin = <bakendClass5>
            <param5> = <value5>
        }
        <backendID6>
        {
            <param6> = <value6>
        }
    }
}
```

Moreover, a same *Backend* identifier can be declared in the both sections in order to update it. Indeed, such a declaration triggers a parameters merger. In case of parameters conflicts, the local parameters are always choosen. Here is an example:

```
<Systems>
{
    Agents
    {
        <Agent1>
        {
            ...
            LogBackendsConfig
            {
                <backendID1>
                {
                    <param1> = <value1>
                    <param2> = <value2>
                }
            }
        }
    }
}
Resources
{
    LogBackends
    {
        <backendID1>
        {
            Plugin = <backendClass1>
            <param1> = <value4>
            <param3> = <value3>
        }
    }
}
```

In this case, *gLogger* in <*Agent1*> will have one *Backend* instance of the <*backendClass1*>*Backend* class which will have 3 parameters:

- <param1> = <value1>
- <param2> = <value2>
- <param3> = <value3>

Use the *Backend* resources

Once our *Backend* resources are declared, we have to specify where we want to use them and we have many possibilities. First of all, we can add them for the totality of the components. This can be made in the */Operations/defaults/* section. Here is the way to proceed:

```
Operations
{
    Defaults
    {
        Logging
        {
            DefaultBackends = <backendID1>, <backendID2>, <backendID3>
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

We can also add them for a specific component type, the agents or the services for instance. Such a declaration will overwrite the previous one for the component type choosen:

```
Operations
{
    Defaults
    {
        Logging
        {
            Default<componentType>sBackends = <backendID1>, <backendID2>, <backendID3>
        }
    }
}
```

Do not forget the *s* between *<componentType>* and *Backends*. In this case, all the *<componentType>* components will have the same resources if we do not overwritten locally. This can be made by the use of the *LogBackends* option used inside any component like this:

```
<Agent1>
{
    LogBackends = <backend1>, <backend2>, <backend3>
}
```

If none of these options is specified, the *stdout Backend* will be used.

Some examples and summaries

Configuration example

Here is a configuration which contains *Logging* and *Backend* configuration:

```
Systems
{
    FrameworkSystem
    {
        Agents
        {
            SimplestAgent
            {
              LogLevel = INFO
              LogBackends = stdout,stderr,file, file2, es2
              LogBackendsConfig
              {
                file
                {
                    FileName = /tmp/logtmp.log
                }
                file2
                {
                    Plugin = file
```

(continues on next page)

(continued from previous page)

```
                    FileName = /tmp/logtmp2.log
                }
              }
              LogColor = False
            }
            AnotherAgent
            {
                LogLevel = NOTICE
                LogBackends = stdout, es2
                LogBackendsConfig
                {
                    es2
                    {
                        UserName = lchb34
                        Password = passw0rd
                    }
                }
            }
        }
    }
}
Operations
{
    Defaults
    {
        Logging
        {
            DefaultBackends = stdout
            DefaultAgentsBackends = stderr
        }
    }
}
Resources
{
    LogBackends
    {
        es2
        {
            Plugin = elasticSearch
            Host = lhcb
            Port = 9540
            UserName = lhcb
            Password = 123456
        }
    }
}
```

To summarize, this file configures two agents respectively named *SimplestAgent* and *AnotherAgent*. In *SimplestAgent*, it sets the level of *gLogger* at *info*, adds 5 *Backend* objects to it, which are *stdout*, *stderr*, two *file Backend* objects and an *ElastiSearch* access. Thus, each log record superior to *info* level, created by a *Logging* object in the agent, will be sent to 5 different outputs: *stdout*, *stderr*, */tmp/logtmp.log*, */tmp/logtmp2.log* and ElasticSearch. In *AnotherAgent*, the same process is performed, and each log record superior to *notice* level is sent to *stdout* and another ElasticSearch database because of the redifinition. None of the default *Backend* objects of the *Operations* section are used because of the overwriting. In addition, the log records will be not displayed with color.

Summary of the command line argument configuration

Here is a complete table explaining the changes provided by the command line argument *-d*:

Argument	ShowHeader	showThread	Level
Default(Executors/Agents/Services)	True	False	Notice
Default(Scripts)	False	False	Notice
-d	DefaultValue	DefaultValue	Verbose
-dd	True	DefaultValue	Verbose
-ddd	True	True	Debug

About multiple processes and threads

Multiple processes

gLogger object is naturally different for two distinct processes and can not save the application from process conflicts. Indeed, *gLogger* is not process-safe, that means that two processes can encounter conflicts if they try to write on a same file at the same time. So, be careful to avoid the case.

Multiple threads

gLogger is completely thread-safe, there is no conflict possible especially in the case when two threads try to write on a same file at the same time.

About the use of external libraries

DIRAC uses some external libraries which have their own loggers, mainly based on the standard logging Python library like *gLogger*. Logs providing by these libraries can be useful in debugging, but not in production. The *enableLogsFromExternalLib* and *disableLogsFromExternalLib* methods allow us to enable or disable the display of these logs. The first method initializes a specific logger for external libraries like this:

- a level at Debug
- a display on the standard error output
- a log format close to the one used in *DIRAC*

We can call these methods each time that we use an external library and we want to see the logs inside or not.

Filter

The output given by the different logger can be further controlled through the use of *filters*. Any configured backend can be given the paramter *Filter*, which takes a comma separated list of filterIDs.

```
Resources
{
    LogBackends
    {
        <backendID1>
        {
            Plugin = <backendClass1>
```

(continues on next page)

(continued from previous page)

```
            Filter = MyFilter[,MyOtherFilter]*
            <param1> = <value4>
            <param3> = <value3>
        }
    }
}
```

Each filter can be configured with a given plugin type and the parameters used for the given plugin. See the documentation for the `LogFilters` for the available plugins and their parameters.

Each filter is queried, and only the the log record passes *all* filters is passed onwards.

```
Resources
{
    LogFilters
    {
        MyFilter
        {
            Plugin = FilterPlugin
            Parameter = Value, Value2
        }
    }
}
```

Filter implementation

The filter implementations need to be located in the *Resources/LogFilters* folder and can be any class that implements a `filter` function that takes a log record as an argument. See the existing implementations in `LogFilters` as examples.

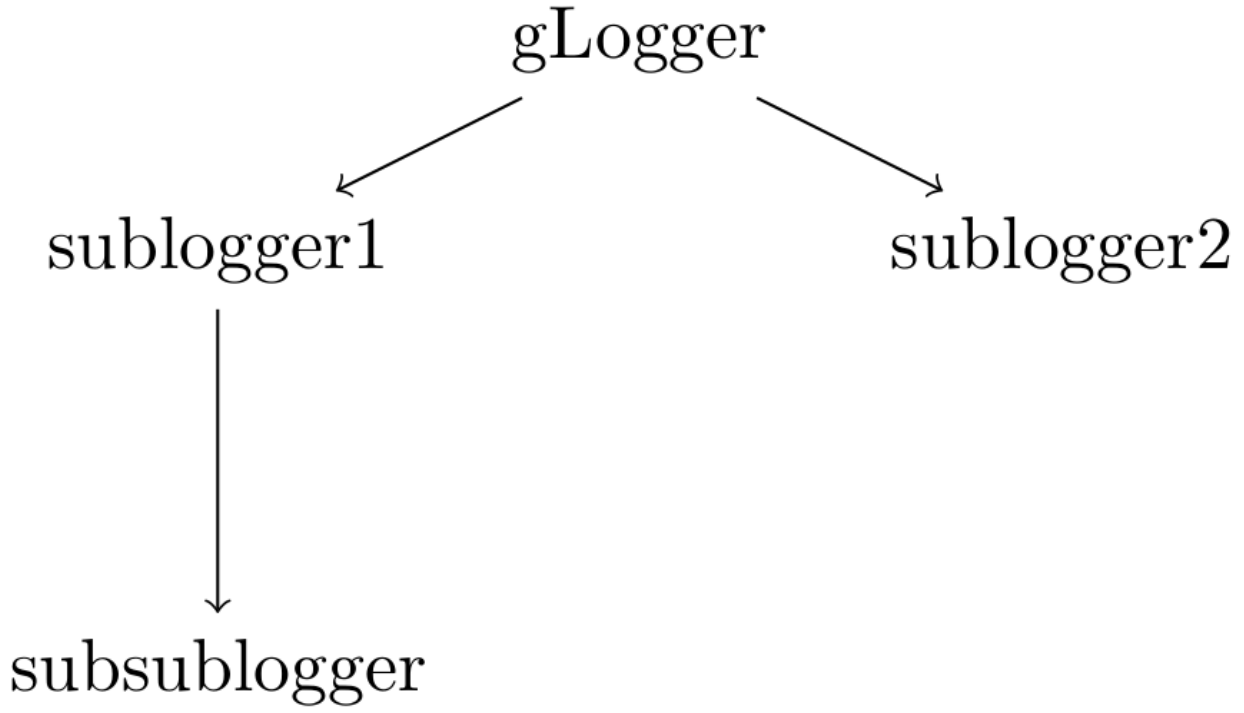
Advanced part

You can find more information about *gLogger* and its functionalities in the *Advanced use* part.

Advanced use

Get a children tree

As we said in the *Basics* part, all *Logging* objects can own a list of children and a parent, and is part of a *Logging* tree like this:



Here is a snippet presenting the creation of the tree seen above:

```
# level 1
logger = gLogger.getSubLogger("logger")
# level 2
sublogger1 = logger.getSubLogger("sublogger1")
sublogger2 = logger.getSubLogger("sublogger2")
# level 3
subsublogger = sublogger1.getSubLogger("subsublogger")
```

Set a child level

The truth about the levels

In the basic part, we talked about the different ways to set a *Logging* level. Only the *gLogger* level was allowed to be set.

This is because, in truth, *Logging* objects have two different levels: their own level, set to *debug* and unchangeable, and the level of its *Backend* objects. Thus, when we want to change the *Logging* level, we change the *Backend* objects level of this *Logging* in reality.

In this way, every log records of every levels are created by every *Logging* objects and can be send to a central logging server. The other *Backend* objects can sort the log records according to the level choosen by the user to send them or not to the output.

The level propagation

As every *Logging* object is part of a tree, the level of a parent can be propagated to its children. Thus, we do not have to set all the children levels:

```
# gLogger level: NOTICE
logger = gLogger.getSubLogger("logger")
print logger.getLevel()
# > NOTICE
```

While the children levels are not define by the user, they are modified according to the parent level:

```
logger = gLogger.getSubLogger("logger")
sublogger = logger.getSubLogger("sublogger")
print logger.getLevel()
print sublogger.getLevel()
# > NOTICE
# > NOTICE
logger.setLevel("error")
print logger.getLevel()
print sublogger.getLevel()
# > ERROR
# > ERROR
```

The only way to stop the propagation is to use the *setLevel* method on a *Logging*. For instance, in the previous example, *logger* has now its own level, and it can not be changed by its parent:

```
logger = gLogger.getSubLogger("logger")
print logger.getLevel()
# > NOTICE
logger.setLevel("error")
print logger.getLevel()
# > ERROR
gLogger.setLevel("debug")
print logger.getLevel()
# > ERROR
```

Nevertheless, the propagation is still existing for the children of *logger*:

```
logger = gLogger.getSubLogger("logger")
sublogger = logger.getSubLogger("sublogger")
print logger.getLevel()
print sublogger.getLevel()
# > NOTICE
# > NOTICE
logger.setLevel("error")
print logger.getLevel()
print sublogger.getLevel()
# > ERROR
# > ERROR
gLogger.setLevel("debug")
print gLogger.getLevel()
print logger.getLevel()
print sublogger.getLevel()
# > DEBUG
# > ERROR
# > ERROR
```

(continues on next page)

(continued from previous page)

```
logger.setLevel("verbose")
print gLogger.getLevel()
print logger.getLevel()
print sublogger.getLevel()
# > DEBUG
# > VERBOSE
# > VERBOSE
```

To summarize, a *Logging* receives its parent level until the user sets its level with the *setLevel* method.

The *setLevel* utility

As we said before, the *setLevel* method modifies the *Backend* objects level of the current *Logging* so if this last mentionned have no *Backend* objects, set its level become useless.

Furthermore, the *setLevel* method is useful only if we add it some *Backend* objects.

Add a *Backend* object on a child *Logging*

registerBackend(s) presentation

Now, it is possible to add some *Backend* objects to any *Logging* via the *registerBackend* method. This method takes two parameters, a name of a *Backend* objects, and a dictionary of attribute names and their values associated. Here is an example of use:

```
logger = gLogger.getSubLogger("logger")
logger.registerBackend('stdout')
logger.registerBackend('file', {'FileName': 'file.log'})
# An alternative:
# logger.registerBackends(['stdout', 'file'], {'FileName': 'file.log'})
```

This, will create *stdout* and *file Backend* objects in *logger*. The alternative method named *registerBackends* takes a *Backend* objects list as first argument. This method can be really efficient to add some *Backend* objects in one time but also restrictive due to the unicity of the dictionary keys.

Log records propagation

Obviously, each log record created by a child *Logging* goes up in its parent if the true *Logging* level allowed it, but as it is always at *debug*, it goes up anyway. The log record goes up until *gLogger* and it is displayed in all the *Backend* objects encounter in the parents if the level allowed it.

In this way, *gLogger* display every log records of every *Logging* object, even if you add *Backend* objects in a child, the log record will appears multiple times in this case. Here is an example:

```
# gLogger has a stdout Backend
logger = gLogger.getSubLogger("logger")
logger.registerBackend('stdout')
logger.verbose("message")
# > 2017-04-25 15:51:01 UTC Framework/Atom/logger VERBOSE: message
# > 2017-04-25 15:51:01 UTC Framework/Atom/logger VERBOSE: message
gLogger.info("message")
# > 2017-04-25 15:51:01 UTC Framework/Atom/logger INFO: message
```

We can also notice that the log records do not go down in the tree.

The truth about the returned value of the level methods

The boolean contained in the level methods seen in the *Basics* part indicates, in reality, if the log record will appear or not in the *Backend* objects of the current *Logging*. Thus, the boolean can be at *False* and the log record can appear in one of its parent anyway.

The *registerBackend(s)* utility

This functionality gives the possibility to isolate some log records from a specific *Logging* or isolate log records above a specific level. For example, we want only, at minimum, *error* log records providing by a specific child named *logger* in a file named *file.log*. Here is a snippet of this example:

```
# gLogger: stdout Backend, NOTICE level
logger = gLogger.getSubLogger("logger")
logger.registerBackend('file', {'FileName': 'file.log'})
logger.setLevel("error")
logger.verbose("appears only in stdout")
logger.notice("appears only in stdout")
logger.error("appears in stdout and in file.log")
# in stdout:
# > ... UTC Framework/Atom/logger VERBOSE: appears only in stdout
# > ... UTC Framework/Atom/logger NOTICE: appears only in stdout
# > ... UTC Framework/Atom/logger ERROR: appears in stdout, in file.log
# in file.log:
# > ... UTC Framework/Atom/logger ERROR: appears in stdout, in file.log
```

Modify a display for different *Logging* objects

showThreadIDs and *showHeaders* propagation

Now that it is possible to add *Backend* objects to any *Logging*, we have also the possibility to modify their display formats. To do such an operation, we have to use the *showThreadIDs* and *showHeaders* methods in a child. Of course, this child must contain at least one *Backend* to be efficient.

Thus, these methods function exactly as the *setLevel* method, so they can be propagate in the children if the options are not modified by the user.

showThreadIDs and *showHeaders* utility

Here, the utility is to modify the display format of the isolate log records from a specific *Logging* to not be embarrassed with extra information that we do not want for example:

```
# gLogger: stdout Backend, NOTICE level, showHeaders at True
logger = gLogger.getSubLogger("logger")
logger.registerBackend('file', {'FileName': 'file.log'})
logger.setLevel("error")
logger.showHeaders(False)
logger.verbose("appears only in stdout")
logger.notice("appears only in stdout")
```

(continues on next page)

(continued from previous page)

```
logger.error("appears in stdout and in file.log")
# in stdout:
# > ... UTC Framework/Atom/logger VERBOSE: appears only in stdout
# > ... UTC Framework/Atom/logger NOTICE: appears only in stdout
# > ... UTC Framework/Atom/logger ERROR: appears in stdout, in file.log
# in file.log:
# > appears in stdout, in file.log
```

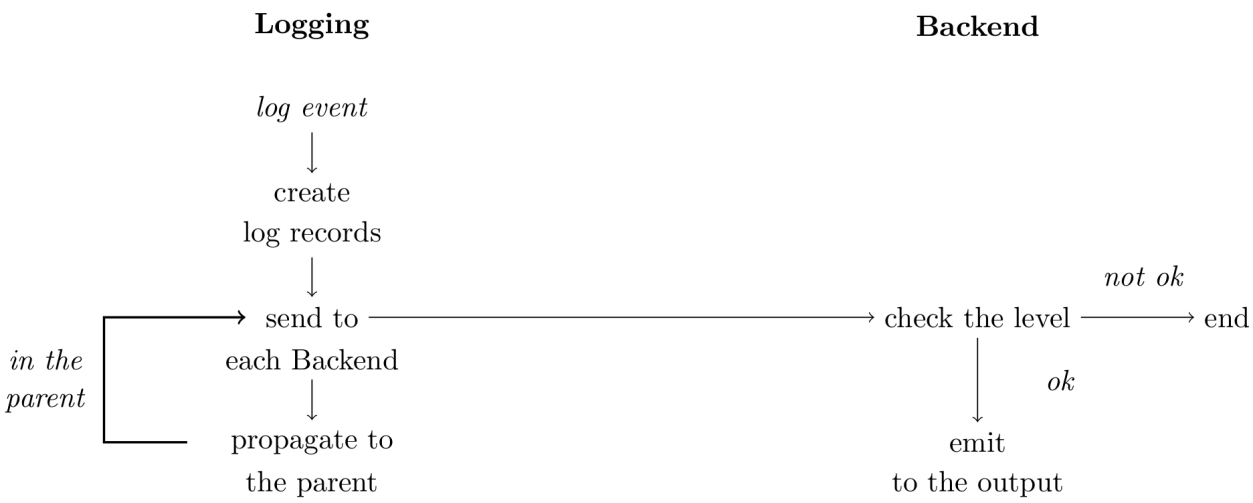
The *LogColor* case

This option can not be modified in the children of *gLogger*, even by *gLogger* itself after the configuration, so the children receive the *gLogger* configuration.

Some examples and summaries

Summary diagram

Here is a diagram showing the complete path of a log record from its creation to its emission in an output:



Backends

This section presents all the existing *Backend* classes that you can use in your program, followed by their parameters.

StdoutBackend

Description

Used to emit log records to the standard output.

Parameters

No parameter

StderrBackend

Description

Used to emit log records to the standard error output.

Parameters

No parameter

FileBackend

Description

Used to emit log records in a specific file.

Parameters

Option	Description	Default value
FileName	name of the file where the log records must be sent	Dirac-log_[pid].log

ServerBackend

Description

Used to emit log records in the *SystemLogging* service of *DIRAC* in order to store them in the *SystemLoggingDB* database. This *Backend* only allows log records superior or equal to *Error* to be sent to the service.

Parameters

Option	Description	Default value
SleepTime	sleep time in seconds	150

ElasticSearchBackend

Description

Used to emit log records in the an ElasticSearch database. The *Backend* acccepts logs from *Debug* to *Always* level.

Parameters

Option	Description	Default value
Host	host machine where the ElasticSearch DB is installed	''
Port	port where the ElasticSearch DB listen	9203
User	username of the ElasticSearch DB (optional)	None
Password	password of the ElasticSearch DB (optional)	None
Index	ElasticSearch index	''
BufferSize	maximum size of the buffer before sending	1000
FlushTime	maximum waiting time in seconds before sending	1

MessageQueueBackend

Description

Used to emit log records in a MessageQueue server using Stomp protocol. The *Backend* acccepts logs from *Debug* to *Always* level.

Parameters

Option	Description	Default value
MsgQueue	MessageQueueRessources from DIRAC	''

MsgQueue represents a MessageQueue resources from DIRAC under this form:

```
mardirac3.in2p3.fr::Queues::TestQueue
```

You will find more details about these resources in the *Message Queues* section.

Changes

Here is a list of the different changes due to the replacement of *gLogger*.

Vocabulary changes

Now, *Logger* objects are renamed *Logging*. In the same way, a sub *Logger* becomes a child *Logging*. To finish, a message becomes a log record.

Logger creation

child attribute in the *getSubLogger* method

There is no possibility to remove the system and the component names from the log record anymore. In this way, the *child* attribute becomes totally useless and should not be used. Here is the only way to create a child *Logging* now:

```
gLogger.getSubLogger("logger")
```

Logging and child *Logging*

Before the update, when a sub *Logger* got a sub *Logger*, we had always the same display:

```
log = gLogger.getSubLogger("log")
sublog = log.getSubLogger("sublog")
log.always("message")
sublog.always("message")
# ... Framework/log ALWAYS: message
# ... Framework/log ALWAYS: message
```

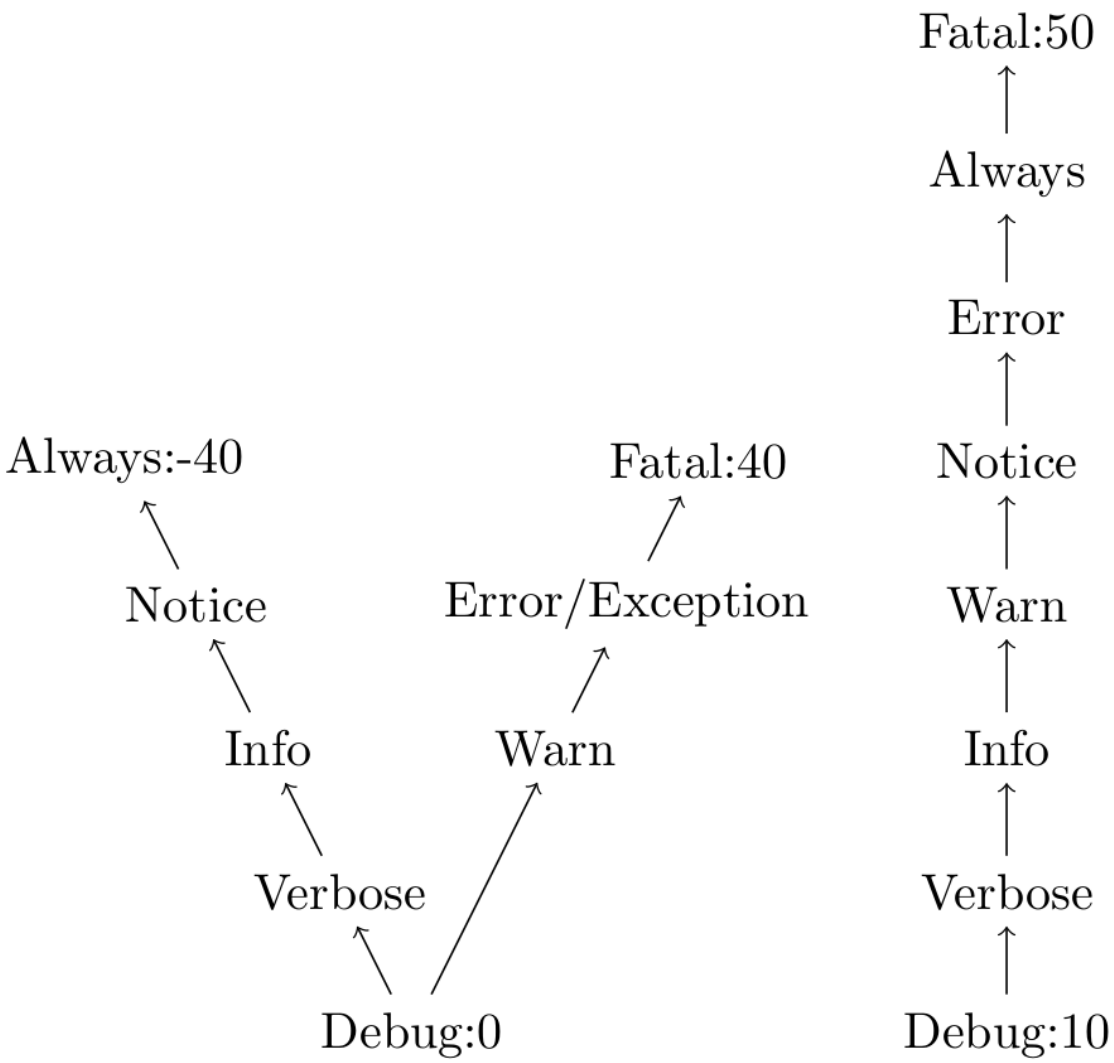
Now, the child *Logging* keeps this name in the display, and the one of all its parents:

```
log = gLogger.getSubLogger("log")
sublog = log.getSubLogger("sublog")
log.always("message")
sublog.always("message")
# ... Framework/log ALWAYS: message
# ... Framework/log/sublog ALWAYS: message
```

Levels

Level system

There are still 9 different levels in DIRAC, but the system changes. In fact, the old *gLogger* was composed by a *V* level model from *Always* to *Fatal*. Now, the level system becomes linear. Here is a figure presenting the old level system at the left, and the new at the right:



You can notice that the *exception* level disappears, but it still possible to create *exception* log records. They will appear as an *error* message with an additional stack trace.

setLevel() functionality

If the developer does not have set a level to his *Logging*, this one takes the level of its parent by default. In this way, each time the parent level is modified, the level of its children changes too. It is a propagation:

```
gLogger.setLevel('notice')
log = gLogger.getSubLogger('log')
sublog = log.getSubLogger('sublog')
gLogger.getLevel()
log.getLevel()
sublog.getLevel()
# > NOTICE
# > NOTICE
# > NOTICE
```

(continues on next page)

(continued from previous page)

```
gLogger.setLevel('error')
gLogger.getLevel()
log.getLevel()
sublog.getLevel()
# > ERROR
# > ERROR
# > ERROR
```

It is possible to limit this propagation setting the level of a *Logging* with the *setLevel()* method. Thus, even if the parent level change, the level of the *Logging* will stay the same. See this example based on the previous snippet:

```
# gLogger, log, sublog level: ERROR
log.setLevel('verbose')
gLogger.getLevel()
log.getLevel()
sublog.getLevel()
# > ERROR
# > VERBOSE
# > VERBOSE

gLogger.setLevel('debug')
gLogger.getLevel()
log.getLevel()
sublog.getLevel()
# > DEBUG
# > VERBOSE
# > VERBOSE
```

Message

lExcInfo and *lException* attributes

As the *child* attribute, these attributes are now useless and should not be used. Here is the only way to create an exception log record now:

```
gLogger.exception("message")
```

Display

Multiple line messages

The old *gLogger* allowed the developers to create log records on multiple lines with a prefix on each line:

```
2017-04-25 15:51:01 UTC Framework/log ALWAYS: this is a message
2017-04-25 15:51:01 UTC Framework/log ALWAYS: on multiple lines
```

Now, this functionality does not exist anymore. The prefix is only present on the first line:

```
2017-04-25 15:51:01 UTC Framework/log ALWAYS: this is a message
on multiple lines
```

Exception message display

There is also a minor change on the *exception* messages. At the top, there is the old exception display, at the bottom the new:

```
... EXCEPT: message
... EXCEPT: == EXCEPTION == ZeroDivisionError
... EXCEPT: File "....py", line 119, in ...
... EXCEPT: a = 1 / 0
... EXCEPT:
... EXCEPT: ZeroDivisionError: integer division or modulo by zero
... EXCEPT: ===============
```

```
... ERROR: message
Traceback (most recent call last):
File "....py", line 32, in <module>
a = 1/0
ZeroDivisionError: integer division or modulo by zero
```

registerBackends() and registerBackend() for all loggers

Now, each *Logging* can use the *registerBackend(s)* method for their own needs. In this way, you can easily isolate log records from a specific *Logging* object.

Nevertheless, all log records from a child *Logging* are sent to the parent *Logging* which displays these log records if it can and send these ones to its parent and so on. Thus, all log records from all *Logging* objects go to *gLogger* which displays every log messages:

```
# gLogger has no Backend, DEBUG level
gLogger.registerBackend('stdout')

log = gLogger.getSubLogger('log')
log.registerBackends(['stderr', 'stdout'])

sublog = log.getSubLogger('sublog')

subsublog = sublog.getSubLogger('sublog')
subsublog.registerBackend('file')

subsublog.verbose("message")
# file
# > ...VERBOSE: message
# stderr
# > ...VERBOSE: message
# stdout
# > ...VERBOSE: message
# > ...VERBOSE: message
```

As you can see, the subsublog message goes up in the chain and is displayed by all of its parents. You can also notice its double presence in *stdout*.

Local *showHeaders* and *showThreadIDs*

Before, the *showHeaders* and the *showThreadIDs* options were globals, and any logger could change their values and this could impact all the loggers. This is not the case anymore since these options are locals to the *Logging* objects. It

works exactly like the *setLevel()* method.

If the developer does not have set a format to his *Logging*, this one takes the format of its parent by default. In this way, each time the parent format is modified, the format of its children changes too. It is a propagation:

```
# gLogger has a stdout Backend, DEBUG level
gLogger.showHeaders(True)
log = gLogger.getSubLogger('log')
log.registerBackends(['stderr'])
log.verbose("message")
# stdout
# > ...VERBOSE: message
# stderr
# > ...VERBOSE: message

gLogger.showHeaders(False)
log.verbose("message")
# stdout
# > message
# stderr
# > message
```

It is possible to limit this propagation setting the format of a *Logging* with the *showHeaders()* or *showThreadIDs()* methods. Thus, even if the parent format changes, the format of the *Logging* object will stay the same:

```
# gLogger has a stdout Backend, DEBUG level
gLogger.showHeaders(True)
log = gLogger.getSubLogger('log')
log.registerBackends(['stderr'])
log.showHeaders(True)
log.verbose("message")
# stdout
# > VERBOSE: message
# stderr
# > VERBOSE: message

gLogger.showHeaders(False)
log.verbose("message")
# stdout
# > message
# stderr
# > VERBOSE: message
```

Backend configuration

Now, the *Backend* configuration in the configuration becomes more readable and can be centralized.

```
LogBackends = <backend1>, <backend2>, <backend3>
BackendOptions
{
    <param backend2> = <value1>
    <param backend3> = <value2>
}
```

This configuration becomes:

```
LogBackends = <backend1>, <backend2>, <backend3>
LogBackendsConfig
{
    <backend2>
    {
        <param backend2> = <value1>
    }
    <backend3>
    {
        <param backend3> = <value2>
    }
}
```

The first main advantage of this new feature is that you can define many *Backend* objects of a same type and provide them different specifications like this:

```
LogBackends = file, f01, log
LogBackendsConfig
{
    f01
    {
        Type = file
        FileName = log1.txt
    }
    log
    {
        Type = file
        FileName = log2.txt
    }
}
```

Here you have 3 *file Backend* objects which will send log records in 3 differents files. The only rule to this functionality is to precise the type of the *Backend* if it is non conventional.

The second main advantage is that you can centralize a configuration to have it either for some different components, or for all the components of a same type, or for all the components. Here is an example of a centralized configuration:

```
Operations
{
    Defaults
    {
        Logging
        {
            DefaultAgentsBackends = stdout, file
        }
    }
}
Systems
{
    ...
    Agents
    {
        SimplestAgent
        {
            ...
        }
        AnotherAgent
```

(continues on next page)

(continued from previous page)

```
        {
            ...
        }
    }
}
```

In this example, *SimplestAgent* and *AnotherAgent* which have no *Backend* configuration will inherit the *DefaultAgentsBackends* configuration: *stdout* and *file*.

Multiple processes and threads

Multiple threads

gLogger is now thread-safe. This means that you have the possibility to write safely in one file with two different threads.

gLogger Development

Here is the *gLogger* documentation for developer briefly presenting the different components of the logging system and their locations.

The logging system package

The source code is contained in the *FrameworkSystem/private/standardLogging* package. There, we can find the *Logging*, *LoggingRoot* and *LogLevels* classes and the *Handler* and *Formatter* packages. We can also find the *Backend* package that use the *Handler* and *Formatter* package in *Resources/LogBackends*.

Logging

Logging is a wrapper of the logger object from the standard *logging* library which integrates some *DIRAC* concepts. It is the equivalent to the *Logger* class in the old logging system.

It is used like an interface to use the logger object of the *logging* library. Its purpose is to replace transparently the old *gLogger* object in the existing code in order to minimize the changes.

LoggingRoot

LoggingRoot inherits from *Logging*. It is specific because it is the first parent of the chain. In this context, it has more possibilities because it is the one and the only that can initialize the root logger of the standard *logging* library and it can configure it with the configuration thanks to the *initialize* method.

LogLevels

LogLevels is used to integrate custom levels to the standard library. Actually, it contains a class dictionary attribute named *levelDict* containing all the level names and their associated integer values. Its purpose is to make string-integer level conversion.

LogBackend package

Backend objects are used to create an abstraction of the *Handler* and *Formatter* concepts from the standard library. It is an equivalent of the *Backend* concept of the old *gLogger*. All the existing *Backend* objects are located in this package which currently contains:

- 'stdout': StdoutBackend
- 'stderr': StderrBackend
- 'file': FileBackend
- 'server': ServerBackend

In order to create custom *Backend* objects, we just have to make a new class named *[Backendname]Backend* in this package inheriting from *AbstractBackend*. For instance, the class name of the *stdout Backend* is *StdoutBackend*.

Then, to use it, we just have to add its name in the configuration as usual.

Handler package

The *Handler* package contains all the custom *Handler* objects created for a *DIRAC* use. All these *Handlers* must inherit from the standard *logging.Handler*. The package currently contains:

- ServerHandler: used to send log records to the *SystemLogging DIRAC* service

In order to create custom *Handler* objects, we just have to write a new class in this package inheriting from one of the *Handler* of the *logging* library.

Formatter package

The *Formatter* package contains all the custom *Formatter* objects create for *DIRAC* use. All these *Formatter* must inherit from the standard *logging.Formatter*. The package currently contains:

- BaseFormatter: used only to add some attributes in the constructor
- ColorBaseFormatter: used to color the log records

In order to create custom *Formatter* objects, we just have to create a new class in this package inheriting from *BaseFormatter*.

gLogger use

To be used in all the existing code, this logging system has to be instantiated in the name of *gLogger*.

gLogger instantiation

LoggingRoot is instantiated in the name of *gLogger* in *FrameworkSystem/Client/Logger*.

gLogger import

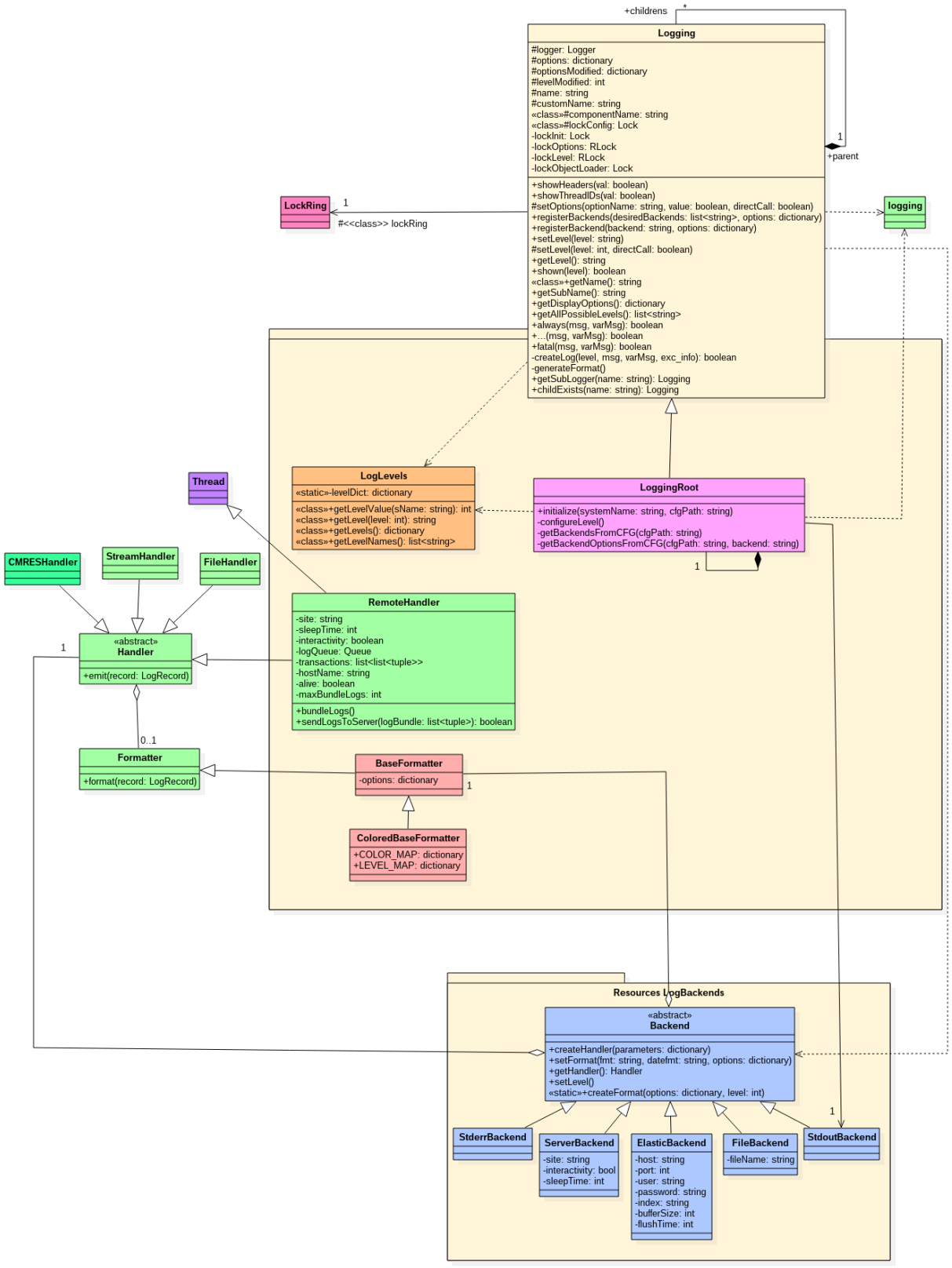
To call *gLogger* with the simple *from DIRAC import gLogger*, we have to put our variable in the *__init__* file of *DIRAC*.

gLogger and the *DIRAC* components

Once instantied, *gLogger* can be configured thanks to its *initialize* method. Some components like the services, agents and executors use it. You can retrieve the method usage in the *ConfigurationSystem/Client/LocalConfiguration* file in the *initLogger* method used by the *dirac-service*, *dirac-agent* and *dirac-executor* scripts.

The class diagram

Here is a class diagram presenting the system logging:


+childrens
*
Logging
#logger: Logger
#options: dictionary
#optionsModified: dictionary
#levelModified: int
#name: string
#customName: string
«class»#componentName: string
«class»#lockConfig: Lock
-lockInit: Lock
-lockOptions: RLock
-lockLevel: RLock
-lockObjectLoader: Lock
+showHeaders(val: boolean)
+showThreadIDs(val: boolean)
#setOptions(optionName: string, value: boolean, directCall: boolean)
+registerBackends(desiredBackends: list<string>, options: dictionary)
+registerBackend(backend: string, options: dictionary)
+setLevel(level: string)
#setLevel(level: int, directCall: boolean)
+getLevel(): string
+shown(level): boolean
«class»+getName(): string
+getSubName(): string
+getDisplayOptions(): dictionary
+getAllPossibleLevels(): list<string>
+always(msg, varMsg): boolean
+...(msg, varMsg): boolean
+fatal(msg, varMsg): boolean
-createLog(level, msg, varMsg, exc_info): boolean
-generateFormat()
+getSubLogger(name: string): Logging
+childExists(name: string): Logging
1
+parent
LockRing
1
#<<class>> lockRing
logging
LogLevels
«static»-levelDict: dictionary
«class»+getLevelValue(sName: string): int
«class»+getLevel(level: int): string
«class»+getLevels(): dictionary
«class»+getLevelNames(): list<string>
LoggingRoot
+initialize(systemName: string, cfgPath: string)
-configureLevel()
-getBackendsFromCFG(cfgPath: string)
-getBackendOptionsFromCFG(cfgPath: string, backend: string)
1
Thread
CMRESHandler
StreamHandler
FileHandler
RemoteHandler
-site: string
-sleepTime: int
-interactivity: boolean
-logQueue: Queue
-transactions: list<list<tuple>>
-hostName: string
-alive: boolean
-maxBundleLogs: int
+bundleLogs()
+sendLogsToServer(logBundle: list<tuple>): boolean
1
«abstract»
Handler
+emit(record: LogRecord)
0..1
Formatter
+format(record: LogRecord)
BaseFormatter
-options: dictionary
1
ColoredBaseFormatter
+COLOR_MAP: dictionary
+LEVEL_MAP: dictionary
Resources.LogBackends
«abstract»
Backend
+createHandler(parameters: dictionary)
+setFormat(fmt: string, datefmt: string, options: dictionary)
+getHandler(): Handler
+setLevel()
«static»+createFormat(options: dictionary, level: int)
1
StderrBackend
ServerBackend
-site: string
-interactivity: bool
-sleepTime: int
ElasticBackend
-host: string
-port: int
-user: string
-password: string
-index: string
-bufferSize: int
-flushTime: int
FileBackend
-fileName: string
StdoutBackend

The old version of gLogger

This is the old version of gLogger. Please, go to *gLogger* to find the documentation on the current version. You can also see the different changes in the *Changes*.

Logger creation

Get a sublogger

gLogger is considered like the root logger. From it, we can create a child logger with the command:

```
gLogger.getSubLogger("logger")
```

This child logger can be used like gLogger and from it we can also get a sublogger and so on. We recommend you to not create a sublogger from a sublogger because there is no particular interest. Otherwise, note that the created sublogger is identified by its name and can be used again with the *getSubLogger()* method. For instance :

```
logger = gLogger.getSubLogger("logger")
newLogger = gLogger.getSubLogger("logger")
#logger is the same object as newLogger
```

child attribute in sublogger

gLogger and its children are owned by a system by default. It means that the name of the logger is preceded by the system and component name in the display. To prevent this feature, we can notify the program changing a boolean value : *child* in parameter of the *getSubLogger()* method like this :

```
gLogger.getSubLogger("logger", child=False)
```

This allows us to remove the system and component name. However, this feature seems buggy and should be used carefully. We recommend you to use only with a direct sublogger of gLogger and only if you execute a service, an agent or a script.

Levels

Level names and numbers

There are 9 different levels in DIRAC :

Level name	Level number
Always	40
Notice	30
Info	20
Verbose	10
Debug	0
Warn	-20
Error	-30
Exception	-30
Fatal	-40

They are numbered from 40 to -40. We use them according to the context attaching a certain level to a logger or to a message.

Set a level to a logger

We can set a certain level to a logger to hide some logs. It is a *V* system, which means that it functions with absolute values. For instance, if you set the level of gLogger to *Always*, only always and fatal logs will appear because their absolute values are superiors or equals to 40. To set a level, we use the *setLevel()* method like this :

```
logger.setLevel("notice")
```

Here, we set a notice level to this logger. However, once we have set a level to *gLogger*, these children will have the same level restriction, even if we try to change its level. In this way, the example logger will not send messages inferior to the absolute value of the *gLogger* level.

Get a logger level

We can obviously get a level associate to a logger via the *getLevel()* method.

Message

Naturally, it exists some functions to send a message. These methods take level names. In this way, we have :

- always(msg, varMsg='')
- notice(msg, varMsg='')
- info(msg, varMsg='')
- verbose(msg, varMsg='')
- debug(msg, varMsg='')
- warn(msg, varMsg='')
- error(msg, varMsg='')
- exception(msg, varMsg='', lException=False, lExcInfo=False)
- fatal(msg, varMsg='')

There are a *Msg* and *varMsg* where you can put any string you want in. There is no real difference between the two parameters.

The *exception* function contains two more parameters. The first has no effect on the message and should stay at False. Otherwise, the second parameter is more interesting because it allows or not the display of the file and the line where the exception occurs in the stack trace. We warn you that this method works only if an exception occurs.

```
try:
  1/0
except Exception:
  gLogger.exception("Division by 0", lExcInfo=True)
  gLogger.exception("Division by 0")
#will display:
#Division by 0
#== EXCEPTION == ZeroDivisionError
#  File "toto.py", line 132, in test_exception
```

(continues on next page)

(continued from previous page)

```
#    1/0
#
#ZeroDivisionError: integer division or modulo by zero
#===============
#
#Division by 0
#== EXCEPTION == ZeroDivisionError
#
#ZeroDivisionError: integer division or modulo by zero
#===============
```

These methods attach a certain level to the message, and as we seen above, if the absolute value of the *gLogger* level is superior to the absolute value of the message level, the log is not created.

```
glogger.setLevel("notice")
glogger.debug("this message will not be displayed")
#the last line will return False
```

Display

Basic display

The basic display for log message is:

:: [Year]-[Month]-[Day] [Hour]:[Minute]:[Second] UTC /[Component]/[Logname] [Levelname] : [Message]

Example:

```
2017-04-25 15:51:01 UTC Framework/logMultipleLines ALWAYS: this is a message
```

The date is UTC formatted and the system and the component names come from the configuration file. This display can vary according to the component, the backend and different option parameters.

Component

Client component

All messages from a client , wherever located, are displayed like:

```
[Year]-[Month]-[Day] [Hour]:[Minute]:[Second] UTC Framework/[Logname][Levelname] :␣
↪[Message]
```

The component name disappears and the system name becomes *Framework*. That is because there are no Client component in configuration files and *Framework* is the default system name.

Script Component

All messages from a script are displayed like:

```
[Message]
```

That is because the *parseCommandLine()* method modify one option parameter in *gLogger* : *showHeaders* to False. Let is talk more about these options.

Optional Parameter

showHeader option

showHeader is a boolean variable inside *gLogger* which allow us to hide or not the prefix of the message from the log. It can be changed via the *showHeader(val)* method and its default value is obviously True.

showThreads option

As the previous option, *showThreads* is a boolean variable inside *gLogger* which allow us to hide or not the thread ID in the log. This thread ID is created from the original thread ID of Python and modified by the backend to become a word. It is positioned between the log name and the level name like this:

```
2017-04-25 15:51:01 UTC Framework/logMultipleLines [PokJl] ALWAYS: this is a message
```

Its default value is False and we can set it via *showThreadIDs(val)* method. Nevertheless, if the *showHeaders* option is False, this option will have no effect on the display.

LogShowLine option

This option is only available from the *cfg* file and allows us to add extra information about the logger call between the logger name and the level of the message, like this:

```
2017-04-28 14:56:54 UTC TestLogger/SimplestAgent[opt/dirac/DIRAC/FrameworkSystem/
↪private/logging/Logger.py:160] INFO: Result
```

It is composed by the caller object path and the line in the file. As the previous option, it has no effect on the display if the *showHeaders* option is False.

LogColor option

This option is only available from the *cfg* file too, and only for *PrintBackend*. It allows us to add some colors according to the message level in the standard output like this:

```
2017-04-28 14:56:54 UTC TestLogger/SimplestAgent DEBUG: Result
2017-04-28 14:56:54 UTC TestLogger/SimplestAgent WARN: Result
2017-04-28 14:56:54 UTC TestLogger/SimplestAgent ERROR: Result
```

child attribute from *getSubLogger()* method

Previously, we saw the basic use of the *child* attribute from the *getSubLogger()* method. Actually, this attribute is considerably more complex and can modify the display in several ways but it seems to be illogic and buggy, so be careful using this attribute with a sublogger of a sublogger. Here is a simple example of its use with an agent running:

```
child = True: 2017-05-04 08:37:10 UTC TestLogger/SimplestAgent/log ALWAYS:␣
↪LoggingChildTrue
child = False: 2017-05-04 08:37:10 UTC log ALWAYS: LoggingChildFalse
```

Backends

Currently, there are four different backends inherited from a base which build the message according to the options seen above and another called *LogShowLine*. These four backends just write the message at associated place. There are :

Backend	Output
PrintBackend	standard output
StdErrBackend	error output
RemoteBackend	logserver output
FileBackend	file output

They need some information according to their nature. The PrintBackend needs a color option while the FileBackend needs a file name. In addition, the RemoteBackend needs a sleep time, an interactivity option and a site name. These information are collected from the *cfg* file.

Configuration

Configuration via the *cfg* file

Logger configuration

It is possible to configure some options of the logger via the *cfg* file. These options are :

Option	Description	Excpected value(s)
LogLevel	Set a level to gLogger	All the level names
LogBackends	Add backends to *gLogger* backend list	stdout, stderr, file, server
LogShowLine	Add information about the logger call	True, False
LogColor	Add color on messages, only for PrintBackend	True, False

Backend configuration

We also have the possibility to configure backend options via this file. To do a such operation, we just have to create a *BackendsOptions* section inside the component. Inside, we can add these following options:

Option	Description	Excpected value(s)
FileName	Set a file name for FileBackend	String value
SleepTime	Set a sleep time for RemoteBackend	Int value
Interactivity	Flush messages or not, for Remote Backend	True, False

cfg file example

Here is a component section which contains logger and backend configuration:

```
Agents
{
    SimplestAgent
    {
```

(continues on next page)

(continued from previous page)

```
      LogLevel = INFO
      LogBackends = stdout,stderr,file
      LogColor = True
      LogShowLine = True

      PollingTime = 60
      Message = still working...

      BackendsOptions
      {
        FileName = /tmp/logtmp.log
      }
    }
}
```

Configuration via command line argument

Moreover, it is possible to change the display via one program argument which is picked up by *gLogger* at its initalization. According to the number of *d* in the argument, the logger active or not different options and set a certain level. Here is a table explaining the working:

Argument	ShowHeader	showThread	Level
Default(Client/Agent/Services)	True	False	Notice
Default(Script)	False	False	Notice
-d	DefaultValue	DefaultValue	Verbose
-dd	True	DefaultValue	Verbose
-ddd	True	True	Debug

Multiple processes and threads

Multiple processes

DIRAC is composed by many micro services running in multiple processes. *gLogger* object is naturally different for two distinct processes and can not save the application from process conflicts. Indeed, *gLogger* is not process-safe, that means that two processes can encounter conflicts if they try to write on a same file at the same time. So, be careful to avoid the case.

Multiple threads

gLogger does not contain any safety against thread conflicts too, so be careful to not write on one file at the same time with two distinct threads.

3.7.12 DIRAC Resources

DIRAC Resources are logical entities representing computing resources and services usually provided by third parties. DIRAC is providing an abstract layer for various types of such services, e.g. Computing or Storage Elements, File Catalogs, etc. For each particular kind of service an implementation is provided and objects representing each service is created using its logical name by an appropriate Factory.

This section describes how Resources of different types can be used for developing DIRAC applications

FileCatalog

The full code documentation is available here `FileCatalog`

The *FileCatalog* relies on plugins to actually perform the operations, and will just loop over them

How to use it

Warning: *FileCatalog* class should only be used when no interactions with the Storages are expected. Typically, using FileCatalog to add new files without copying them will lead to lost data. If you want consistency between both, use the DataManager class

```
# Necessary import
from DIRAC.Resources.Catalog.FileCatalog import FileCatalog

# Instanciate a FileCatalog
fc = FileCatalog()

# LFNs to use as example
lfns = ['/lhcb/user/c/chaen/zozo.xml']
directory  = ['/lhcb/data/']

# Get the namespace metadata
fc.getFileMetadata(lfns)

# {'OK': True,
#  'Value': {'Failed': {},
#   'Successful': {'/lhcb/user/c/chaen/zozo.xml': {'Checksum': '29eddd7b',
#     'ChecksumType': 'Adler32',
#     'CreationDate': datetime.datetime(2015, 1, 23, 10, 28, 2),
#     'FileID': 171670021L,
#     'GID': 1470L,
#     'GUID': 'ECEC10C9-E7F3-36CA-8935-A9B483E97D2C',
#     'Mode': 436,
#     'ModificationDate': datetime.datetime(2015, 1, 23, 10, 28, 2),
#     'Owner': 'chaen',
#     'OwnerGroup': 'lhcb',
#     'Size': 769L,
#     'Status': 'AprioriGood',
#     'UID': 20269L}}}}


# Listing a directory
fc.listDirectory(directory)

# {'OK': True,
#  'Value': {'Failed': {},
#   'Successful': {'/lhcb/data/': {'Datasets': {},
#     'Files': {},
#     'Links': {},
#     'SubDirs': {'/lhcb/data/2008': True,
```

(continues on next page)

(continued from previous page)

```
#       '/lhcb/data/2009': True,
#       '/lhcb/data/2010': True,
#       '/lhcb/data/2011': True,
#       '/lhcb/data/2012': True,
#       '/lhcb/data/2013': True,
#       '/lhcb/data/2014': True,
#       '/lhcb/data/2015': True,
#       '/lhcb/data/2016': True,
#       '/lhcb/data/2017': True,
#       '/lhcb/data/2018': True}}}}}
```

Adding a new Catalog

The best doc for the time being is to look at an example like `FileCatalogClient`

Message Queues

Message Queues are fully described in the DIRAC Configuration as explained in the *Message Queues*. In the code, Message Queues can be used to publish messages which are arbitrary json structures. The *MQProducer* objects are used in this case:

```
from DIRAC.Resources.MessageQueue.MQCommunication import createProducer

result = createProducer( "mardirac3.in2p3.fr::Queues::TestQueue" )
if result['OK']:
   producer = result['Value']
# Publish a message which is an arbitrary json structure
result = producer.put( message )
```

The Messages are received by consumers. Consumers are objects of the MQConsumer class. These objects can request messages explicitly:

```
from DIRAC.Resources.MessageQueue.MQCommunication import createConsumer

result = createConsumer( "mardirac3.in2p3.fr::Queues::TestQueue" )
if result['OK']:
   consumer = result['Value']
result = consumer.get( message )
if result['OK']:
  message = result['Value']
```

consumers can be instantiated with a callback function that will be called automatically when new messages will arrive:

```
from DIRAC.Resources.MessageQueue.MQCommunication import createConsumer

def myCallback( headers, message ):
  <function implementation>

 result = createConsumer( "mardirac3.in2p3.fr::Queues::TestQueue", callback =␣
↪myCallback )
 if result['OK']:
    consumer = result['Value']
```

The destination name (queue or topic) in the consumer/producer instantiation must be given as fully qualified name like "mardirac3.in2p3.fr::Queues::TestQueue" or "mardirac3.in2p3.fr::Topics::TestTopic".

StorageElement

The full code documentation is available here `StorageElementItem`

The *StorageElement* relies on plugins to actually perform the operations, and will just loop over them

How to use it

Warning: *StorageElement* class should only be used when no interactions with the Catalogs are expected. Typically, using StorageElement to add new files without registering them will lead to dark data. If you want consistency between both, use the DataManager class

```
# Necessary import
from DIRAC.Resources.Storage.StorageElement import StorageElement

# Instanciate a StorageElement
se = StorageElement('CERN-USER')

# LFNs to use as example
lfns = ['/lhcb/user/c/chaen/zozo.xml']

# Get the physical metadata
se.getFileMetadata(lfns)

# {'OK': True,
#  'Value': {'Failed': {},
#   'Successful': {'/lhcb/user/c/chaen/zozo.xml': {'Accessible': True,
#     'Checksum': '29eddd7b',
#     'Directory': False,
#     'Executable': False,
#     'File': True,
#     'FileSerialNumber': 51967,
#     'GroupID': 1470,
#     'LastAccess': '2017-07-25 15:06:19',
#     'Links': 1,
#     'ModTime': '2017-07-25 15:06:19',
#     'Mode': 256,
#     'Readable': True,
#     'Size': 769L,
#     'StatusChange': '2017-07-25 15:06:19',
#     'UserID': 56212,
#     'Writeable': False}}}}

# Get the URL, using operation defaults for the protocol
se.getURL(lfns)

# {'OK': True,
#  'Value': {'Failed': {},
#   'Successful': {'/lhcb/user/c/chaen/zozo.xml': 'root://eoslhcb.cern.ch//eos/lhcb/
↪grid/user/lhcb/user/c/chaen/zozo.xml'}}}
```

(continues on next page)

(continued from previous page)

```
# Specify the protocol to use
se.getURL(lfns, protocol = 'srm')

# {'OK': True,
#  'Value': {'Failed': {},
#   'Successful': {'/lhcb/user/c/chaen/zozo.xml': 'srm://srm-eoslhcb.cern.ch:8443/srm/
↪v2/server?SFN=/eos/lhcb/grid/user/lhcb/user/c/chaen/zozo.xml'}}}
```

Adding a new plugin/protocol

The best doc for the time being is to look at an example like `GFAL2_XROOTStorage`

3.7.13 Developing Web Portal Pages

3.7.14 Code quality

DIRAC code should be coded following the conventions explained in *Coding Conventions*. There are automatic tools that can help you to follow general good code quality rules.

Specifically, 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 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.

3.8 Documenting your developments

Where should you document your developments? Well, at several places, indeed, depending on the documentation we are talking about:

3.8.1 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.

3.8.2 Release documentation

Releases documentation can be found in 2 places: release notes, and github wiki:

- release notes are automatically created from the first comment in the pull requests, please describe the changes between BEGRINRELEASENOTES and ENDRELEASENOTES as presented by the template provided
- 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.

3.8.3 Full development documentation

As said at the beginning of this guide, this documentation is in git at DIRAC/docs. 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.

Some parts of the documentation can use UML diagrams. They are generated from .uml files with plantuml. Sphinx support plantuml but ReadTheDocs didn't, so you have to convert .uml in .png with `java -jar plantuml.jar file.uml`.

3.8.4 Component Options documentation

The agent, service and executor options are documented in their respective module docstring via literal include of their options in the ConfigTemplate.cfg:

```
.. literalinclude:: ../ConfigTemplate.cfg
  :start-after: ##BEGIN MyComponent
  :end-before: ##END
  :dedent: 2
  :caption: MyComponent options
```

Around the section in the *ConfigTemplate.cfg* configuring the component the *##BEGIN MyComponent* and *##END* tags need set so that the include is restricted to the section belonging to the component. The options *:dedent:* and *:caption:* are optional, but create a nicer output.

3.9 Testing (VO)DIRAC

3.9.1 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.

3.9.2 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.

3.9.3 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 integration tests show some broken software, you should not bother running any system test.

3.9.4 Who should write (and run) the 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.

3.10 Tools and methodology

3.10.1 Unit tests

In DIRAC unit tests should be prepared by 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[1] should be read carefully to isolate all the paths of executions. For each of such cases the developer should prepare, formulate and define all required inputs and outputs, configurations, internal and external objects states, underlying components etc.. 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.

Step 3. **Test execution**

[1] Or even better software requirements document, if any of such exists. Otherwise this is a great opportunity to prepare one.

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...

3.10.2 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[2]. 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.

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.

[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.

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`:

```
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 = [ basestring ]
  def export_isThere( self, cheese ):
    return cheeseShopDB.isThere( cheese )

  types_buyCheese = [ basestring, float ]
  def export_buyCheese( self, cheese, quantity ):
    return cheeseShopDB.buyCheese( cheese, quantity )

  # ... and so on, so on and so on, i.e:
  types_insertCheese = [ basestring, float, float ]
  def export_insertCheese( self, cheeseName, price, quantity ):
    return cheeseShopDB.insertCheese( cheeseName, price, quantity )
```

And here for `CheeseShopClient` class:

```
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

class CheeseShopClient( Client ):

  def __init__( self, money, shopOwner = None ):
```

(continues on next page)

[3] And eventually is killing him with a gun. At least in a TV show.

(continued from previous page)

```
    self.__money = money
    self.shopOwner = shopOwner

  def buy( self, cheese, quantity = 1.0 ):

    # is it really cheese, you're asking for?
    if not isinstance( cheese, Cheese ):
      raise SpanishInquisitionError( "It's stone dead!" )

    # and the owner is in?
    if not self.shopOwner:
      return S_ERROR("Shop is closed!")

    # and cheese is in the shop depot?
    res = self.shopOwner.isThere( cheese.name )
    if not res["OK"]:
      return res

    # and you are not asking for too much?
    if quantity > res["Value"]["Quantity"]:
      return S_ERROR( "Not enough %s, sorry!" % cheese.name )

    # and you have got enough money perhaps?
    price = quantity * res["Value"]["Price"]
    if self.__money < price:
      return S_ERROR( "Not enough money in your pocket, get lost!")

    # so we're buying
    res = self.shopOwner.buyCheese( cheese.name, quantity )
    if not res["OK"]:
      return res
    self.__money -= price

    # finally transaction is over
    return S_OK( self.__money )
```

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 shoudn'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]:

- Want to buy Norwegian blue parrot:

[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.

  - 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:

```
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 = Chesse( "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 } ) )
```

(continues on next page)

(continued from previous page)

```
    # 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" ] )
```

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.

3.10.3 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.

The DIRAC/tests part of DIRAC repository

The DIRAC repository contains a tests section `https://github.com/DIRACGrid/DIRAC/tree/integration/tests` that holds integration, regression, workflow, system, and permormance tests. These tests are not only used for the certification process. Some of them, in fact, might be extremely useful for the developers.

Integration tests for jobs

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.

Integration tests for services

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 this test file

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.

Integration tests are a good example of the type of tests that can be run by a machinery. Continuous integration tools like Jenkins are indeed used for running these type of tests.

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 DIRAC/tests (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 DIRAC.tests.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.
- if you have tests that need to access other DBs, you should also have them ready, again used 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=v6r20 # the branch of DIRAC that will be checkout

export PRERELEASE=True # if you want to test a DIRAC pre-release
export DEBUG=True

export DB_USER=Dirac # Normally it's always "Dirac"
export DB_PASSWORD=password # replace it with what you need --- in Jenkins you␣
↪can inject passwords
export DB_ROOTUSER=admin  # replace it with what you need - either "root" or
↪"admin"
export DB_ROOTPWD=password # replace it with what you need --- in Jenkins you can␣
↪inject passwords
export DB_HOST=db-test.example.org # may also be localhost if needed
export DB_PORT=5501 # replace it with what you need
export NoSQLDB_HOST=localhost # elasticsearch will be installed locally
export NoSQLDB_PORT=9200 # default
```

(continues on next page)

(continued from previous page)

```
# moving into TestCode directory for convenience
mkdir -p $PWD/TestCode
cd $PWD/TestCode

git clone git://github.com/DIRACGrid/DIRAC.git
cd DIRAC
git checkout rel-$DIRACBRANCH

# moving to base dir
cd ../..

set -e # may be removed
source TestCode/DIRAC/tests/Jenkins/dirac_ci.sh

# now we start installing the server

X509_CERT_DIR=$SERVERINSTALLDIR/etc/grid-security/certificates/ fullInstallDIRAC
 ↪# this will install EVERYTHING!!! ---> will be long!

unset PYTHONOPTIMIZE


echo "**** INSTALLATION DONE ****"
echo "**** STARTING INTEGRATION TESTS ****"

cp -r $TESTCODE/DIRAC/tests/ $SERVERINSTALLDIR/DIRAC/

echo -e '***' $(date -u) "**** Core TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/Test_ElasticsearchDB.py >>␣
 ↪testOutputs.txt 2>&1

echo -e '***' $(date -u) "**** Accounting TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/AccountingSystem/Test_
 ↪DataStoreClient.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u) "**** FRAMEWORK TESTS (partially skipped) ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/Framework/Test_
 ↪InstalledComponentsDB.py >> testOutputs.txt 2>&1
#pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/Framework/Test_LoggingDB.py >>␣
 ↪testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** RMS TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/RequestManagementSystem/Test_
 ↪Client_Req.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** RSS TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/ResourceStatusSystem/Test_
 ↪FullChain.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/ResourceStatusSystem/Test_
 ↪Publisher.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/ResourceStatusSystem/Test_
 ↪ResourceManagement.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/ResourceStatusSystem/Test_
 ↪ResourceStatus.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/ResourceStatusSystem/Test_
 ↪SiteStatus.py >> testOutputs.txt 2>&1
```

(continues on next page)

(continued from previous page)

```
echo -e '***' $(date -u)  "**** WMS TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪JobDB.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪JobLoggingDB.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪TaskQueueDB.py >> testOutputs.txt 2>&1
python $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪Client_WMS.py $WORKSPACE/TestCode/DIRAC/tests/Integration/
↪WorkloadManagementSystem/sb.cfg >> testOutputs.txt 2>&1
python $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪SandboxStoreClient.py $WORKSPACE/TestCode/DIRAC/tests/Integration/
↪WorkloadManagementSystem/sb.cfg >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_
↪JobWrapper.py >> testOutputs.txt 2>&1
python $SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/
↪createJobXMLDescriptions.py >> testOutputs.txt 2>&1
$SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_dirac-
↪jobexec.sh >> testOutputs.txt 2>&1
$SERVERINSTALLDIR/DIRAC/tests/Integration/WorkloadManagementSystem/Test_TimeLeft.
↪sh >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** DMS TESTS ****\n"
## DFC
echo "Test DFC DB" >> testOutputs.txt 2>&1
python $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_
↪FileCatalogDB.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "Reinitialize the DFC DB\n" >> testOutputs.txt 2>&1
diracDFCDB >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "Run the DFC client tests as user without admin␣
↪privileges" >> testOutputs.txt 2>&1
echo -e '***' $(date -u)  "Getting a non privileged user\n" >> testOutputs.txt 2>&
↪1
dirac-proxy-init -C $WORKSPACE/ServerInstallDIR/user/client.pem -K $WORKSPACE/
↪ServerInstallDIR/user/client.key $DEBUG
python $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_Client_
↪DFC.py >> testOutputs.txt 2>&1
#diracDFCDB
#python $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_
↪FileCatalogDB.py >> testOutputs.txt 2>&1

echo "Reinitialize the DFC DB" >> testOutputs.txt 2>&1
diracDFCDB >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "Restart the DFC service\n" &>> testOutputs.txt
dirac-restart-component DataManagement FileCatalog $DEBUG &>> testOutputs.txt

echo -e '***' $(date -u)  "Run it with the admin privileges" >> testOutputs.txt 2>
↪&1
echo -e '***' $(date -u)  "getting the prod role again\n" >> testOutputs.txt 2>&1
dirac-proxy-init -g prod -C $WORKSPACE/ServerInstallDIR/user/client.pem -K
↪$WORKSPACE/ServerInstallDIR/user/client.key $DEBUG >> testOutputs.txt 2>&1
python $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_Client_
↪DFC.py >> testOutputs.txt 2>&1
```

(continues on next page)

(continued from previous page)

```
diracDFCDB
python $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_
↪FileCatalogDB.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** FTS TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/DataManagementSystem/Test_Client_
↪FTS3.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** MONITORING TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/Monitoring/Test_
↪MonitoringReporter.py >> testOutputs.txt 2>&1
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/Monitoring/Test_MonitoringSystem.
↪py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** TS TESTS ****\n"
pytest $SERVERINSTALLDIR/DIRAC/tests/Integration/TransformationSystem/Test_Client_
↪Transformation.py >> testOutputs.txt 2>&1

echo -e '***' $(date -u)  "**** Resources TESTS ****\n"
python $SERVERINSTALLDIR/DIRAC/tests/Integration/Resources/Storage/Test_Resources_
↪GFAL2StorageBase.py ProductionSandboxSE >> testOutputs.txt 2>&1


echo -e '***' $(date -u) "**** TESTS OVER ****\n"

cp testOutputs.txt $WORKSPACE/

echo -e '***' $(date -u) "**** Now stopping/removing stuff ****\n"

clean

echo -e '***' $(date -u) "*** DONE ****\n"
```

This test is VERY complete, as you can see. If you are only testing locally, it may be too much, but as it is it's perfect for a job running in Jenkins.

At the same time, if you are a developer you should be able to extrapolate from the above those parts that you need, in case you are testing only one specific service.

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 DIRAC repository contains, in the *tests* package 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.

The server lbcertifdirac6.cern.ch is used as "DIRAC certification machine". With "certification machine" we mean that it is a full DIRAC installation, that connects to grid resources, and through which we certify pre-production versions. Normally, the latest DIRAC pre-releases are installed there. Its access is restricted to some power users, for now, but do request access if you need to do some specific system test. This installation is usually not done for running private tests, but in a controlled way can be sometimes tried.

3.10.4 The certification process

Each DIRAC release go through a long and detailed certification process. A certification process is a series of steps that include unit, integration, validation and system tests. We use detailed trello boards and slack channel. Please DO ASK to be included in such process.

The template for DIRAC certification process can be found at the trello board and the slack channel is here

3.10.5 Footnotes

3.11 Developer Guides for DIRAC Systems

Here the reader can find technical documentation for developing DIRAC systems

Table of contents

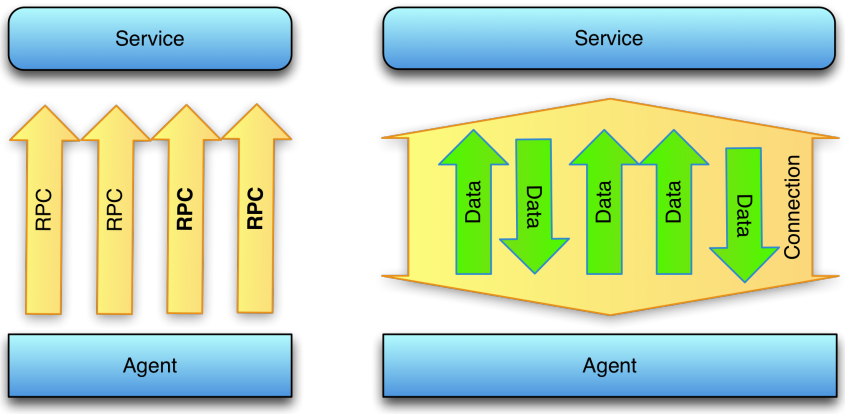
3.11.1 Framework Overview

Information regarding use of the DIRAC Framework to build new components

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 asynchonous 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:

```
""" This is a simple example of implementation of a Ping/Pong service for executors.

    This service does not any specific configuration to run, only the Port number, and authz e.g.:

    {
      Port = 9145
      {
        Authorization
        {
          Default = all
        }
      }
    }
"""

from DIRAC import S_OK
from DIRAC.Core.DISET.RequestHandler import RequestHandler


class PingPongHandler(RequestHandler):

  MSG_DEFINITIONS = {'Ping': {'id': (int, long)},
                     'Pong': {'id': (int, long)}}

  auth_conn_connected = ['all']

  def conn_connected(self, trid, identity, kwargs):
    """
    This function will be called when a new client connects.
    It is not mandatory to have this function

    params:
      @trid: Transport ID: Unique for each connection
      @identity: Unique for each client even if it reconnects
      @kwargs: Arguments sent by the client for the connection
    """
    # Do something with trid/identity/kwargs if needed
    return S_OK()

  auth_conn_drop = ['all']

  def conn_drop(self, trid):
    """
    This function will be called when a client disconnects.
    It is not mandatory to have this function
    """
```

(continues on next page)

(continued from previous page)

```
47      return S_OK()
48
49    auth_msg_Ping = ['all']
50
51    def msg_Ping(self, msgObj):
52      """
53      Callback for Ping message
54      """
55      pingid = msgObj.id
56      result = self.srv_msgCreate("Pong")
57      if not result['OK']:
58        # Something went wrong :P
59        return result
60      pongObj = result['Value']
61      pongObj.id = pingid
62      # Could have been
63      # return self.srv_msgReply( pongObj )
64      return self.srv_msgSend(self.srv_getTransportID(), pongObj)
```

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 it's 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:

```
1  import sys
2  import time
3  from DIRAC import S_OK, S_ERROR
```

(continues on next page)

(continued from previous page)

```
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
  return S_ERROR( "Could not reconnect... :P" )

if __name__ == "__main__":
  msgClient = MessageClient( "Framework/PingPong" )
  msgClient.subscribeToMessage( 'Pong', pongCB )
  msgClient.subscribeToDisconnect( disconnectedCB )
  result = msgClient.connect()
  if not result[ 'OK' ]:
    print "CANNOT CONNECT: %s" % result[ 'Message' ]
    sys.exit(1)
  result = sendPingMsg( msgClient )
  if not result[ 'OK' ]:
    print "CANNOT SEND PING: %s" % result[ 'Message' ]
    sys.exit(1)
  #Wait 10 secs of pingpongs :P
  time.sleep( 10 )

```

Let's start with like 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.

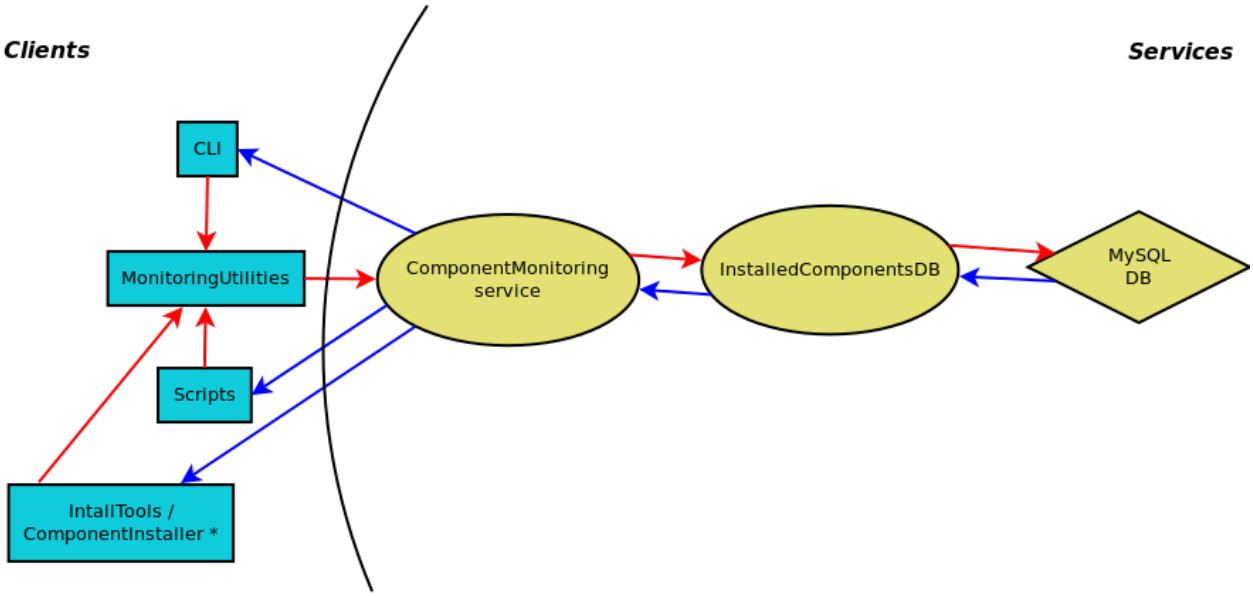
Function *sendPingMsg* in line 5 onwards just creates a *Ping* message and sends it to the server via the supplied *msgClient*.

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.

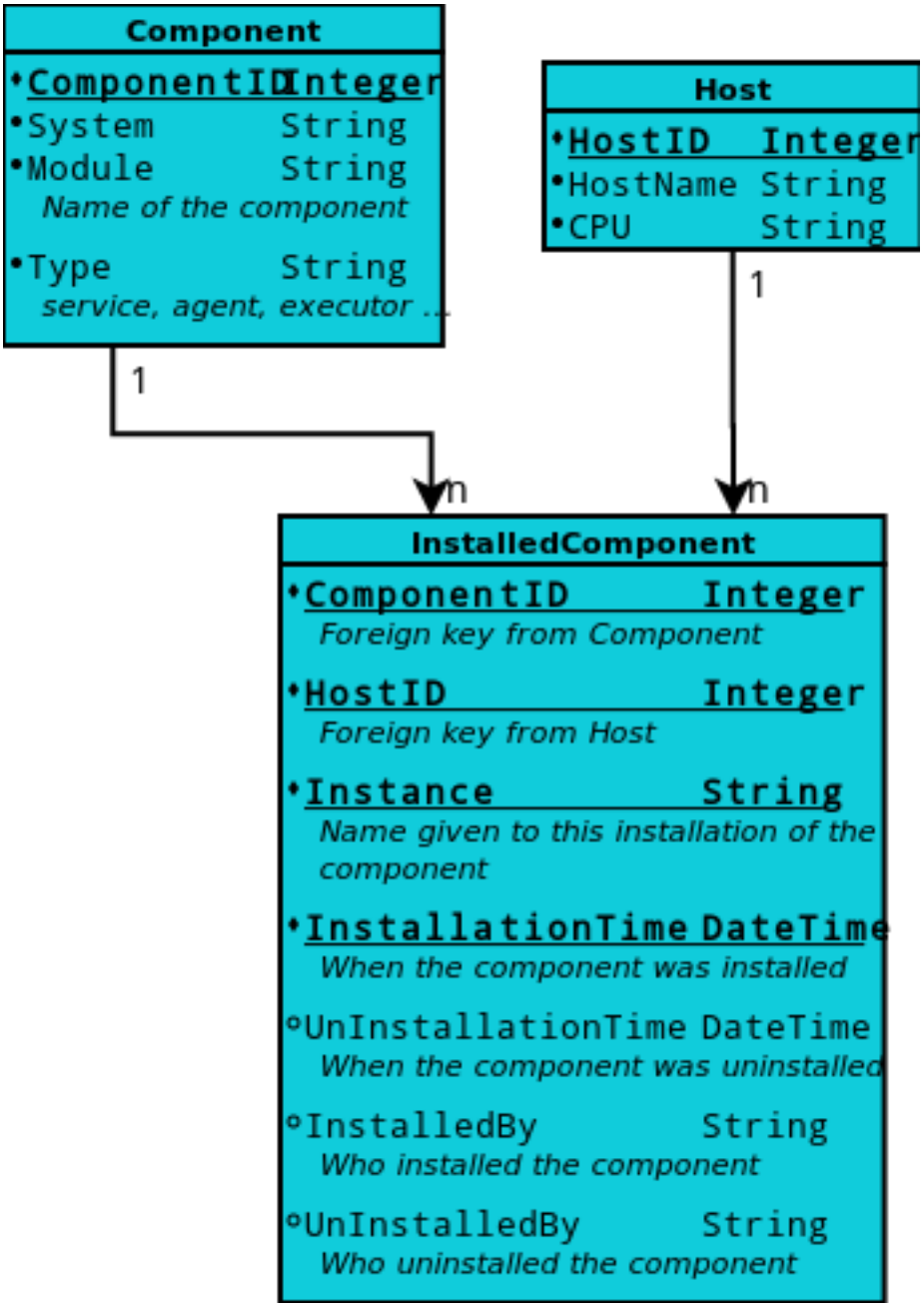
Static Component Monitoring

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:



* InstallTools is replaced by ComponentInstaller in v6r15

All of the static information is stored in a MySQL database, InstalledComponentsDB. This database contains 3 tables, as illustrated below:



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:

```
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 )
```

(continues on next page)

(continued from previous page)

```
  module = Column( 'Module', String( 32 ), nullable = False )
  cType = Column( 'Type', String( 32 ), nullable = False )

  def __init__( self, system = null(), module = null(), cType = null() ):
    self.system = system
    self.module = module
    self.cType = cType
    self.installationList = []
```

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:

```
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:

```
engine = create_engine( 'sqlite:////route/to/my/db.db' , pool_recycle = 3600, echo_
↪pool = True, echo = True )
```

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:

```
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'
```

Dynamic Component Monitoring

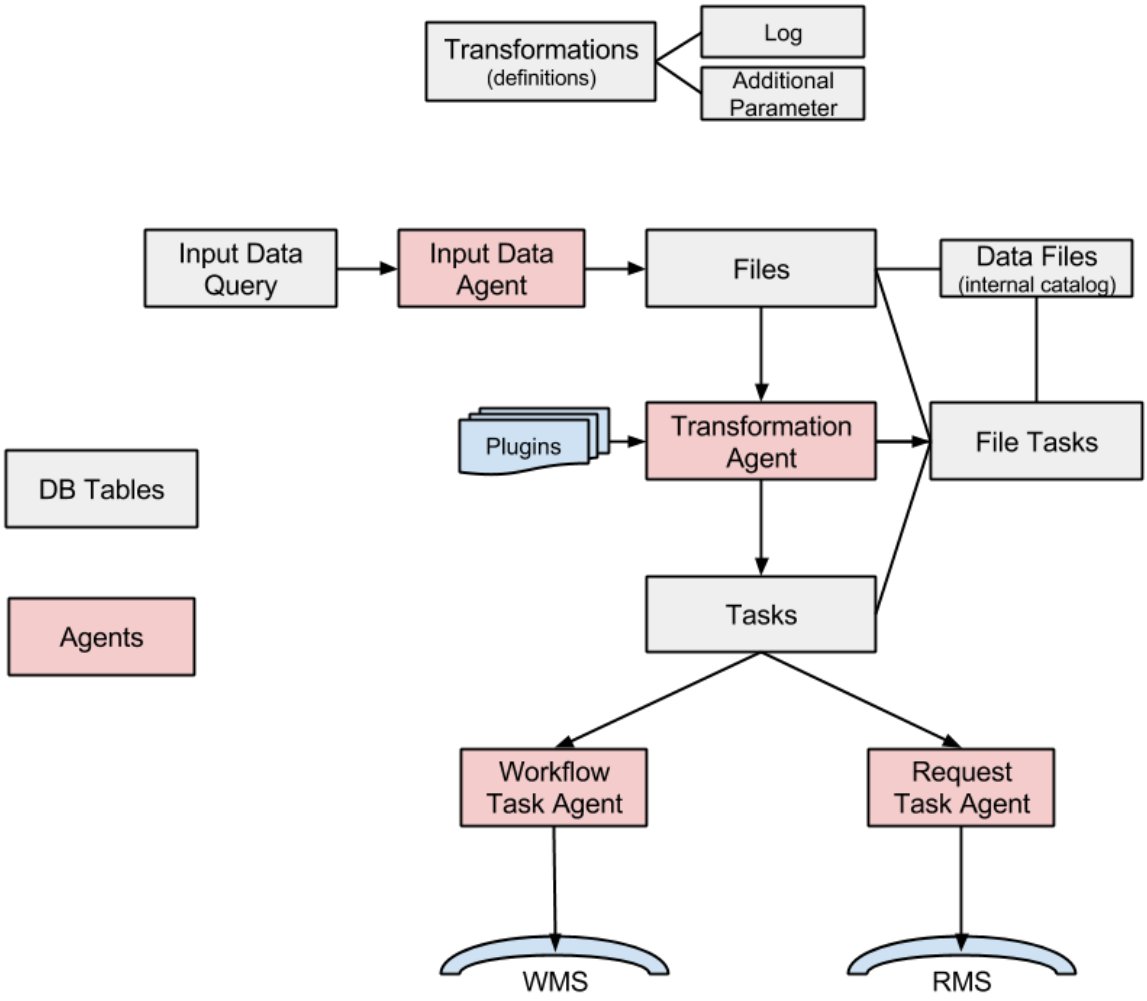
This system takes care of managing monitoring information of DIRAC component. It is based on ElasticSearch database. It is based on MonitoringSystem. The information is collected by the __storeProfiling periodic task on the SystemAdministartor. The task is disabled by default. The MonitoringReporter is used to propagate the DB whith the collected values.

3.11.2 TransformationSystem

Table of Contents

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:

```
mysql> use TransformationDB;
Database changed
mysql> show tables;
+------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Tables_in_TransformationDB   |
+------------------------------+
| AdditionalParameters         |
| DataFiles                    |
| TaskInputs                   |
| TransformationFileTasks      |
| TransformationFiles          |
| TransformationInputDataQuery |
| TransformationLog            |
| TransformationTasks          |
| Transformations              |
+------------------------------+
```

Note that since version v6r10, there are important changes in the TransformatioDB, 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 it submits to the WMS. There are some capabilities in the form of TaskManager plugins, please refer to <https://github.com/DIRACGrid/DIRAC/wiki/DIRAC-v6r13#changes-for-transformation-system>'_. These plugins determine how the destination site is chosen.
  - 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

Table of contents

3.11.3 Monitoring System

Overview

The system is storing monitoring information. It means the data stored in the database is time series. It can be used to monitor: -computing task (for example: Grid jobs, etc.) -computing infrastructures (for example: machines, etc.) -data movement (for example: Data operation etc.)

This system is based on ElasticSearch, RabbitMQ and DIRAC plotting facilities. It allows to introduce new monitoring types by adding minimal code.

Architecture

It is based on layered architecture and is based on DIRAC architecture:

- **Services**
  - MonitoringHandler: DISET request handler base class for the MonitroingDB
- **DB**
  - MonitoringDB: It is a based on ElasticSearch database and provides all the methods which needed to create the reports. Currently, it supports only one type of query: It creates a dynamic buckets which will be used to retrieve the data points. The query used to retrieve the data points is retrieveBucketedData. As you can see it uses the ElasticSearch QueryDSL language. Before you modify this method please learn this language.
  - **private:**
    - Plotters: It contains all Plotters used to create the plots. More information will be provided later.
    - DBUtils: It provides utilities used to manipulate the data.
    - MainReporter: It contains all available plotters and it has a reference to the database. It uses the db to retrieve the data and the Plotter to create the plot.
    - TypeLoader: It loads all Monitoring types.
- **Clients**
  - MonitoringClient is used to interact withe the Monitoring service.
  - Types contains all Monitoring types.

How to add new monitoring type?

A new monitoring type can be added:

- You have to define the monitoring values and the conditions. For example: cond1, cond2, monitoring value id ex1 Monitoring/Client/Types/Example.py For more information please have a look WMSHistory.py

  self.setKeyFields([‘cond1’, ‘cond2’]) self.setMonitoringFields([‘ex1’])

- create the plotter: MonitoringSystem/Client/private/Plotters/ExamplePlotter.py Note: The file name must ends with Plotter word. You have to implement two functions:

  def _reportExample(self, reportRequest): def _plotExample(self, reportRequest, plotInfo, filename):

  In the Monitoring page you will see and Example. But if you want to rename it: _reportExample = ‘Test1’ def _reportExample(self, reportRequest):

  More information: WMSHistoryPlotter.py

- Add the new monitoring to the WebAppDIRAC Monitoring application.

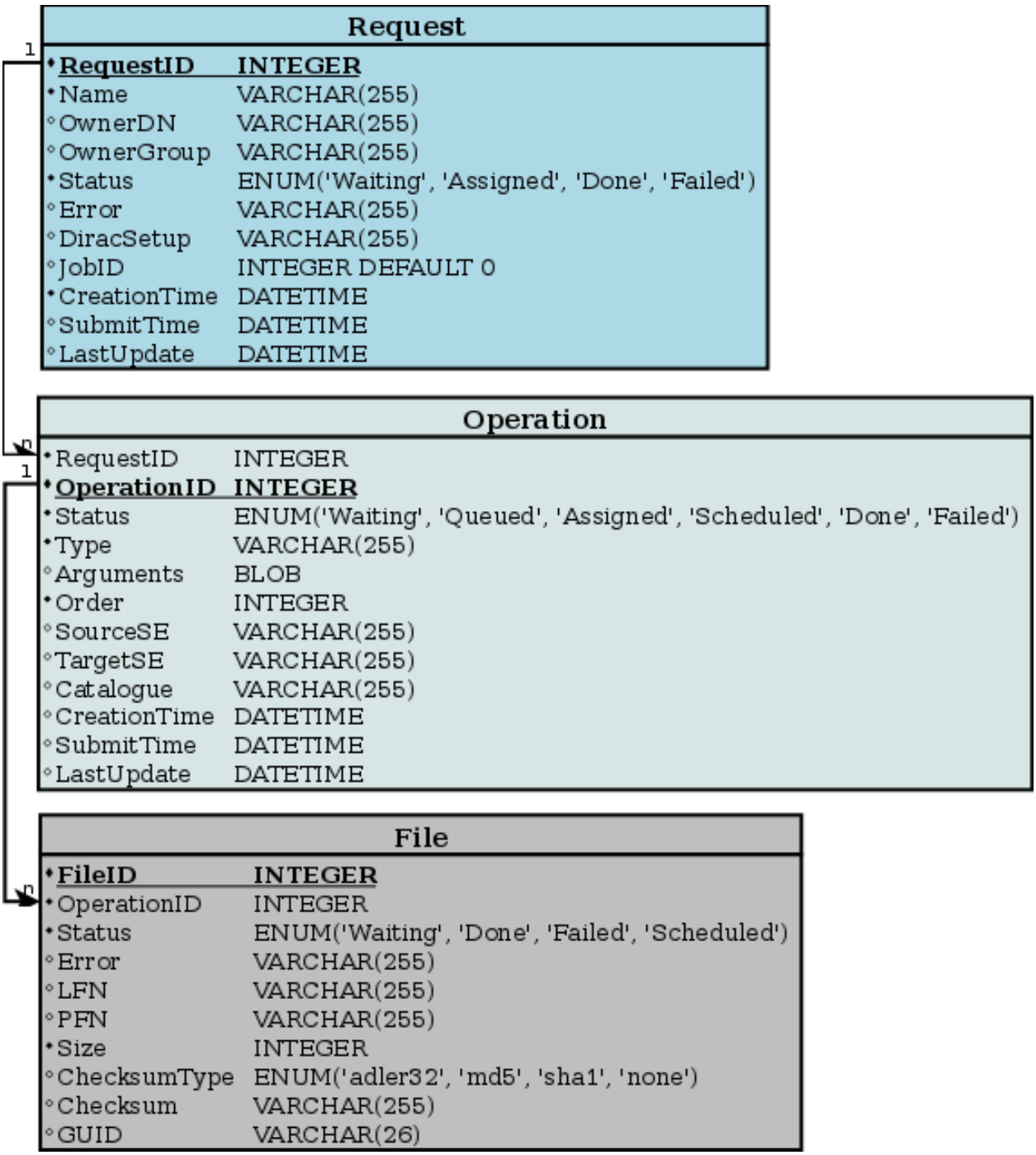
3.11.4 Request Management System

System Overview

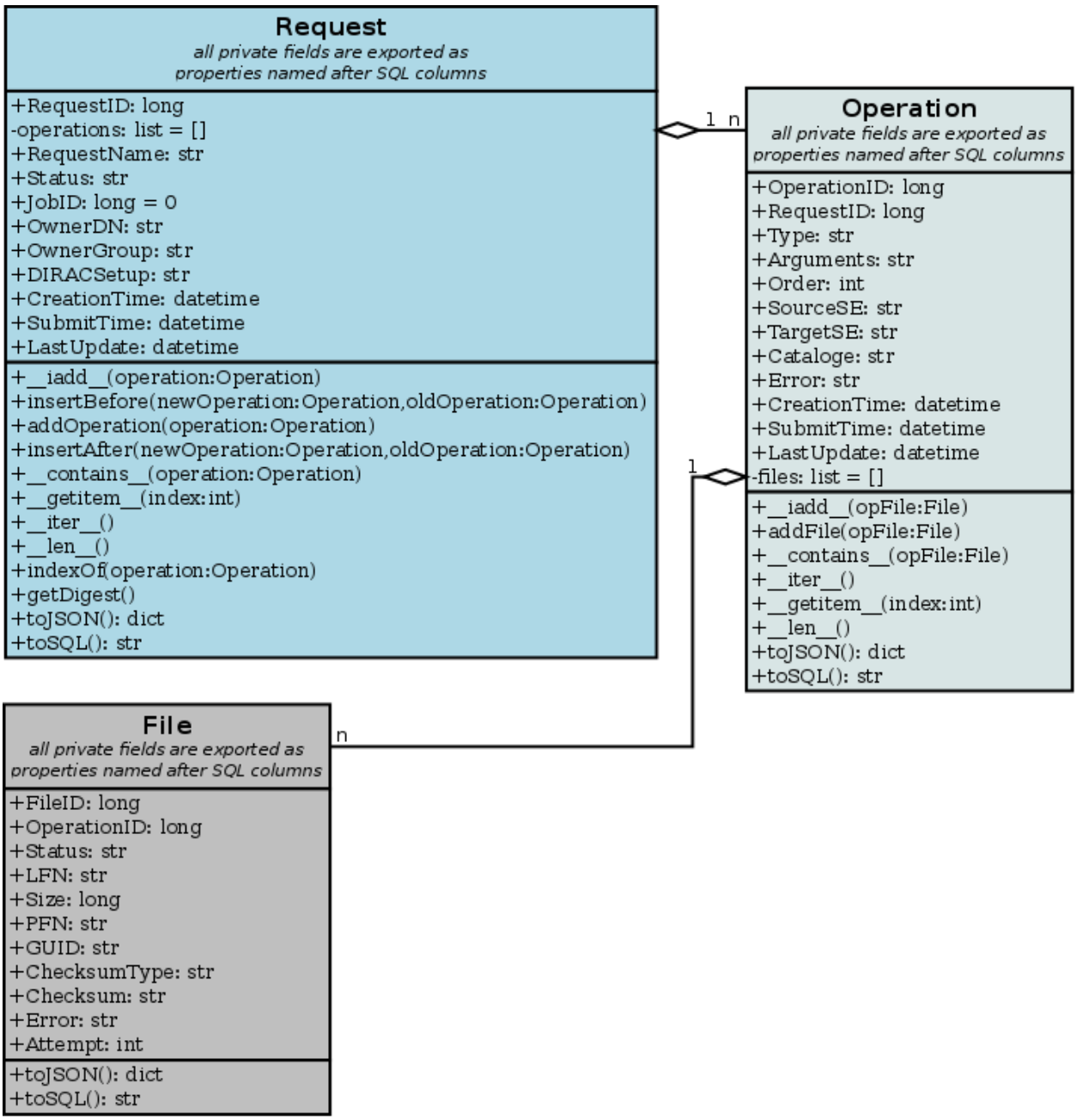
The Request Management System (RMS) is designed for management of simple operations that are performed asynchronously on behalf of users - owners of the requests. The RMS is used for multiple purposes: failure recovery (failover system), data management tasks and some others. It is designed as an open 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 avaiable and useful queries to the *ReqDB* are exposed to the request client (*ReqClient*) through *ReqManager* service.

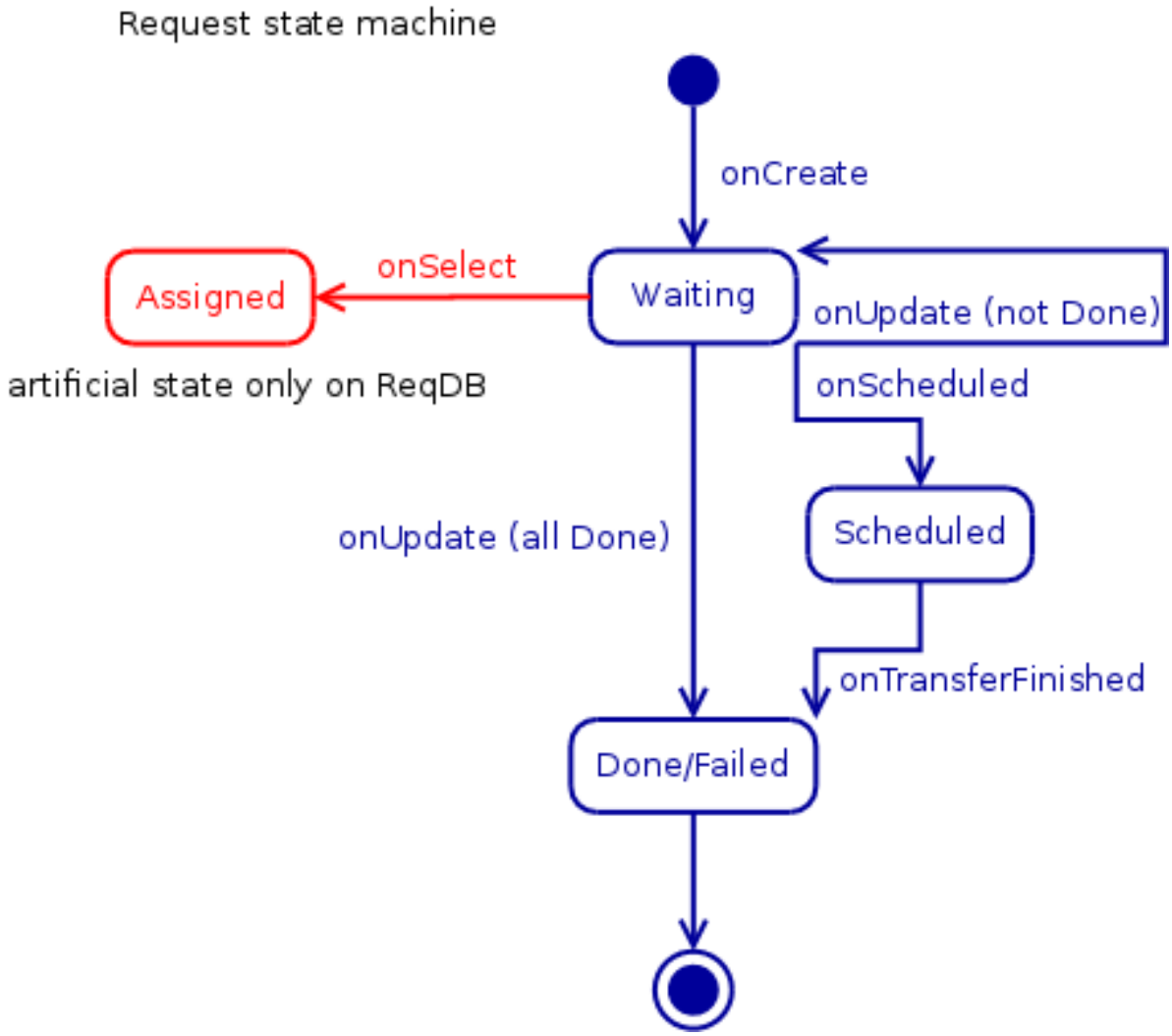


Each table in the *ReqDB* has a corresponding class in the new API fully supporting CRUD operations. Each table column is exposed as a property in the related class.

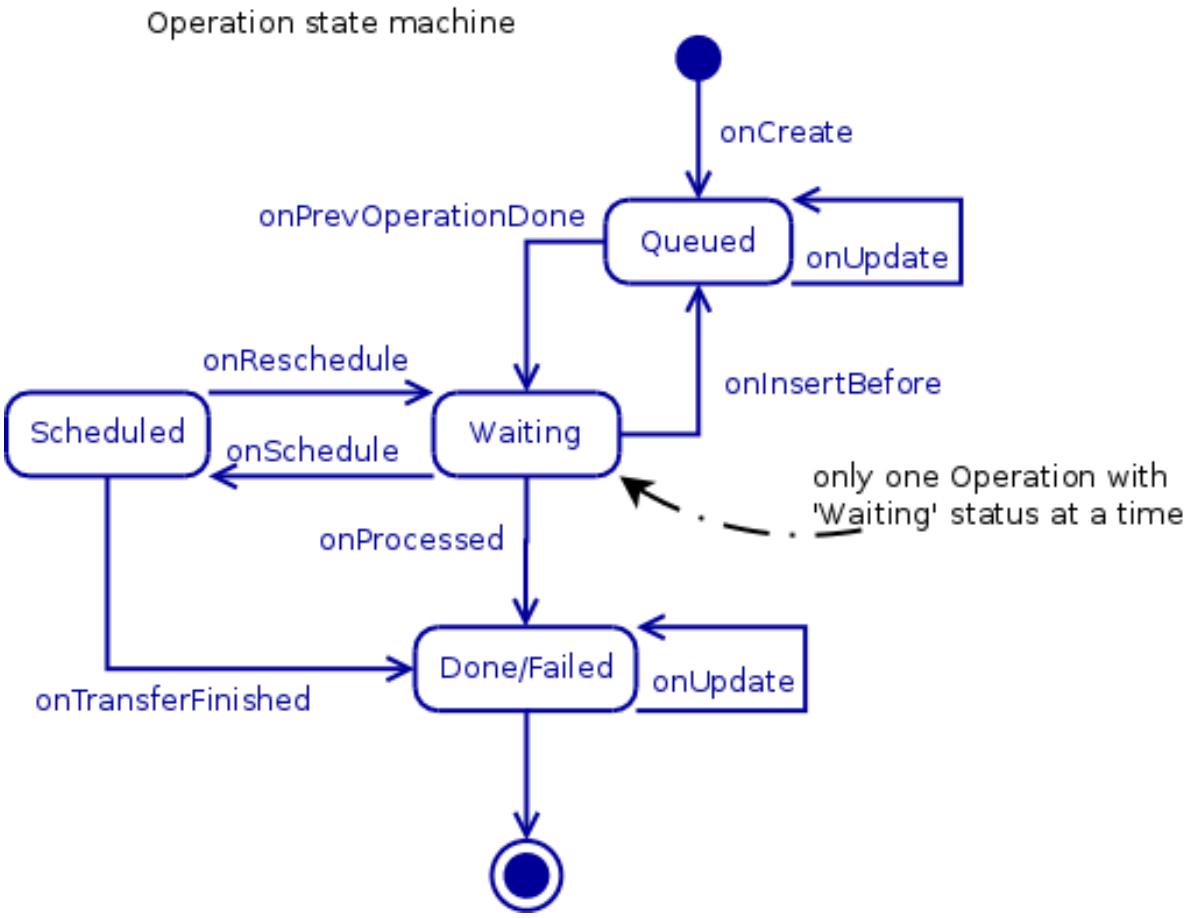


The record class is instrumented with the internal observer for its children (a *Request* instance is observing states for all defined *Operations*, each *Operation* is observing states of all its *Files*) and built in state machine, which automatizes state propagation:

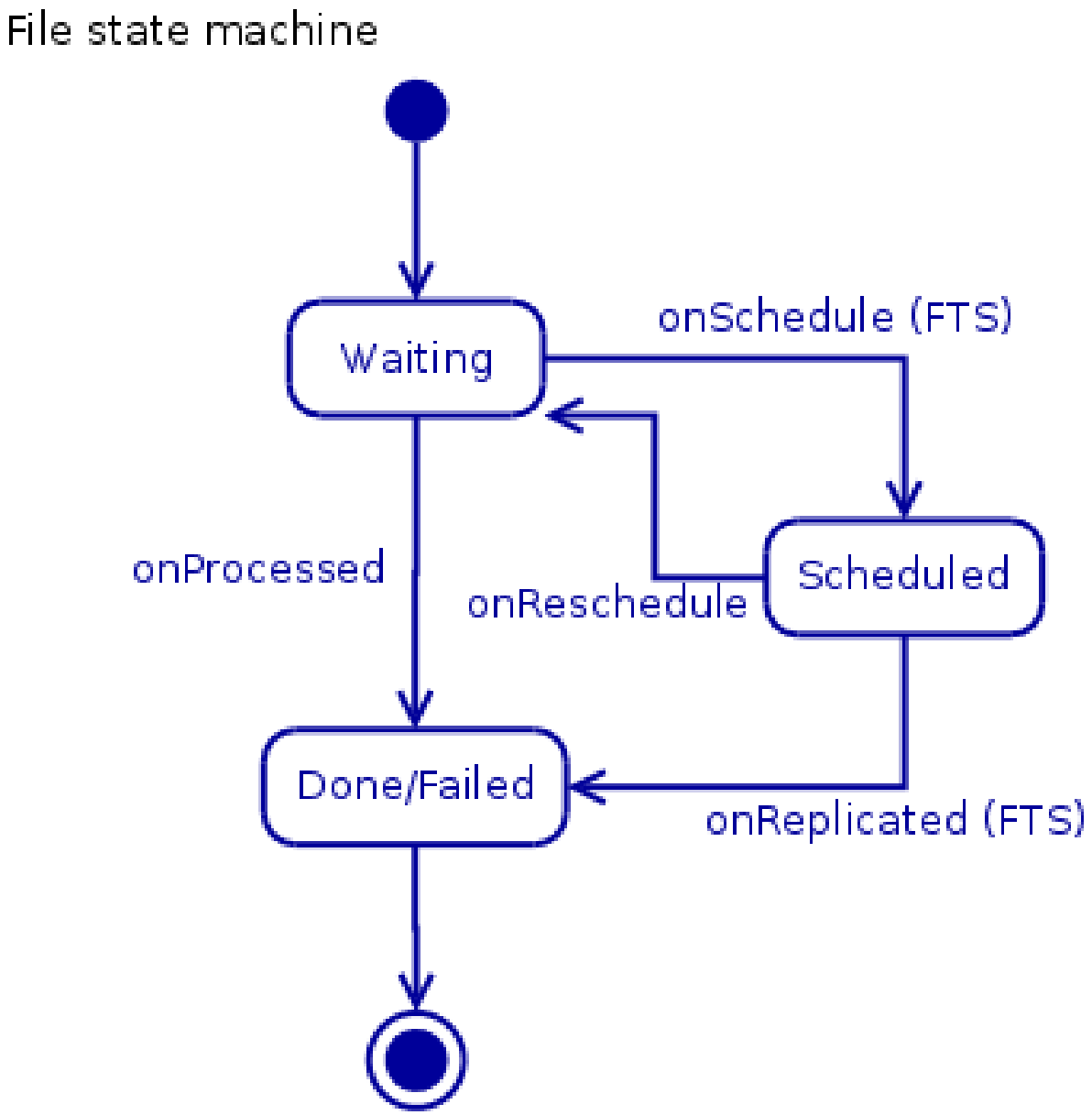
- state machine for *Request*



- 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:

```
>>> 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"
```

(continues on next page)

(continued from previous page)

```
>>> operation = Operation() # # create new operation
>>> operation.Type = "ReplicateAndRegister"
>>> operation.TargetSE = [ "CERN-USER", "PIC-USER" ]
>>> opFile = File() # #  create File instance
>>> opFile.LFN = "/foo/bar/baz" # # and fill some bits
>>> opFile.Checksum = "123456"
>>> opFile.ChecksumType = "adler32"
>>> operation.addFile( opFile ) # # add File to Operation
>>> request.addOperation( operation ) # # add Operation to Request
```

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:

```
>>> for op in request: print op.Type
ReplicateAndRegister
>>> for opFile in operation: print opFile.LFN, opFile.Status, opFile.Checksum
/foo/bar/baz Waiting 123456
>>> len( request ) # # number of operations
1
>>> len( operation ) # # number of files in operation
1
>>> request[0].Type # # __getitem__, there is also __setitem__ and __delitem__␣
↪defined for Request and Operation
'ReplicateAndRegister'
>>> operation in request # # __contains__ in Request
True
>>> opFile in operation # # __contains__ in Operation
True
```

Once the request is ready, you can insert it to the *ReqDB*:

```
>>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
>>> rc = ReqClient() # # create client
>>> rc.putRequest( request ) # # put request to ReqDB
```

Warning: Even though it is tempting, you cannot reuse a File object in multiple Operations, even if they are the same LFN. After all, they are different entries in the DB...

Read

Reading request back can be done using two methods defined in the *ReqClient*:

- for reading:

```
>>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
>>> rc = ReqClient() # # create client
>>> rc.peekRequest( "foobarbaz" ) # # get request from ReqDB for reading
```

- for execution (request status on DB side will flip to 'Assigned'):

```
>>> from DIRAC.RequestManagementSystem.Client.ReqClient import ReqClient
>>> rc = ReqClient() # # create client
>>> rc.getRequest( "foobarbaz" ) # # 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:

```
>>> del request[0] # # remove 1st operation using __delitem__
>>> request[0] = Operation() # # overwrite 1st operation using __setitem__
>>> request.addOperation( Operation() ) # # add new operation
>>> request.insertBefore( Operation(), request[0] ) # # insert new operation at head
>>> request.insertAfter( Operation(), request[0] ) # # insert new opration after 1st
```

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

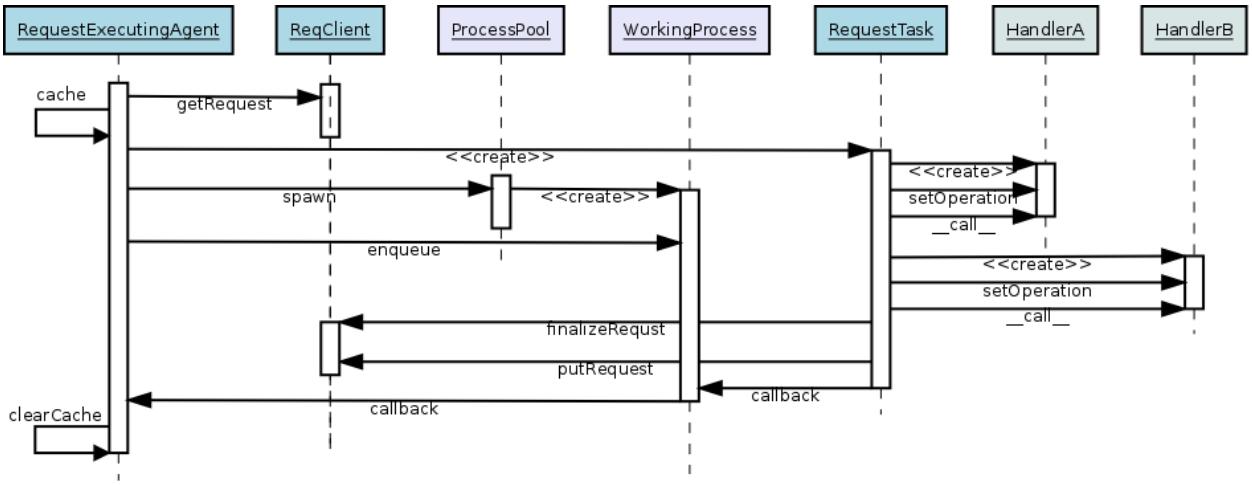
The validation of a new Request that is about to enter the system for execution is checked by the *RequestValidator* helper class - a gatekeeper checking if request is properly defined. The *validator* is blocking insertion of a new record to the *ReqDB* in case of missing or malformed attributes and 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"
>>> gRequestValidator.validate( invalid )
{'Message': "Operation #0 of type 'ForwardDISET' is missing Arguments attribute.", 'OK
↪': False}
```

A word of caution has to be clearly stated over here: the 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. The agent will try to execute request as a whole in one go.



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*
- The Operation executions are attempted several times, and the delay between retry increments

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*: Remove files from an SE
  - *PutAndRegister*: Upload local files to an SE and register it
  - *RegisterFile*: Register files
  - *RemoveFile*: Remove files from all SEs and the catalogs
  - *RemoveReplica*: Remove replicas from an SE and the catalog
  - *ReplicateAndRegister*: Replicate a file to an SE and register it

- RequestManagement (under RMS/Agent/RequestOperation)
  - *ForwardDISET*: Asynchronous execution of DISET call

Note that all the DataManagement operation support an extra parameter in their respective Handler sections: *TimeOutPerfile*. The timeout for the operation is then calculated from this value and the number of files in the Operation.

The *ReplicateAndRegister* section accepts extra attributes, specific to FTSTransfers:

- FTSMode (default False): if True, delegate transfers to FTS
- FTSBannedGroups: list of DIRAC group whose transfers should not go through FTS.

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__):

```
""" KillParrot operation handler """
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"
```

(continues on next page)

(continued from previous page)

```
      gMonitor.addMark( "ParrotsDead", 1)
    # # return S_OK/S_ERROR (always!!!)
    return S_OK()
```

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*:

```
>>> 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
```

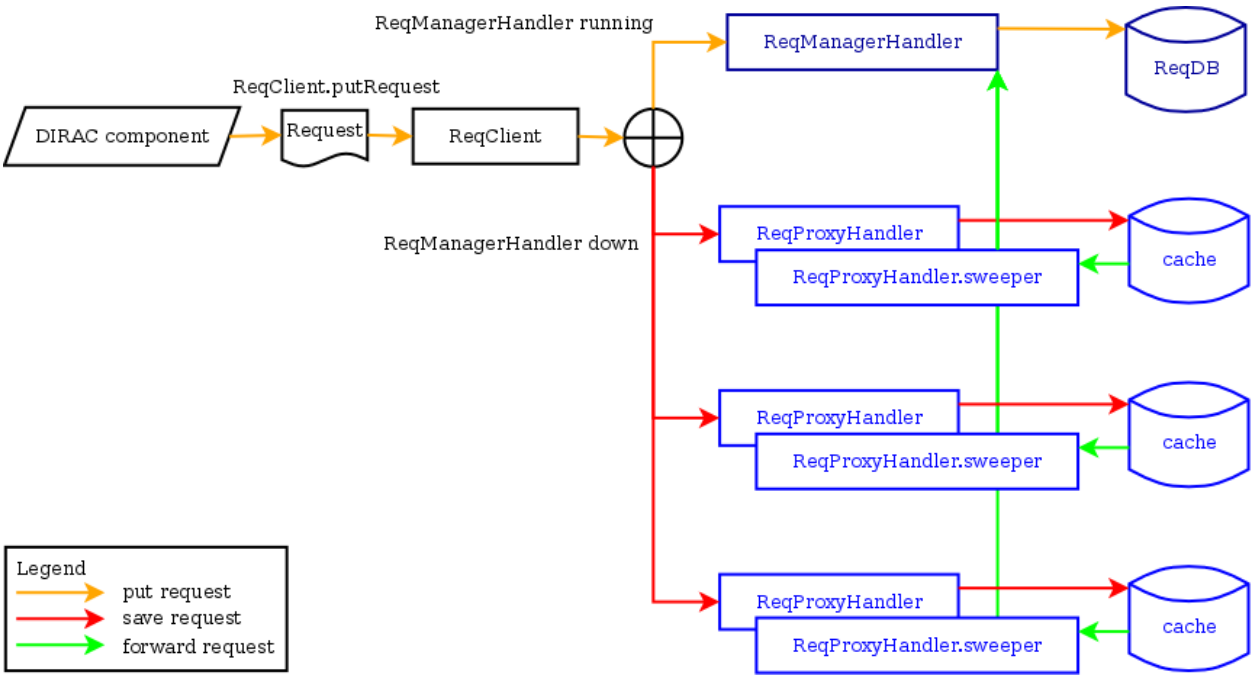
In principle, several *RequestExecutingAgent* can work in parallel, but be aware that their are race conditions that might lead to requests being executed multiple time.

3.11.5 ReqManager and ReqProxies

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 starts 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:

```
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/RequestManagement/RequestProxy
    }
  }
}
```

Don't forget to put correct FQDNs instead of <central>, <hostA>, <hostB> in above example!

3.12 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 enpoint is an *HTTPS* server provided in the *RESTDIRAC* module. This *HTTPS* server requires Tornado. If you don't have it installed just do:

```
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.

3.12.1 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 certificatea it has to *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 yo 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.

3.12.2 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:

```
{
  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:

```
>>> 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 separared list of possible values. An example would be:

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

3.13 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.

3.13.1 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
- installation instructions can be found: Eclipse wiki

Install ExtJS

- download from Sencha page and un-zip it. Note if you have installed WebAppDIRAC, you can found it under WebApp/static/extjs directory.

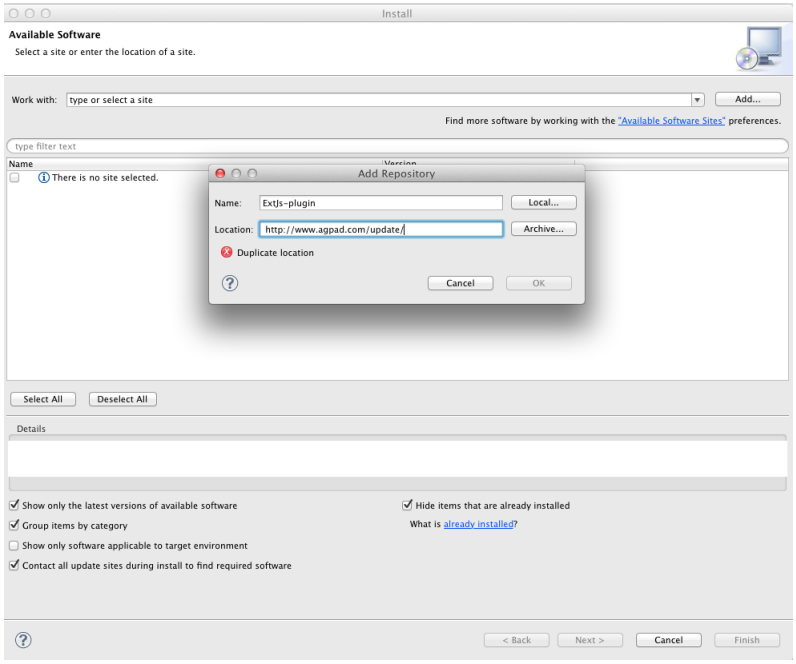
Eclipse and ExtJS

We used the DuckQuoc's blog to set up our Eclipse. There is an other 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 appears:

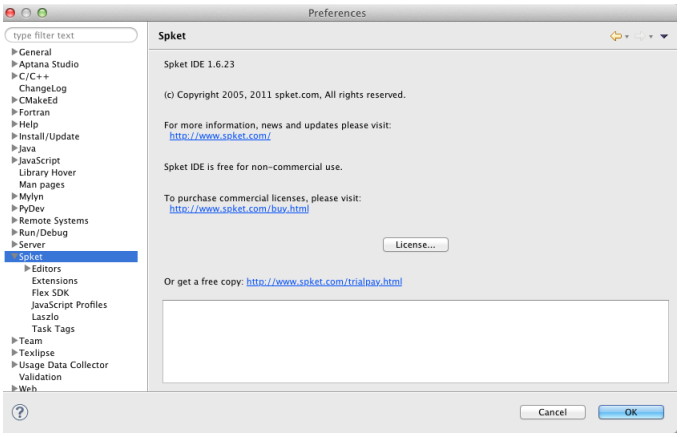


Please give a name and use the following link: http://www.agpad.com/update/

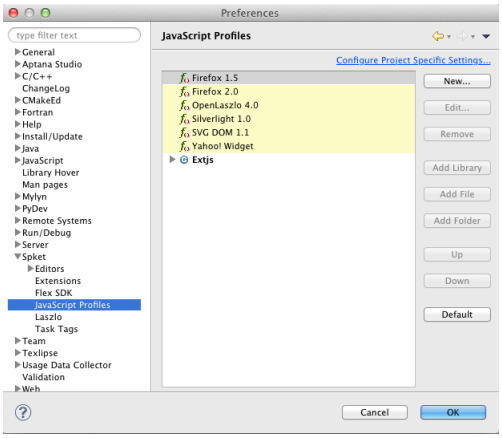
1. Click Ok -> select all components
2. Accept the term and conditions -> Finish
3. Wait till the package will be downloaded and installed in case of warning click OK.
4. Restart Eclipse (If it will not restart automatically)

Create Spket profile for ExtJs (Configuration panel):

- Click “Eclipse” -> “Preferences…” You will see the following configuration form:



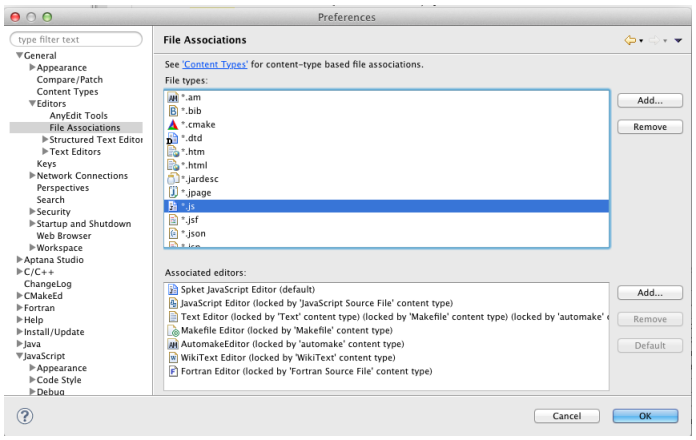
- select “Spket JavaScript Profile” and click to the New button and then type ExtJs.



- 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>‘ 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”



- Please select *.js and then select “Spket JavaScript Editor” and click on the “Default button”
- Restart Eclipse.

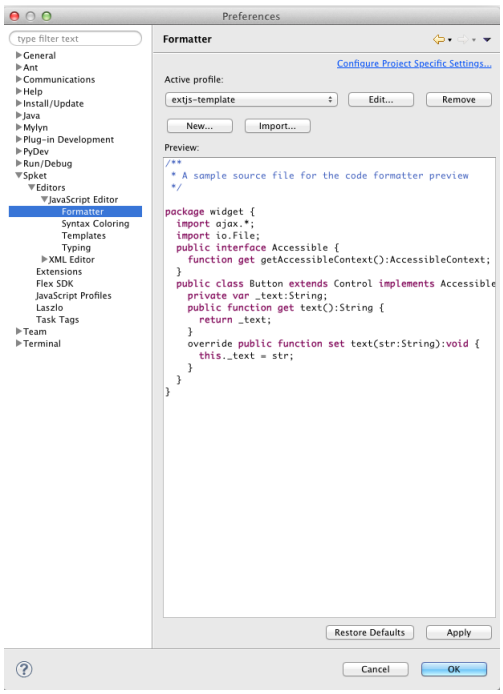
Auto-complete feature

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. 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 you java jdk. We tested with jdk6 and jdk7. We did not discovered any problem using those versions.

3.13.2 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:

```
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

3.13.3 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/). T here, 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**
      - **<Module name folder such as DIRAC, LHCbDIRAC, WebAppDIRAC>: contains the client side implen**

        Application 1

        Application 2
    - **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 extends **WebHandler** class which is the base class for all server side applications handling clients requests. The starting definition of this class is as follows:

```
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. Lets 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.:

```
from WebAppDIRAC.Lib.WebHandler import WebHandler

class MyAppHandler(WebHandler):
```

(continues on next page)

(continued from previous page)

```
   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 how 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"][0]
```

or in a full example:

```
def web_ping(self):
   pingValue = self.request.arguments["ping_val"][0]
   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 app**
  - **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 or**
    - 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:

```
Ext.define('DIRAC.MyApp.classes.MyApp', {
   extend : 'Ext.dirac.core.Module',
   requires :[]
});
```

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.
- **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):

```
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 provid**
  - **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 multichecking, 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 MyAppHa

- **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:

```
from WebAppDIRAC.Lib.WebHandler import WebHandler
from DIRAC.Core.DISET.RPCClient import RPCClient
import random


class MyAppHandler(WebHandler):
    """
            The main class inherits from WebHandler
    """
    """
            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**
      - build
      - classes
      - css
      - images

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:

```
Ext.define('DIRAC.MyApp.classes.MyApp', {
   extend : 'Ext.dirac.core.Module',
   requires :[],
   initComponent:function(){},
   buildUI:function(){}
});
```

As explained before, first we have to be implement the **initComponent** and the **buildUI** methods.:

```
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
```

(continues on next page)

(continued from previous page)

```
    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({
```

(continues on next page)

(continued from previous page)

```
        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);
                    }
            });

        },
        scope : me
    });

    // creating a button with a click handler
    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]
```

(continues on next page)

(continued from previous page)

```
    });

    /*
            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'
            },
            timeout : 1800000
        },
        fields : [{
                    name : 'value',
                    type : 'int'
         }],
        autoLoad : true,
        pageSize : 50,

    });

    /*
            Creating the grid object.
            Pay attention that the region config property is set up to center
            which means that the grid will take the rest of the available␣
↪area.
            Also we set the store config property to refer to the store␣
↪object
            we created previously.
    */
    me.grid = Ext.create('Ext.grid.Panel', {
        region : 'center',
        store : me.dataStore,
        header : false,
        columns : [{
            header : 'Value',
            sortable : true,
            dataIndex : 'value',
            align : 'left'
        }]
    });

    /*
            Embedding the panel and the grid within the working area of the␣
↪application
    */
    me.add([me.leftPanel,me.grid]);
```

(continues on next page)

(continued from previous page)

```
}
```

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:

```
requires:   ['Ext.panel.Panel', 'Ext.form.field.TextArea', 'Ext.Button',
↪'Ext.toolbar.Toolbar', 'Ext.data.JsonStore', 'Ext.grid.Panel']
```

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:

```
WebApp
{
  DevelopMode = True
  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
  }
}
```

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, a 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
```

(continues on next page)

(continued from previous page)

```
    {
      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 compiled 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:

```
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:

```
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:

```
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

DIRAC framework provides already implemented widgets which can be found under (https://github.com/DIRACGrid/WebAppDIRAC/tree/integration/WebApp/static/core/js/utils). More details about the widgets can be found in the developer documentation: https://localhost:8443/DIRAC/static/doc/index.html or in the portal (https://hostname/DIRAC/static/doc/index.html).

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
```

3.14 How DIRAC works underneath

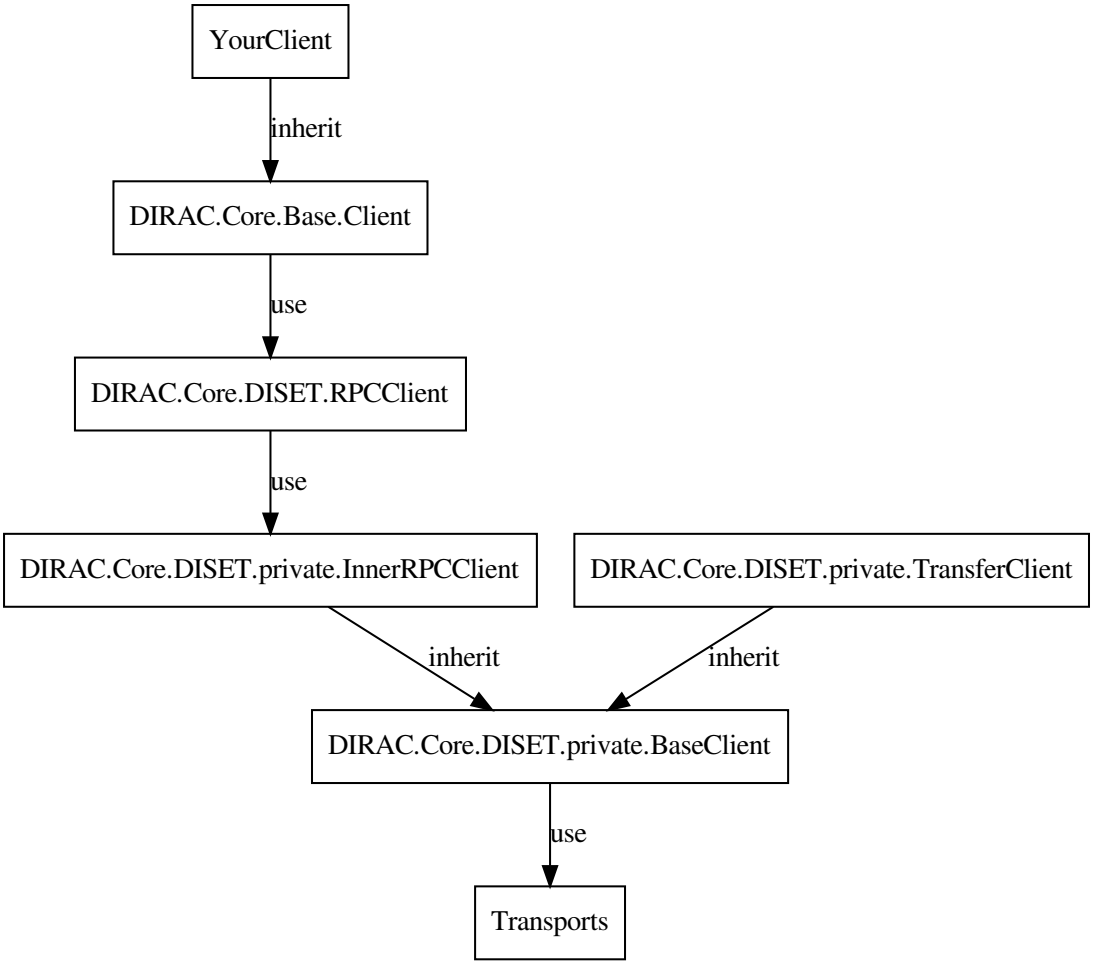
3.14.1 DIRAC Internals Core documentation

Documentation about the low level behavior of DIRAC

Client Service Interactions

Clients

The client can interact with services using RPC calls. DIRAC provides an abstraction which allows to easily add new clients.



The BaseClient class contains the low level logic to discover the URLs from System/Component, the connection retry and failover mechanism, the discovery of the credentials to use, and the mechanism to initialize actions with the server. It relies on a Transport class to actually perform the communication.

InnerRPCClient inherits from BaseClient, and just contain the logic to perform an RPC call: it proposes to the server an RPC action. TransferClient implements the FileTransfer logic.

RPCClient translates non existing methods into RPC calls. It does this by using an InnerRPCClient.

the Client class contains a similar logic to emulate methods, but parses specific arguments at each call (url, rpc and timeout) to instantiate an RPCClient. All parameters given to it at initialization will be passed to RPCClient, and propagated down to BaseClient.

Refer to the code documentation of each class for more details. It is good to know however that many connection details can be specified at the creation of the client, and are not necessarily taken from the environment, like the proxy to use.

Here is a rough overview of what is happening when you are calling a method from a client.

```
from DIRAC.Core.Base.Cliet import Client

c = Client()
c.serverURL('DataManagement/FileCatalog') # The subclient would have to define it␣
↪themselves as well

# This returns a function pointing to Client.executeRPC
pingMethod = c.ping
# Calling <class 'DIRAC.Core.Base.Client.Client'>.__getattr__(('ping',))

# When performing the executiong, the whole chain happens
pingMethod()


# Calling <class 'DIRAC.Core.Base.Client.Client'>.executeRPC(): this parses the␣
↪specific arguments URL and  timeout
#   if given in the call parameters
# Calling <class 'DIRAC.Core.Base.Client.Client'>._getRPC(False, '', 120): we␣
↪generate an RPCClient
# Calling <class 'DIRAC.Core.DISET.RPCClient.RPCClient'>.__getattr__('ping'): we␣
↪forward the call to the RPCClient
# Calling <class 'DIRAC.Core.DISET.RPCClient.RPCClient'>.__doRPC('ping',()): the␣
↪RPCClient emulates the existance of the
#   function and forwards it to the InnerRPCClient
# Calling <class 'DIRAC.Core.DISET.private.InnerRPCClient.InnerRPCClient'>.executeRPC(
↪'ping', ()): the RPC call is finally
#   executed
```

ThreadConfig

This special class is to be used in case you want to execute an operation on behalf of somebody else. Typically the WebApp uses it. This object is a singleton, but all its attributes are thread local. The host/identity wanting to use that requires the TrustedHost property.

Let's take an example

```
from DIRAC.Core.DISET.RPCClient import RPCClient
rpc = RPCClient('System/Component')

rpc.ping()
```

In the previous code, the code will be executed as whatever is set in the environment: host certificate or proxy.

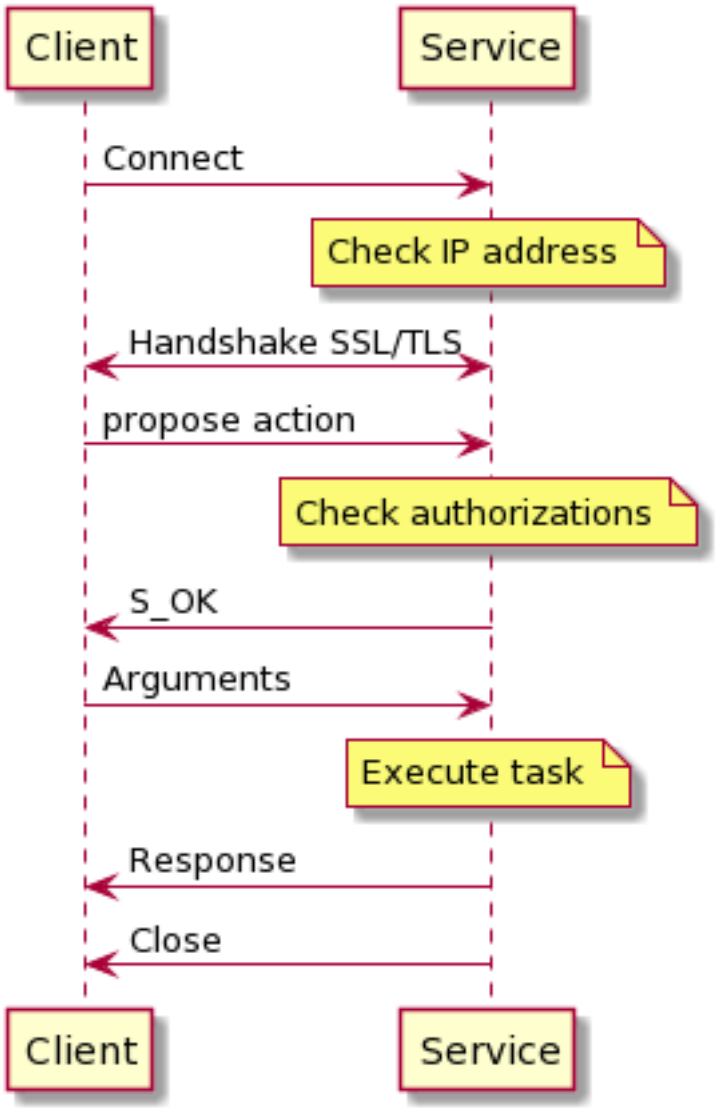
```
from DIRAC.Core.DISET.RPCClient import RPCClient
from DIRAC.Core.DISET.ThreadConfig import ThreadConfig

thConfig = ThreadConfig()
thConfig.setDN('/Whatever/User')

rpc = RPCClient('System/Component')
rpc.ping()
```

In that case, the call will still be performed using whatever is set in the environment, however the remote service will act as if the request was done by `/Whatever/user` (providing that the TrustedHost property is granted). And because of the `threading.local` inheritance, we can have separate users actions like below.

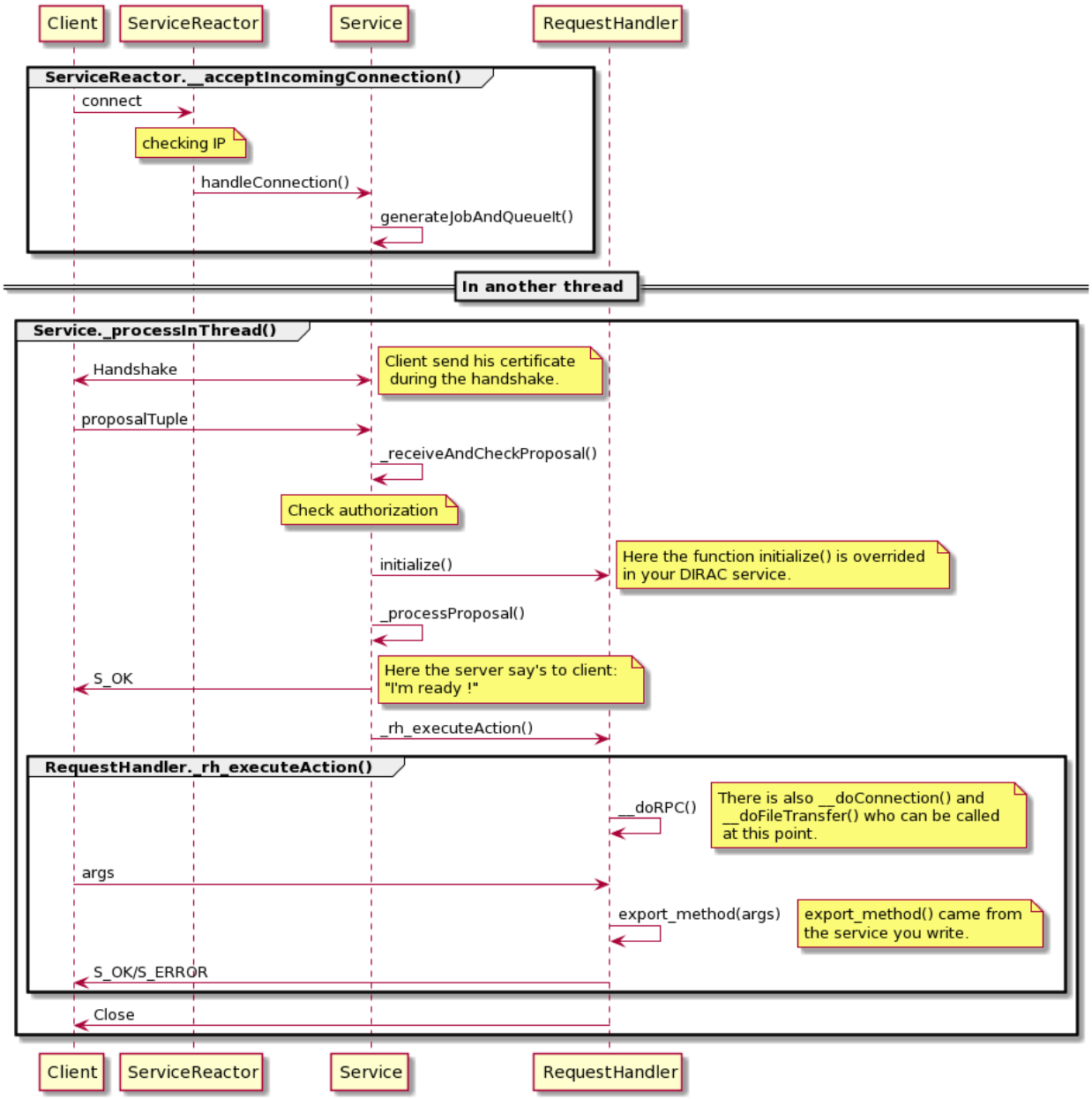
```
import threading
from DIRAC.Core.DISET.RPCClient import RPCClient
from DIRAC.Core.DISET.ThreadConfig import ThreadConfig

thConfig = ThreadConfig()

class myThread (threading.Thread):

  def __init__(self, name):
     super(myThread, self).__init__()
     self.name = name

  def run(self):
     thConfig.setDN(self.name)
     rpc = RPCClient('DataManagement/FileCatalog')
     rpc.ping()


threads = []

thread1 = myThread("/Whatever/user1")
thread2 = myThread("/Whatever/user2")

thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
   t.join()
```

Service

Here a simplified sequence diagram of client-server communication.



In most of the cases, a RPC call follows this diagram. Before starting anything, the service checks the IP. Then the client sends his certificate during the handshake and right after he sends the remote procedure who need to be called. The service checks the authorization and sends a signal to client when ready. The client sends all arguments that the service needs and finally the service execute its task. Below you can find a more complete diagram. Before calling the request handler, if the IP is banned, the service closes the connection. For other steps if an error occurred the service sends S_ERROR before closing connection.



Complete path of packages are not on the diagram for readability:

- serviceReactor: `DIRAC.Core.DISET.ServiceReactor`
- service: `DIRAC.Core.DISET.private.Service`
- requestHandler: `DIRAC.Core.DISET.RequestHandler`

You can see that the client sends a proposalTuple, proposalTuple contain (service, setup, ClientVO) then (typeOfCall, method) and finaly extra-credentials. e.g:

```
(('Framework/serviceName', 'DeveloperSetup', 'unknown'), ('RPC', 'methodName'), '')
```

You have to notice that the service can call __doFileTransfer() but functions relative to file transfer are not implemented and always return S_ERROR. If needed you can implement these functions by overwriting methods from `DIRAC.Core.DISET.RequestHandler` in your service. Here the methods you have to overwrite:

```
def transfer_fromClient(self, fileId, token, fileSize, fileHelper):
def transfer_toClient(self, fileId, token, fileHelper):
def transfer_bulkFromClient(self, bulkId, token, bulkSize, fileHelper):
def transfer_bulkToClient(self, bulkId, token, fileHelper):
def transfer_listBulk(self, bulkId, token, fileHelper):
```

Client must send ('FileTransfer', direction) instead of ('RPC', method), direction can be "fromClient", "toClient", "bulkFromClient", "bulkToClient" or "listBulk".

Serialization

The serialization mechanism currently used in DIRAC is called DEncode. It is a custom serialization mechanism.

The aim in the medium term is to replace it with standard JSON serialization.

We will describe here these two.

DEncode

DEncode (`DEncode`) contains two functions:

- encode: returns the string representation of the input.
- decode: returns the decoded information from the string input, as well as the length of the information decoded.

```
from DIRAC.Core.Utilities.DEncode import encode, decode

myData = {'a' : [1,2,3], 2 : 'toto' }

# Encode the structure
myEncodedData = encode(myData)

# myEncodedData is the string 'di2es4:totos1:ali1ei2ei3eee'

# Decode the data back
decode(myEncodedData)

# returns a tuple containing the decoded data
# and the length decoded
# ({2: 'toto', 'a': [1, 2, 3]}, 27)
```

DEncode supports the following type:

- boolean
- datetime
- dict
- int
- float (CAUTION, see bellow)
- list
- long
- none
- string

- tuple
- unicode

It is a know fact that DEncode is not stable for floats:

```
from DIRAC.Core.Utilities.DEncode import encode, decode

d = 133143986190.0

import sys

sys.maxint > d
# True

encode(d)
# 'f1.3314398619e+11e'

decode(encode(d))
# (133143986190.00002, 18)
```

Notice that 133143986190.0 != 133143986190.00002

JEncode

Warning: This serialization is not in use yet

JEncode (`JEncode`) is based on JSON, but exposes the same interface as DEncode, that is an *encode* and a *decode* functions.

However, because of the nature of JSON (https://tools.ietf.org/html/rfc7159), there are some limitations and changes with respect to DEncode:

- all UTF-8 by default. Non default would be other UTF encoding
- Tuples are converted to arrays
- the keys of dictionaries are always strings. This means that any other type of key will be cast to a string (including numbers !). As a consequence, it is up to the sender/receiver to cast that in whatever type is desired.

JEncode contains a special serializer and deserializer which enhance the default one with:

- Support for datetime: the serialization format is hardcoded and corresponds to *%Y-%m-%d %H:%M:%S* (see https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior). This means that milliseconds are not kept. Note as well that only dates starting after 01-01-1900 are serializable.
- Support for custom object serialization inheriting from *JSerializable* (`JSerializable`). See the Code documentation for more details on the restrinctions and how to use it.

3.14.2 Components authentication and authorization

DIRAC components (services, agents and executors) by default use the certificate of the host onto which they run for authentication and authorization purposes.

Components can be instructed to use a "shifter proxy" for authN and authZ of their service calls. A shifter proxy is proxy certificate, which should be:

- specified in the "Operations/<setup>/Shifter" section of the CS
- uploaded to the ProxyManager (i.e. using "–upload" option of dirac-proxy-init)

Within an agent, in the "initialize" method, we can specify:

```
self.am_setOption('shifterProxy', 'DataManager')
```

when used, the requested shifter's proxy will be added in the environment of the agent with simply:

```
os.environ['X509_USER_PROXY'] = proxyDict['proxyFile']
```

and nothing else.

Which means that, still, each and every agent or service or executors by default will use the server certificate because, e.g. in dirac-agent.py script we have:

```
localCfg.addDefaultEntry("/DIRAC/Security/UseServerCertificate", "yes")
```

Which means that, if no further options are specified, all the calls to services OUTSIDE of DIRAC will use the proxy in os.environ['X509_USER_PROXY'], while for all internal communications the server certificate will be used.

If you want to use proxy certificate inside an agent for ALL service calls (inside AND outside of DIRAC) add:

```
gConfigurationData.setOptionInCFG('/DIRAC/Security/UseServerCertificate', 'false')
```

in the initialize or in the execute (or use a CS option in the local .cfg file)

Two decorators are available for safely doing all that:

- `executeWithoutServerCertificate()`
- `executeWithUserProxy()`

3.14.3 Authentication and authorization

When a client calls a service, he needs to be identified. If a client opens a connection a `BaseTransport` object is created then the service use the handshake to read certificates, extract informations and store them in a dictionary so you can use these informations easily. Here an example of possible dictionary:

```
{
        'DN': '/C=ch/O=DIRAC/[...]',
        'group': 'devGroup',
        'CN': u'ciuser',
        'x509Chain': <X509Chain 2 certs [...][...]>,
        'isLimitedProxy': False,
        'isProxy': True
}
```

When connection is opened and handshake is done, the service calls the `AuthManager` and gave him this dictionary in argument to check the authorizations. More generally you can get this dictionary with `BaseTransport.getConnectingCredentials`.

AuthManager

AuthManager.authQuery() returns boolean so it is easy to use, you just have to provide a method you want to call, and credDic. It's easy to use but you have to instantiate correctly the AuthManager. For initialization you need the complete path of your service, to get it you may use the PathFinder:

```
from DIRAC.ConfigurationSystem.Client import PathFinder
from DIRAC.Core.DISET.AuthManager import AuthManager
authManager = AuthManager( "%s/Authorization" % PathFinder.getServiceSection(
↪"Framework/someService") )
authManager.authQuery( csAuthPath, credDict, hardcodedMethodAuth ) #return boolean
# csAuthPath is the name of method for RPC or 'typeOfCall/method'
# credDict came from BaseTransport.getConnectingCredentials()
# hardcodedMethodAuth is optional
```

To determine if a query can be authorized or not the AuthManager extract valid properties for a given method. First AuthManager try to get it from gConfig, then try to get it from hardcoded list (hardcodedMethodAuth) in your service and if nothing was found get default properties from gConfig.

AuthManager also extract properties from user with credential dictionary and configuration system to check if properties matches. So you don't have to extract properties by yourself, but if needed you can use `DIRAC.Core.Security.CS.getPropertiesForGroup()`

3.15 DIRAC JobWrapper

The JobAgent is creating a file that is made from the JobWrapperTemplate.py file. It creates a temporary file using this file as a template, which becomes Wrapper_<jobID> somewhere in the workDirectory. It is this file that is then submitted as a real "job wrapper script".

The JobWrapper is not a job wrapper, but is an object that is used by the job wrapper (i.e. the JobWrapperTemplate's execute() method) to actually do the work.

The only change made in the "template" file is the following: wrapperTemplate = wrapperTemplate.replace("@SITEPYTHON@", str(siteRoot))

Then the file is submitted in bash using the defined CE (the InProcessCE in the default case)

The sequence executed is ("job" is the JobWrapper object here ;-)):

```
job.initialize( arguments )
#[...]
result = job.transferInputSandbox( arguments['Job']['InputSandbox'] )
#[...]
result = job.resolveInputData()
#[...]
result = job.execute( arguments )
#[...]
result = job.processJobOutputs( arguments )
#[...]
return job.finalize( arguments )
```

The watchdog is started in job.execute(). A direct consequence is that the time taken to download the input files is not taken into account for the WallClock time.

A race condition might happen inside this method. The problem here is that we submit the process in detached mode (or in a thread, not clear as here thread may be used for process), wait 10 seconds and expect it to be started. If this fails, the JobWrapperTemplate gives up, but if however the detached process runs, it continues executing as if nothing happened! It is there that there is the famous gJobReport.setJobStatus('Failed', 'Exception During Execution', sendFlag = False) which is sometimes causing jobs to go to "Failed" and then continue.

There is a nice "feature" of this complex cascade which is that the jobAgent reports "Job submitted as ..." (meaning the job was submitted to the local CE, i.e. the InProcessCE in our case) _after_ the "job" is actually executed!!!

The JobWrapper can also interpret error codes from the application itself. An error code is, for example, the DErrno.WMSRESC (1502) error code, which will instruct the JobWrapperTemplate to reschedule the current job.

CHAPTER 4

Documentation sources

User Guide Everything users need to know, including client installations	*Developer Guide* Adding new functionality to DIRAC
Administrator Guide Administration of the DIRAC services (server installations)	CodeDocumentation/index Code reference

CHAPTER 5

Indices and tables

- genindex
- search