autosubmit Documentation

Release 3.9.1

Domingo Manubens - Joan López

Jul 01, 2021

Contents

1 Introduction **1**
- 1.1 What is Autosubmit ? . . . 1
- 1.2 Why is Autosubmit needed ? . . . 1
- 1.3 How does Autosubmit work ? . . . 2

2 Tutorial **5**
- 2.1 Quick start guide . . . 5

3 Installation **17**
- 3.1 How to install . . . 17
- 3.2 How to configure . . . 18

4 Usage **21**
- 4.1 Command list . . . 21
- 4.2 Tutorials (How to) . . . 22

5 Defining the workflow **23**
- 5.1 Simple workflow . . . 23
- 5.2 Running jobs once per startdate, member or chunk . . . 23
- 5.3 Dependencies . . . 24
- 5.4 Job frequency . . . 26
- 5.5 Job synchronize . . . 27
- 5.6 Job split . . . 29
- 5.7 Job delay . . . 30
- 5.8 Rerun dependencies . . . 31

6 Frequent Questions and Answers **33**

7 Troubleshooting **35**
- 7.1 How to change the job status stopping autosubmit . . . 35
- 7.2 How to change the job status without stopping autosubmit . . . 35
- 7.3 My project parameters are not being substituted in the templates. . . . 35
- 7.4 Unable to recover remote logs files. . . . 35
- 7.5 Other possible errors . . . 35

8 Error codes and solutions **37**
- 8.1 Experiment Locked - Critical Error 7000 . . . 37

8.2 Database Issues - Critical Error codes [7001-7005] . . . 37
8.3 Wrong User Input - Critical Error codes [7010-7030] . . . 38
8.4 Platform issues - Critical Error codes. Local [7040-7050] and remote [7050-7060] . . . 38
8.5 Uncatalogued codes - Critical Error codes [7060+] . . . 39
8.6 Minor errors - Error codes [6000+] . . . 40

9 Developing a project **41**

10 Variables reference **43**
10.1 Job variables . . . 43
10.2 Platform variables . . . 44
10.3 Project variables . . . 45
10.4 Performance Metrics . . . 45

11 Module documentation **47**
11.1 autosubmit . . . 47
11.2 autosubmit.config . . . 53
11.3 autosubmit.database . . . 63
11.4 autosubmit.git . . . 64
11.5 autosubmit.job . . . 65
11.6 autosubmit.monitor . . . 76
11.7 autosubmit.platform . . . 77

12 Autosubmit GUI **87**
12.1 Autosubmit GUI Main Page . . . 87

Python Module Index **103**

Index **105**

CHAPTER 1

Introduction

1.1 What is Autosubmit ?

```
Autosubmit is a python-based workflow manager to create, manage and monitor␣
↪experiments by using Computing Clusters, HPC's and Supercomputers remotely via ssh.␣
↪It has support for experiments running in more than one HPC and for different␣
↪workflow configurations.
Autosubmit is currently used at Barcelona Supercomputing Centre (BSC) to run models␣
↪(EC-Earth, MONARCH, NEMO, CALIOPE, HERMES...), operational toolchains (S2S4E), data-
↪download workflows (ECMWF MARS), and many other.
Autosubmit has been used to manage models running at supercomputers in BSC, ECMWF,␣
↪IC3, CESGA, EPCC, PDC and OLCF.
Autosubmit is now available via PyPi package under the terms of GNU General Public␣
↪License.
```

```
Get involved or contact us:
```

Autosubmit GitLab:	https://earth.bsc.es/gitlab/es/autosubmit
Autosubmit Mailing List:	autosubmit@bsc.es

1.2 Why is Autosubmit needed ?

Autosubmit is the only existing tool that satisfies the following requirements from the weather and climate community:

- **Automatisation** Job submission to machines and dependencies between jobs are managed by Autosubmit. No user intervention is needed.
- **Data provenance** Assigns unique identifiers for each experiment and stores information about model version, experiment configuration and computing facilities used in the whole process.
- **Failure tolerance** Automatic retrials and ability to rerun chunks in case of corrupted or missing data.

- **Resource management** Autosubmit manages supercomputer particularities, allowing users to run their experiments in the available machine without having to adapt the code. Autosubmit also allows to submit tasks from the same experiment to different platforms.

1.3 How does Autosubmit work ?

You can find help about how to use autosubmit and a list of available commands, just executing:

```
autosubmit -h
```

Execute autosubmit <command> -h for detailed help for each command:

```
autosubmit expid -h
```

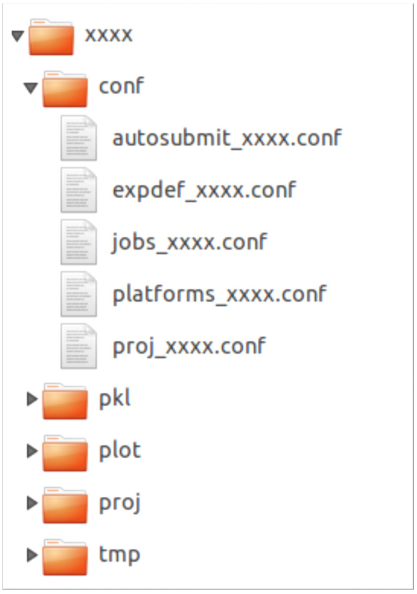
1.3.1 Experiment creation

To create a new experiment, run the command:

```
autosubmit expid -H HPCname -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

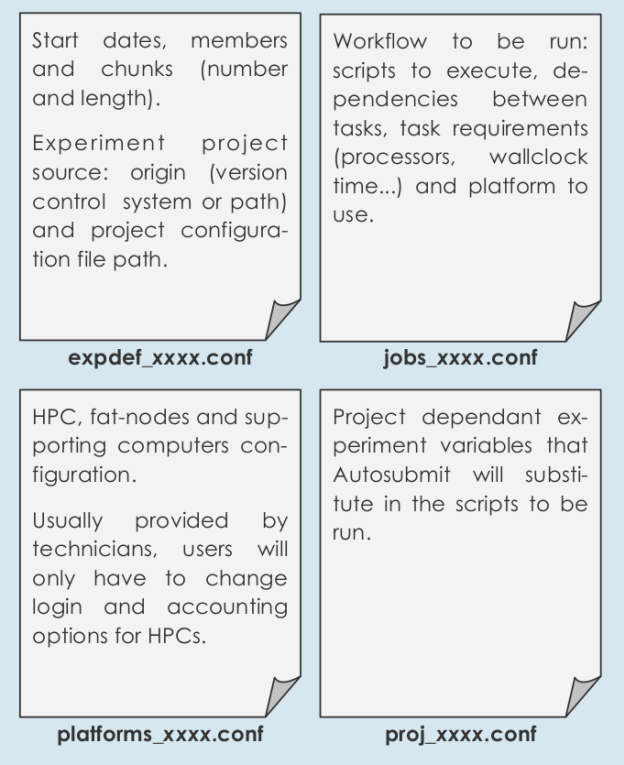
This command assigns a unique four character identifier (`xxxx`, names starting from a letter, the other three characters) to the experiment and creates a new folder in experiments repository with structure shown in Figure 1.1.



1.1: Example of an experiment directory tree.

1.3.2 Experiment configuration

To configure the experiment, edit `expdef_xxxx.conf`, `jobs_xxxx.conf` and `platforms_xxxx.conf` in the `conf` folder of the experiment (see contents in Figure 1.2).



1.2: Configuration files content

After that, you are expected to run the command:

```
autosubmit create xxxx
```

This command creates the experiment project in the `proj` folder. The experiment project contains the scripts specified in `jobs_xxxx.conf` and a copy of model source code and data specified in `expdef_xxxx.conf`.

1.3.3 Experiment run

To run the experiment, just execute the command:

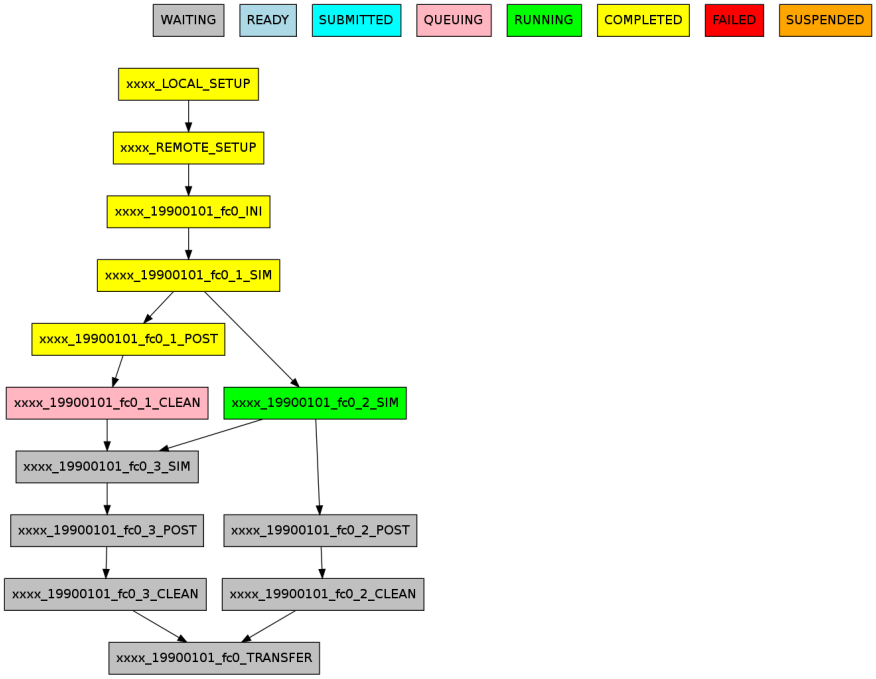
```
autosubmit run xxxx
```

Autosubmit will start submitting jobs to the relevant platforms (both HPC and supporting computers) by using the scripts specified in `jobs_xxxx.conf`. Autosubmit will substitute variables present on scripts where handlers appear in *%variable_name%* format. Autosubmit provides variables for *current chunk*, *start date*, *member*, *computer configuration* and more, and also will replace variables form `proj_xxxx.conf`.

To monitor the status of the experiment, the command:

```
autosubmit monitor xxxx
```

is available. This will plot the workflow of the experiment and the current status.



1.3: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

CHAPTER 2

Tutorial

2.1 Quick start guide

2.1.1 First Step: Experiment creation

To create a new experiment, run the command:

```
autosubmit expid -H HPCname -d Description
```

HPCname is the name of the main HPC platform for the experiment: it will be the default platform for the tasks. *Description* is a brief experiment description.

This command assigns a unique four character identifier (`xxxx`, names starting from a letter, the other three characters) to the experiment and creates a new folder in experiments repository.

Examples:

```
autosubmit expid --HPC ithaca --description "experiment is about..."
```

Caution: The *HPCname*, e.g. ithaca, must be defined in the platforms configuration. See next section *Second Step: Experiment configuration*.

```
autosubmit expid --copy a000 --HPC ithaca -d "experiment is about..."
```

Warning: You can only copy experiments created with Autosubmit 3.0 or above.

2.1.2 Second Step: Experiment configuration

To configure the experiment, edit `expdef_cxxx.conf`, `jobs_cxxx.conf` and `platforms_cxxx.conf` in the `conf` folder of the experiment.

***expdef_cxxx.conf* contains:**

- Start dates, members and chunks (number and length).
- Experiment project source: origin (version control system or path)
- Project configuration file path.

***jobs_cxxx.conf* contains the workflow to be run:**

- Scripts to execute.
- Dependencies between tasks.
- Task requirements (processors, wallclock time...).
- Platform to use.

***platforms_cxxx.conf* contains:**

- HPC, fat-nodes and supporting computers configuration.

Note: *platforms_cxxx.conf* is usually provided by technicians, users will only have to change login and accounting options for HPCs.

Note: There are multiple implementations of the communication with the platforms, so if you are interested in changing the default implementation, you can see how to do it on the 'Usage' section.

You may want to configure Autosubmit parameters for the experiment. Just edit `autosubmit_cxxx.conf`.

***autosubmit_cxxx.conf* contains:**

- Maximum number of jobs to be waiting in the HPC queue.
- Maximum number of jobs to be running at the same time at the HPC.
- Time (seconds) between connections to the HPC queue scheduler to poll already submitted jobs status.
- Number of retrials if a job fails.
- Default output type for some Autosubmit functions

Examples:

```
vi <experiments_directory>/cxxx/conf/expdef_cxxx.conf
```

```
[DEFAULT]
# Experiment identifier
# No need to change
EXPID = cxxx
# HPC name.
# No need to change
HPCARCH = ithaca

[experiment]
```

(continues on next page)

(continued from previous page)

```
# Supply the list of start dates. Available formats: YYYYMMDD YYYYMMDDhh YYYYMMDDhhmm
# Also you can use an abreviated sintax for multiple dates with common parts:
# 200001[01 15] <=> 20000101 20000115
# DATELIST = 19600101 19650101 19700101
# DATELIST = 1960[0101 0201 0301]
DATELIST = 19900101
# Supply the list of members. LIST = fc0 fc1 fc2 fc3 fc4
MEMBERS = fc0
# Chunk size unit. STRING = hour, day, month, year
CHUNKSIZEUNIT = month
# Chunk size. NUMERIC = 4, 6, 12
CHUNKSIZE = 1
# Total number of chunks in experiment. NUMERIC = 30, 15, 10
NUMCHUNKS = 2
# Calendar used. LIST: standard, noleap
CALENDAR = standard
# List of members that can be included in this run. Optional.
# RUN_ONLY_MEMBERS = fc0 fc1 fc2 fc3 fc4
# RUN_ONLY_MEMBERS = fc[0-4]
RUN_ONLY_MEMBERS =

[rerun]
# Is a rerun or not? [Default: Do set FALSE]. BOOLEAN = TRUE, FALSE
RERUN = FALSE
# If RERUN = TRUE then supply the list of chunks to rerun
# LIST = "[ 19601101 [ fc0 [1 2 3 4] fc1 [1] ] 19651101 [ fc0 [16-30] ] ]"
CHUNKLIST =

[project]
# Select project type. STRING = git, svn, local, none
# If PROJECT_TYPE is set to none, Autosubmit self-contained dummy templates will be␣
↪used
PROJECT_TYPE = git
# Destination folder name for project. type = STRING, default = leave empty,
PROJECT_DESTINATION = model

# If PROJECT_TYPE is not git, no need to change
[git]
# Repository URL  STRING = 'https://github.com/torvalds/linux.git'
PROJECT_ORIGIN = https://gitlab.cfu.local/cfu/auto-ecearth3.git
# Select branch or tag, STRING, default = 'master',
# help = {'master' (default), 'develop', 'v3.1b', ...}
PROJECT_BRANCH = develop
# type = STRING, default = leave empty, help = if model branch is a TAG leave empty
PROJECT_COMMIT =

# If PROJECT_TYPE is not svn, no need to change
[svn]
# type = STRING, help = 'https://svn.ec-earth.org/ecearth3'
PROJECT_URL =
# Select revision number. NUMERIC = 1778
PROJECT_REVISION =

# If PROJECT_TYPE is not local, no need to change
[local]
# type = STRING, help = /foo/bar/ecearth
PROJECT_PATH =
```

(continues on next page)

(continued from previous page)

```
# If PROJECT_TYPE is none, no need to change
[project_files]
# Where is PROJECT CONFIGURATION file location relative to project root path
FILE_PROJECT_CONF = templates/ecearth3/ecearth3.conf
# Where is JOBS CONFIGURATION file location relative to project root path
FILE_JOBS_CONF = templates/common/jobs.conf
```

```
vi <experiments_directory>/cxxx/conf/jobs_cxxx.conf
```

```
# Example job with all options specified

## Job name
# [JOBNAME]
## Script to execute. If not specified, job will be omited from workflow.
## Path relative to the project directory
# FILE =
## Platform to execute the job. If not specificied, defaults to HPCARCH in expedf␣
↪file.
## LOCAL is always defined and referes to current machine
# PLATFORM =
## Queue to add the job to. If not specificied, uses PLATFORM default.
# QUEUE =
## Defines dependencies from job as a list of parents jobs separed by spaces.
## Dependencies to jobs in previous chunk, member o startdate, use -(DISTANCE)
# DEPENDENCIES = INI SIM-1 CLEAN-2
## Define if jobs runs once, once per stardate, once per member or once per chunk.␣
↪Options: once, date, member, chunk.
## If not specified, defaults to once
# RUNNING = once
## Specifies that job has only to be run after X dates, members or chunk. A job will␣
↪always be created for the last
## If not specified, defaults to 1
# FREQUENCY = 3
## On a job with FREQUENCY > 1, if True, the dependencies are evaluated against all
## jobs in the frequency interval, otherwise only evaluate dependencies against␣
↪current
## iteration.
## If not specified, defaults to True
# WAIT = False
## Defines if job is only to be executed in reruns. If not specified, defaults to␣
↪false.
# RERUN_ONLY = False
## Defines jobs needed to be rerun if this job is going to be rerun
# RERUN_DEPENDENCIES = RERUN INI LOCAL_SETUP REMOTE_SETUP TRANSFER
## Wallclock to be submitted to the HPC queue in format HH:MM
# WALLCLOCK = 00:05
## Processors number to be submitted to the HPC. If not specified, defaults to 1.
## Wallclock chunk increase (WALLCLOCK will be increased according to the formula␣
↪WALLCLOCK + WCHUNKINC * (chunk - 1)).
## Ideal for sequences of jobs that change their expected running time according to␣
↪the current chunk.
# WCHUNKINC = 00:01
# PROCESSORS = 1
## Threads number to be submitted to the HPC. If not specified, defaults to 1.
# THREADS = 1
```

(continues on next page)

(continued from previous page)

```
## Tasks number to be submitted to the HPC. If not specified, defaults to empty.
# TASKS =
## Memory requirements for the job in MB
# MEMORY = 4096
##  Number of retrials if a job fails. If not specified, defaults to the value given␣
↪on experiment's autosubmit.conf
# RETRIALS = 4
## Some jobs can not be checked before running previous jobs. Set this option to␣
↪false if that is the case
# CHECK = False
## Select the interpreter that will run the job. Options: bash, python, r Default:␣
↪bash
# TYPE = bash
## Specify the path to the interpreter. If empty, use system default based on job␣
↪type  . Default: empty
# EXECUTABLE = /my_python_env/python3


[LOCAL_SETUP]
FILE = LOCAL_SETUP.sh
PLATFORM = LOCAL

[REMOTE_SETUP]
FILE = REMOTE_SETUP.sh
DEPENDENCIES = LOCAL_SETUP
WALLCLOCK = 00:05

[INI]
FILE = INI.sh
DEPENDENCIES = REMOTE_SETUP
RUNNING = member
WALLCLOCK = 00:05

[SIM]
FILE = SIM.sh
DEPENDENCIES = INI SIM-1 CLEAN-2
RUNNING = chunk
WALLCLOCK = 00:05
PROCESSORS = 2
THREADS = 1
TASKS = 1

[POST]
FILE = POST.sh
DEPENDENCIES = SIM
RUNNING = chunk
WALLCLOCK = 00:05

[CLEAN]
FILE = CLEAN.sh
DEPENDENCIES = POST
RUNNING = chunk
WALLCLOCK = 00:05

[TRANSFER]
FILE = TRANSFER.sh
PLATFORM = LOCAL
```

(continues on next page)

(continued from previous page)

```
DEPENDENCIES = CLEAN
RUNNING = member
```

```
vi <experiments_directory>/cxxx/conf/platforms_cxxx.conf
```

```
# Example platform with all options specified

## Platform name
# [PLAFORM]
## Queue type. Options: PBS, SGE, PS, LSF, ecaccess, SLURM
# TYPE =
## Version of queue manager to use. Needed only in PBS (options: 10, 11, 12) and␣
↪ecaccess (options: pbs, loadleveler)
# VERSION =
## Hostname of the HPC
# HOST =
## Project for the machine scheduler
# PROJECT =
## Budget account for the machine scheduler. If omitted, takes the value defined in␣
↪PROJECT
# BUDGET =
## Option to add project name to host. This is required for some HPCs
# ADD_PROJECT_TO_HOST = False
## User for the machine scheduler
# USER =
## Path to the scratch directory for the machine
# SCRATCH_DIR = /scratch
## If true, autosubmit test command can use this queue as a main queue. Defaults to␣
↪false
# TEST_SUITE = False
## If given, autosubmit will add jobs to the given queue
# QUEUE =
## If specified, autosubmit will run jobs with only one processor in the specified␣
↪platform.
# SERIAL_PLATFORM = SERIAL_PLATFORM_NAME
## If specified, autosubmit will run jobs with only one processor in the specified␣
↪queue.
## Autosubmit will ignore this configuration if SERIAL_PLATFORM is provided
# SERIAL_QUEUE = SERIAL_QUEUE_NAME
## Default number of processors per node to be used in jobs
# PROCESSORS_PER_NODE =
## Default Maximum number of jobs to be waiting in any platform queue
## Default = 3
# MAX_WAITING_JOBS = 3
## Default maximum number of jobs to be running at the same time at the platform.
## Applies at platform level. Considers QUEUEING + RUNNING jobs.
## Ideal for configurations where some remote platform has a low upper limit of␣
↪allowed jobs per user at the same time.
## Default = 6
# TOTAL_JOBS = 6

[ithaca]
# Queue type. Options: ps, SGE, LSF, SLURM, PBS, eceaccess
TYPE = SGE
HOST = ithaca
PROJECT = cfu
```

(continues on next page)

(continued from previous page)

```
ADD_PROJECT_TO_HOST = true
USER = dmanubens
SCRATCH_DIR = /scratch/cfu
TEST_SUITE = True
```

```
vi <experiments_directory>/cxxx/conf/autosubmit_cxxx.conf
```

```
[config]
# Experiment identifier
# No need to change
EXPID =
# No need to change.
# Autosubmit version identifier
AUTOSUBMIT_VERSION =
# Default maximum number of jobs to be waiting in any platform
# Default = 3
MAXWAITINGJOBS = 3
# Default maximum number of jobs to be running at the same time at any platform
# Can be set at platform level on the platform_cxxx.conf file
# Default = 6
TOTALJOBS = 6
# Time (seconds) between connections to the HPC queue scheduler to poll already␣
↪submitted jobs status
# Default = 10
SAFETYSLEEPTIME = 10
# Number of retrials if a job fails. Can ve override at job level
# Default = 0
RETRIALS = 0
# Default output type for CREATE, MONITOR, SET STATUS, RECOVERY. Available options:␣
↪pdf, svg, png, ps, txt
# Default = pdf
OUTPUT = pdf
```

Then, Autosubmit *create* command uses the `expdef_cxxx.conf` and generates the experiment:

```
autosubmit create cxxx
```

cxxx is the name of the experiment.

In the process of creating the new experiment a plot has been created.

It can be found in `<experiments_directory>/cxxx/plot/`

2.1.3 Third Step: Experiment run

After filling the experiment configuration and create, user can go into `proj` which has a copy of the model.

A short reference on how to prepare the experiment project is detailed in the following section of this documentation:

Developing a project

The experiment project contains the scripts specified in `jobs_xxxx.conf` and a copy of model source code and data specified in `expdef_xxxx.conf`.

To configure experiment project parameters for the experiment, edit `proj_cxxx.conf`.

***proj_cxxx.conf* contains:**

- The project dependant experiment variables that Autosubmit will substitute in the scripts to be run.

Warning: The `proj_xxxx.conf` has to be defined in INI style so it should has section headers. At least one.

Example:

```
vi <experiments_directory>/cxxx/conf/proj_cxxx.conf
```

```
[common]
# No need to change.
MODEL = ecearth
# No need to change.
VERSION = v3.1
# No need to change.
TEMPLATE_NAME = ecearth3
# Select the model output control class. STRING = Option
# listed under the section : https://earth.bsc.es/wiki/doku.php?id=overview_outclasses
OUTCLASS = specs
# After transferring output at /cfunas/exp remove a copy available at permanent␣
↪storage of HPC
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
MODEL_output_remove = TRUE
# Activate cmorization [Default: leave empty]. BOOLEAN = TRUE, FALSE
CMORIZATION = TRUE
# Essential if cmorization is activated.
# STRING =  (http://www.specs-fp7.eu/wiki/images/1/1c/SPECS_standard_output.pdf)
CMORFAMILY =
# Supply the name of the experiment associated (if there is any) otherwise leave it␣
↪empty.
# STRING (with space) = seasonal r1p1, seaiceinit r?p?
ASSOCIATED_EXPERIMENT =
# Essential if cmorization is activated (Forcing). STRING = Nat,Ant (Nat and Ant is a␣
↪single option)
FORCING =
# Essential if cmorization is activated (Initialization description). STRING = N/A
INIT_DESCR =
# Essential if cmorization is activated (Physics description). STRING = N/A
PHYS_DESCR =
# Essential if cmorization is activated (Associated model). STRING = N/A
ASSOC_MODEL =

[grid]
# AGCM grid resolution, horizontal (truncation T) and vertical (levels L).
# STRING = T159L62, T255L62, T255L91, T511L91, T799L62 (IFS)
IFS_resolution = T511L91
# OGCM grid resolution. STRING = ORCA1L46, ORCA1L75, ORCA025L46, ORCA025L75 (NEMO)
NEMO_resolution = ORCA025L75

[oasis]
# Coupler (OASIS) options.
OASIS3 = yes
# Number of pseduo-parallel cores for coupler [Default: Do set "7"]. NUMERIC = 1, 7,␣
↪10
OASIS_nproc = 7
# Handling the creation of coupling fields dynamically [Default: Do set "TRUE"].
# BOOLEAN = TRUE, FALSE
```

(continues on next page)

(continued from previous page)

```
OASIS_flds = TRUE

[ifs]
# Atmospheric initial conditions ready to be used.
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_
↪conditions:atmospheric
ATM_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
ATM_ini_member =
# Set timestep (in sec) w.r.t resolution.
# NUMERIC = 3600 (T159), 2700 (T255), 900 (T511), 720 (T799)
IFS_timestep = 900
# Number of parallel cores for AGCM component. NUMERIC = 28, 100
IFS_nproc = 640
# Coupling frequency (in hours) [Default: Do set "3"]. NUMERIC = 3, 6
RUN_coupFreq = 3
# Post-procssing frequency (in hours) [Default: Do set "6"]. NUMERIC = 3, 6
NFRP = 6
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
LCMIP5 = TRUE
# Choose RCP value [Default: Do set "2"]. NUMERIC = 0, 1=3-PD, 2=4.5, 3=6, 4=8.5
NRCP = 0
# [Default: Do set "TRUE"]. BOOLEAN = TRUE, FALSE
LHVOLCA = TRUE
# [Default: Do set "0"]. NUMERIC = 1850, 2005
NFIXYR = 0
# Save daily output or not [Default: Do set "FALSE"]. BOOLEAN = TRUE, FALSE
SAVEDDA = FALSE
# Save reduced daily output or not [Default: Do set "FALSE"]. BOOLEAN = TRUE, FALSE
ATM_REDUCED_OUTPUT = FALSE
# Store grib codes from SH files [User need to refer defined  ppt* files for the␣
↪experiment]
ATM_SH_CODES =
# Store levels against "ATM_SH_CODES" e.g: level1,level2,level3, ...
ATM_SH_LEVELS =
# Store grib codes from GG files [User need to refer defined  ppt* files for the␣
↪experiment]
ATM_GG_CODES =
# Store levels against "ATM_GG_CODES" (133.128, 246.128, 247.128, 248.128)
# e.g: level1,level2,level3, ...
ATM_GG_LEVELS =
# SPPT stochastic physics active or not [Default: set "FALSE"]. BOOLEAN = TRUE, FALSE
LSPPT = FALSE
# Write the perturbation patterns for SPPT or not [Default: set "FALSE"].
# BOOLEAN = TRUE, FALSE
LWRITE_ARP =
# Number of scales for SPPT [Default: set 3]. NUMERIC = 1, 2, 3
NS_SPPT =
# Standard deviations of each scale [Default: set 0.50,0.25,0.125]
# NUMERIC values separated by ,
SDEV_SPPT =
# Decorrelation times (in seconds) for each scale [Default: set 2.16E4,2.592E5,2.
↪592E6]
# NUMERIC values separated by ,
TAU_SPPT =
# Decorrelation lengths (in meters) for each scale [Default: set 500.E3,1000.E3,2000.
↪E3]
```

(continues on next page)

(continued from previous page)

```
# NUMERIC values separated by ,
XLCOR_SPPT =
# Clipping ratio (number of standard deviations) for SPPT [Default: set 2] NUMERIC
XCLIP_SPPT =
# Stratospheric tapering in SPPT [Default: set "TRUE"]. BOOLEAN = TRUE, FALSE
LTAPER_SPPT =
# Top of stratospheric tapering layer in Pa [Default: set to 50.E2] NUMERIC
PTAPER_TOP =
# Bottom of stratospheric tapering layer in Pa [Default: set to 100.E2] NUMERIC
PTAPER_BOT =
## ATMOSPHERIC NUDGING PARAMETERS ##
# Atmospheric nudging towards reinterpolated ERA-Interim data. BOOLEAN = TRUE, FALSE
ATM_NUDGING = FALSE
# Atmospheric nudging reference data experiment name. [T255L91: b0ir]
ATM_refnud =
# Nudge vorticity. BOOLEAN = TRUE, FALSE
NUD_VO =
# Nudge divergence. BOOLEAN = TRUE, FALSE
NUD_DI =
# Nudge temperature. BOOLEAN = TRUE, FALSE
NUD_TE =
# Nudge specific humidity. BOOLEAN = TRUE, FALSE
NUD_Q =
# Nudge liquid water content. BOOLEAN = TRUE, FALSE
NUD_QL =
# Nudge ice water content. BOOLEAN = TRUE, FALSE
NUD_QI =
# Nudge cloud fraction. BOOLEAN = TRUE, FALSE
NUD_QC =
# Nudge log of surface pressure. BOOLEAN = TRUE, FALSE
NUD_LP =
# Relaxation coefficient for vorticity. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_VO =
# Relaxation coefficient for divergence. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_DI =
# Relaxation coefficient for temperature. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_TE =
# Relaxation coefficient for specific humidity. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_Q =
# Relaxation coefficient for log surface pressure. NUMERIC in ]0,inf[;
# 1 means half way between model value and ref value
ALPH_LP =
# Nudging area Northern limit [Default: Do set "90"]
NUD_NLAT =
# Nudging area Southern limit [Default: Do set "-90"]
NUD_SLAT =
# Nudging area Western limit NUMERIC in [0,360] [Default: Do set "0"]
NUD_WLON =
# Nudging area Eastern limit NUMERIC in [0,360] [Default: Do set "360"; E<W will span␣
↪Greenwich]
NUD_ELON =
# Nudging vertical levels : lower level [Default: Do set "1"]
NUD_VMIN =
```

(continues on next page)

(continued from previous page)

```
# Nudging vertical levels : upper level [Default: Do set to number of vertical levels]
NUD_VMAX =

[nemo]
# Ocean initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_
↪conditions:oceanic
OCEAN_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
OCEAN_ini_member =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 1200 (ORCA025)
NEMO_timestep = 1200
# Number of parallel cores for OGCM component. NUMERIC = 16, 24, 36
NEMO_nproc = 960
# Ocean Advection Scheme [Default: Do set "tvd"]. STRING = tvd, cen2
ADVSCH = cen2
# Nudging activation. BOOLEAN = TRUE, FALSE
OCEAN_NUDGING = FALSE
# Toward which data to nudge; essential if "OCEAN_NUDGING" is TRUE.
# STRING = fa9p, s4, glorys2v1
OCEAN_NUDDATA = FALSE
# Rebuild and store restarts to HSM for an immediate prediction experiment.
# BOOLEAN = TRUE, FALSE
OCEAN_STORERST = FALSE

[ice]
# Sea-Ice Model [Default: Do set "LIM2"]. STRING = LIM2, LIM3
ICE = LIM3
# Sea-ice initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_
↪conditions:sea_ice
ICE_ini =
# A different IC member per EXPID member ["PERT"] or which common IC member
# for all EXPID members ["fc0" / "fc1"]. String = PERT/fc0/fc1...
ICE_ini_member =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 1200 (ORCA025)
LIM_timestep = 1200

[pisces]
# Activate PISCES (TRUE) or not (FALSE) [Default: leave empty]
PISCES = FALSE
# PISCES initial conditions ready to be used. [Default: leave empty].
# STRING = ID found here : https://earth.bsc.es/wiki/doku.php?id=initial_
↪conditions:biogeochemistry
PISCES_ini =
# Set timestep (in sec) w.r.t resolution. NUMERIC = 3600 (ORCA1), 3600 (ORCA025)
PISCES_timestep = 3600
```

Finally, you can launch Autosubmit *run* in background and with `nohup` (continue running although the user who launched the process logs out).

```
nohup autosubmit run cxxx &
```

2.1.4 Fourth Step: Experiment monitor

The following procedure could be adopted to generate the plots for visualizing the status of the experiment at any instance. With this command we can generate new plots to check which is the status of the experiment. Different job status are represented with different colors.

```
autosubmit monitor  cxxx
```

The location where user can find the generated plots with date and timestamp can be found below:

```
<experiments_directory>/cxxx/plot/cxxx_<date>_<time>.pdf
```

CHAPTER 3

Installation

3.1 How to install

The Autosubmit code is maintained in *PyPi*, the main source for python packages.

- Pre-requisties: These packages (bash, python2, sqlite3, git-scm > 1.8.2, subversion, dialog and GraphViz) must be available at local host machine.

These packages (argparse, python-dateutil, pyparsing, numpy, pydotplus, matplotlib, paramiko,python2-pythondialog and portalocker) must be available for python runtime.

Important: (SYSTEM) Graphviz version must be 2.38, 2.40 is not working, others perhaps works. You can check the version using dot -v.

Important: dot -v command should contain "dot",pdf,png,svg,xlib in device section.

Important: The host machine has to be able to access HPC's/Clusters via password-less ssh. Make sure that the ssh key is in PEM format *ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM*.

To install autosubmit just execute:

```
pip install autosubmit
```

or download, unpack and:

```
python setup.py install
```

Hint: To check if autosubmit has been installed run `autosubmit -v`. This command will print autosubmit's

current version

Hint: To read autosubmit's readme file, run `autosubmit readme`

Hint: To see the changelog, use `autosubmit changelog`

3.2 How to configure

After installation, you have to configure database and path for Autosubmit. In order to use the default settings, just create a directory called *autosubmit* in your home directory before running the configure command. The experiments will be created in this folder, and the database named *autosubmit.db* in your home directory.

```
autosubmit configure
```

For advanced options you can add *–advanced* to the configure command. It will allow you to choose different directories (they must exist) for the experiments and database, as well as configure SMTP server and an email account in order to use the email notifications feature.

```
autosubmit configure --advanced
```

Hint: The `dialog` (GUI) library is optional. Otherwise the configuration parameters will be prompted (CLI). Use `autosubmit configure -h` to see all the allowed options.

For installing the database for Autosubmit on the configured folder, when no database is created on the given path, execute:

```
autosubmit install
```

Danger: Be careful ! autosubmit install will create a blank database.

Lastly, if autosubmit configure doesn't work for you or you need to configure additional info create:

Create or modify /etc/autosubmitrc file or ~/.autosubmitrc with the information as follows:

```
[database]
path = path to autosubmit db
filename = autosubmit.db

[local]
path = path to experiment folders

[conf]
jobs = path to any experiment  jobs conf # If not working on esarchive, you must␣
↪create one from scratch check the how to.
platforms = path to any experiment  platform conf # If not working on esarchive, you␣
↪must create one from scratch check the how to.
```

(continues on next page)

(continued from previous page)

```
[mail]
smtp_server = mail.bsc.es
mail_from = automail@bsc.es

[structures]
path =  path to experiment folders

[globallogs]
path =  path to global logs (for expid,delete and migrate commands)

[historicdb]
path = <experiment_folder>/historic

[autosubmitapi]
url = url of Autosubmit API (The API is provided inside the BSC network)
# Autosubmit API provides extra information for some Autosubmit functions. It is not␣
↪mandatory to have access to it to use Autosubmit.

[hosts]
whitelist = localhost bscesautosubmit01 bscesautosubmit02 # Add localhost only if you␣
↪are not on esarchive system
```

Now you are ready to use Autosubmit !

CHAPTER 4

Usage

4.1 Command list

-expid	Create a new experiment
-create	Create specified experiment workflow
-check	Check configuration for specified experiment
-describe	Show details for specified experiment
-run	Run specified experiment
-inspect	Generate cmd files
-test	Test experiment
-testcase	Test case experiment
-monitor	Plot specified experiment
-stats	Plot statistics for specified experiment
-setstatus	Sets job status for an experiment
-recovery	Recover specified experiment
-clean	Clean specified experiment
-refresh	Refresh project directory for an experiment
-delete	Delete specified experiment
-configure	Configure database and path for autosubmit
-install	Install database for Autosubmit on the configured folder
-archive	Clean, compress and remove from the experiments' folder a finalized experiment
-unarchive	Restores an archived experiment
-migrate_exp	Migrates an experiment from one user to another

-report	extract experiment parameters
-updateversion	Updates the Autosubmit version of your experiment with the current version of the module you are using
-dbfix	Fixes the database malformed error in the historical database of your experiment
-pklfix	Fixed the blank pkl error of your experiment
-updatedescrip	Updates the description of your experiment (See: updateDescrip)

4.2 Tutorials (How to)

- new_experiment
- configuration
- run_modes
- workflow_recovery
- workflow_validation
- stats
- archive
- advanced_features

CHAPTER 5

Defining the workflow

One of the most important step that you have to do when planning to use autosubmit for an experiment is the definition of the workflow the experiment will use. In this section you will learn about the workflow definition syntax so you will be able to exploit autosubmit's full potential

Warning: This section is NOT intended to show how to define your jobs. Please go to *Tutorial* section for a comprehensive list of job options.

5.1 Simple workflow

The simplest workflow that can be defined it is a sequence of two jobs, with the second one triggering at the end of the first. To define it, we define the two jobs and then add a DEPENDECIES attribute on the second job referring to the first one.

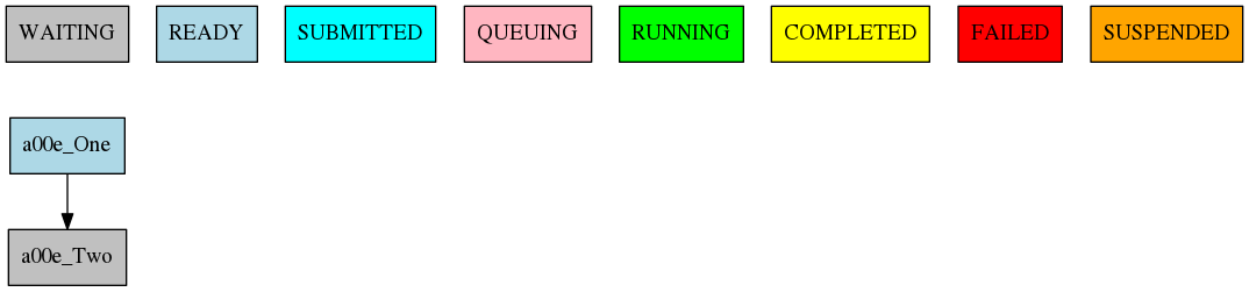
It is important to remember when defining workflows that DEPENDENCIES on autosubmit always refer to jobs that should be finished before launching the job that has the DEPENDENCIES attribute.

```
[One]
FILE = one.sh

[Two]
FILE = two.sh
DEPENDENCIES = One
```

The resulting workflow can be seen in Figure 5.1

5.2 Running jobs once per startdate, member or chunk

Autosubmit is capable of running ensembles made of various startdates and members. It also has the capability to divide member execution on different chunks.

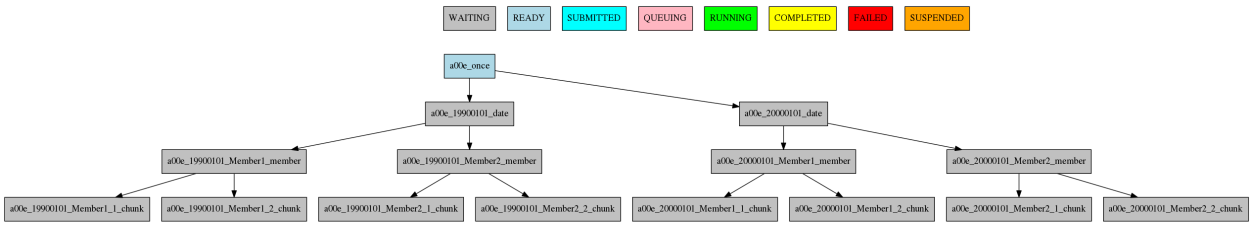


5.1: Example showing a simple workflow with two sequential jobs

To set at what level a job has to run you have to use the RUNNING attribute. It has four possible values: once, date, member and chunk corresponding to running once, once per startdate, once per member or once per chunk respectively.

```
[once]
FILE = Once.sh

[date]
FILE = date.sh
DEPENDENCIES = once
RUNNING = date

[member]
FILE = Member.sh
DEPENDENCIES = date
RUNNING = member

[chunk]
FILE = Chunk.sh
DEPENDENCIES = member
RUNNING = chunk
```

The resulting workflow can be seen in Figure 5.2 for a experiment with 2 startdates, 2 members and 2 chunks.



5.2: Example showing how to run jobs once per startdate, member or chunk.

5.3 Dependencies

Dependencies on autosubmit were introduced on the first example, but in this section you will learn about some special cases that will be very useful on your workflows.

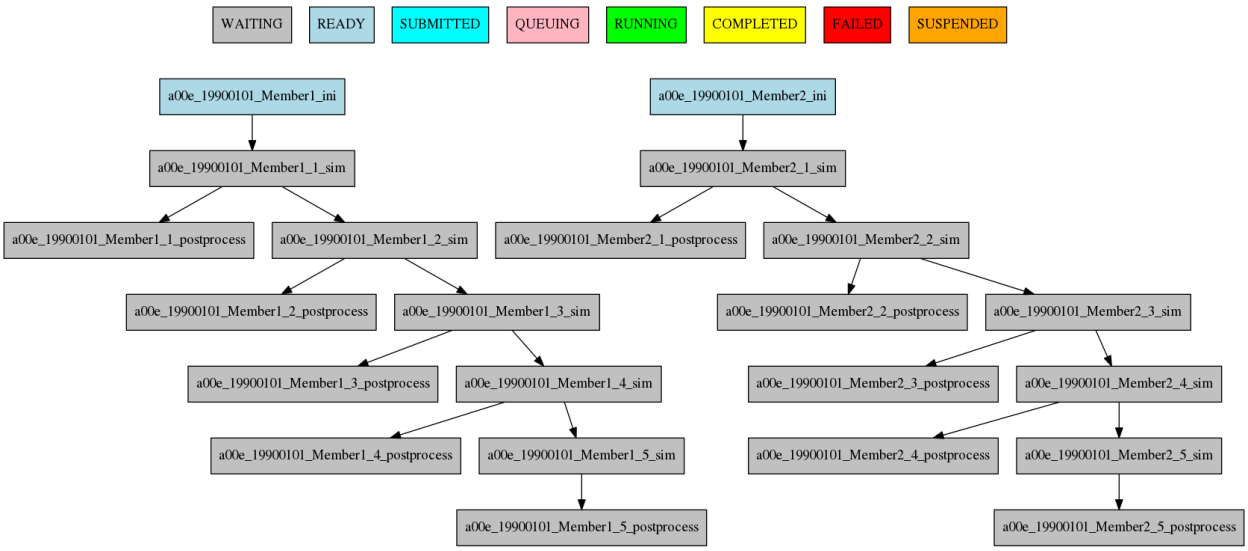
5.3.1 Dependencies with previous jobs

Autosubmit can manage dependencies between jobs that are part of different chunks, members or startdates. The next example will show how to make a simulation job wait for the previous chunk of the simulation. To do that, we add sim-1 on the DEPENDENCIES attribute. As you can see, you can add as much dependencies as you like separated by spaces

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk
```

The resulting workflow can be seen in Figure 5.3

Warning: Autosubmit simplifies the dependencies, so the final graph usually does not show all the lines that you may expect to see. In this example you can see that there are no lines between the ini and the sim jobs for chunks 2 to 5 because that dependency is redundant with the one on the previous sim



5.3: Example showing dependencies between sim jobs on different chunks.

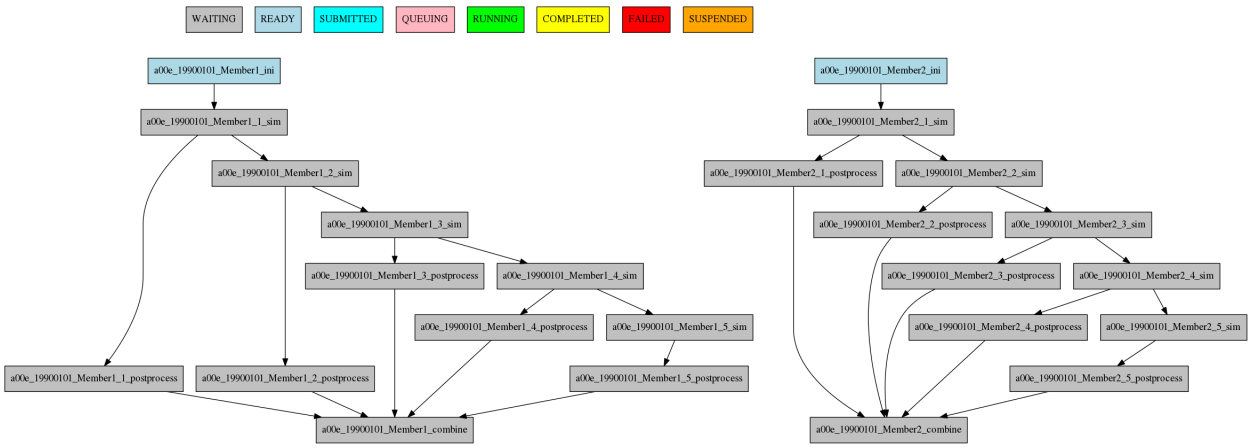
5.3.2 Dependencies between running levels

On the previous examples we have seen that when a job depends on a job on a higher level (a running chunk job depending on a member running job) all jobs wait for the higher running level job to be finished. That is the case on the ini sim dependency on the next example.

In the other case, a job depending on a lower running level job, the higher level job will wait for ALL the lower level jobs to be finished. That is the case of the postprocess combine dependency on the next example.

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
RUNNING = member
```

The resulting workflow can be seen in Figure 5.4



5.4: Example showing dependencies between jobs running at different levels.

5.4 Job frequency

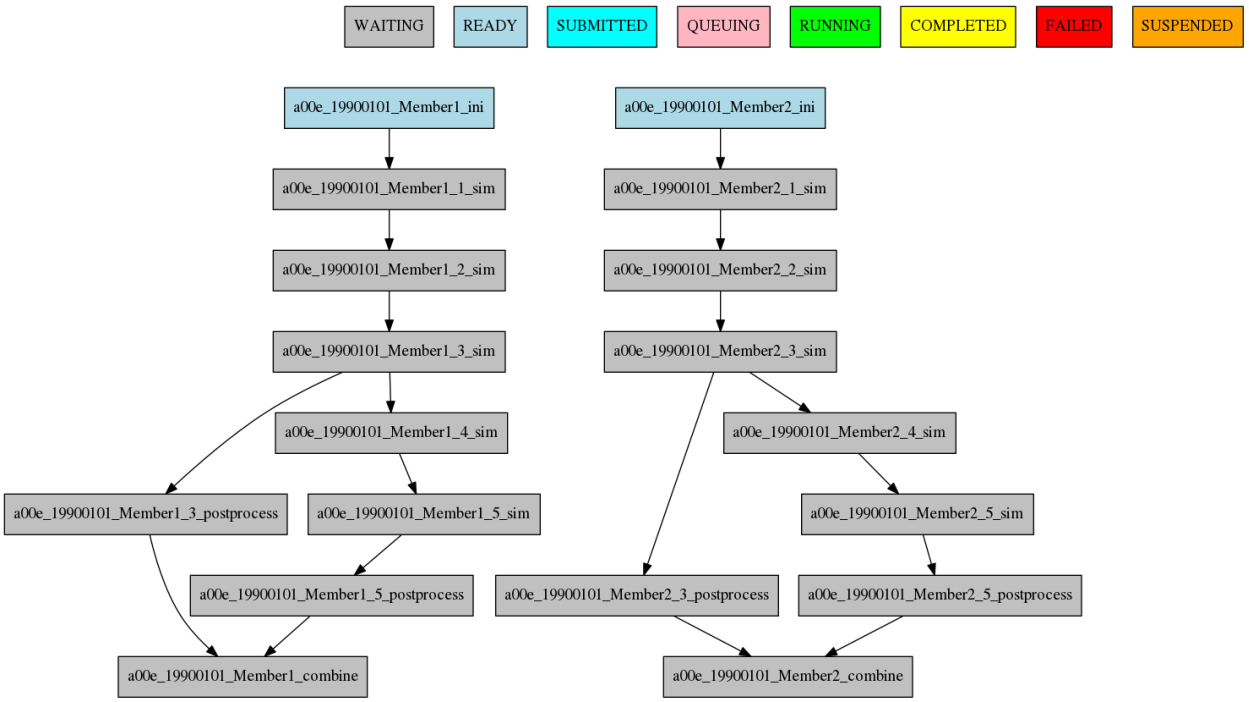
Some times you just don't need a job to be run on every chunk or member. For example, you may want to launch the postprocessing job after various chunks have completed. This behaviour can be achieved using the FREQUENCY attribute. You can specify an integer I for this attribute and the job will run only once for each I iterations on the running level.

Hint: You don't need to adjust the frequency to be a divisor of the total jobs. A job will always execute at the last iteration of its running level

---

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
RUNNING = chunk
FREQUENCY = 3

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
RUNNING = member
```

The resulting workflow can be seen in Figure 5.5



5.5: Example showing dependencies between jobs running at different frequencies.

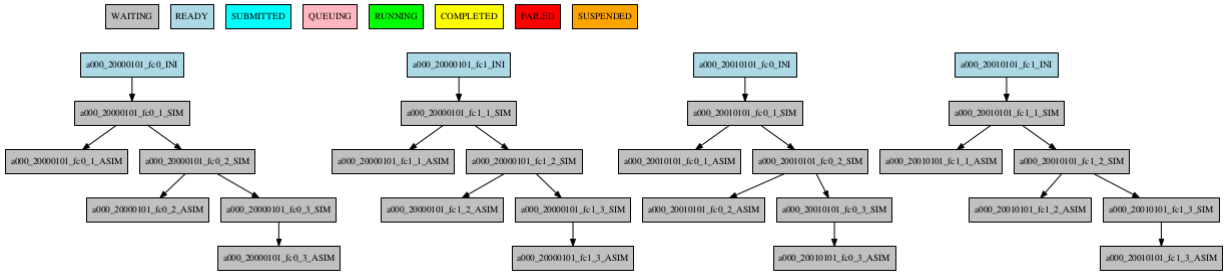
5.5 Job synchronize

For jobs running at chunk level, and this job has dependencies, you could want not to run a job for each experiment chunk, but to run once for all member/date dependencies, maintaining the chunk granularity. In this cases you can use the SYNCHRONIZE job parameter to determine which kind of synchronization do you want. See the below examples with and without this parameter.

Hint: This job parameter works with jobs with RUNNING parameter equals to 'chunk'.

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = INI SIM-1
RUNNING = chunk

[ASIM]
FILE = asim.sh
DEPENDENCIES = SIM
RUNNING = chunk
```

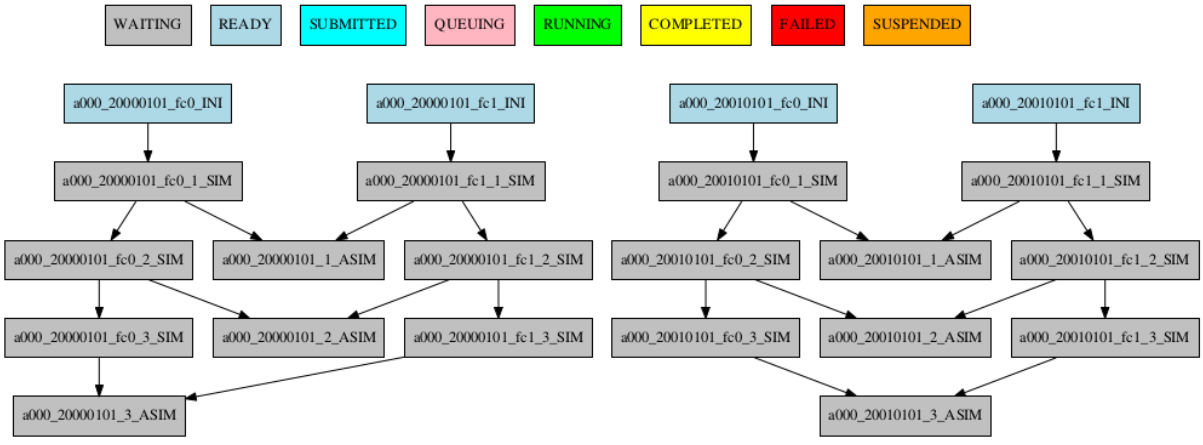
The resulting workflow can be seen in Figure 5.6



5.6: Example showing dependencies between chunk jobs running without synchronize.

```
[ASIM]
SYNCHRONIZE = member
```

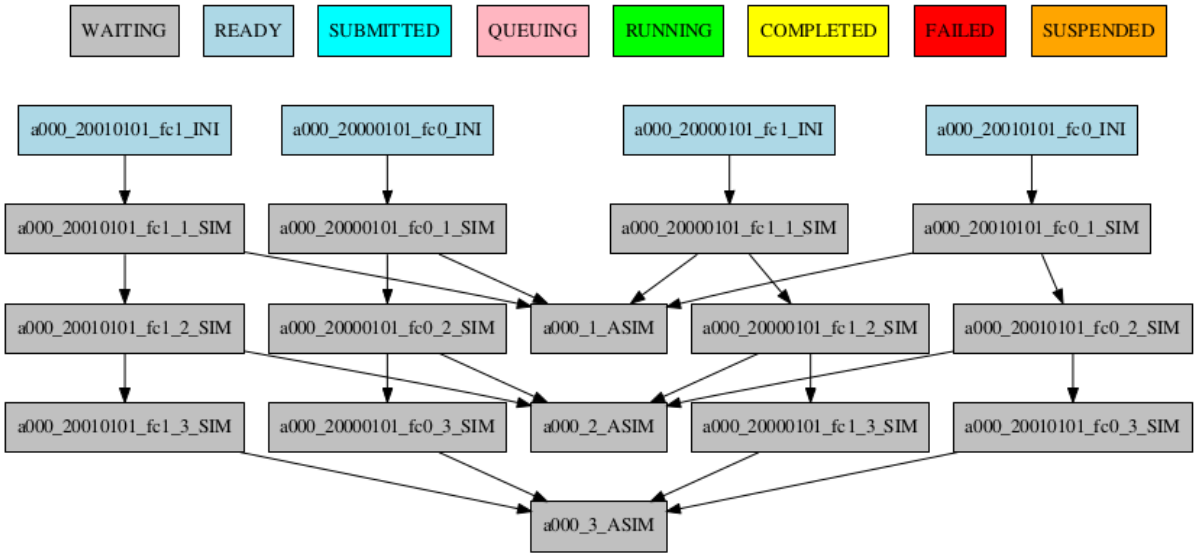
The resulting workflow of setting SYNCHRONIZE parameter to 'member' can be seen in Figure 5.7



5.7: Example showing dependencies between chunk jobs running with member synchronize.

```
[ASIM]
SYNCHRONIZE = date
```

The resulting workflow of setting SYNCHRONIZE parameter to 'date' can be seen in Figure 5.8



5.8: Example showing dependencies between chunk jobs running with date synchronize.

5.6 Job split

For jobs running at chunk level, it may be useful to split each chunk into different parts. This behaviour can be achieved using the SPLITS attribute to specify the number of parts. It is possible to define dependencies to specific splits within [], as well as to a list/range of splits, in the format [1:3,7,10] or [1,2,3]

Hint: This job parameter works with jobs with RUNNING parameter equals to 'chunk'.

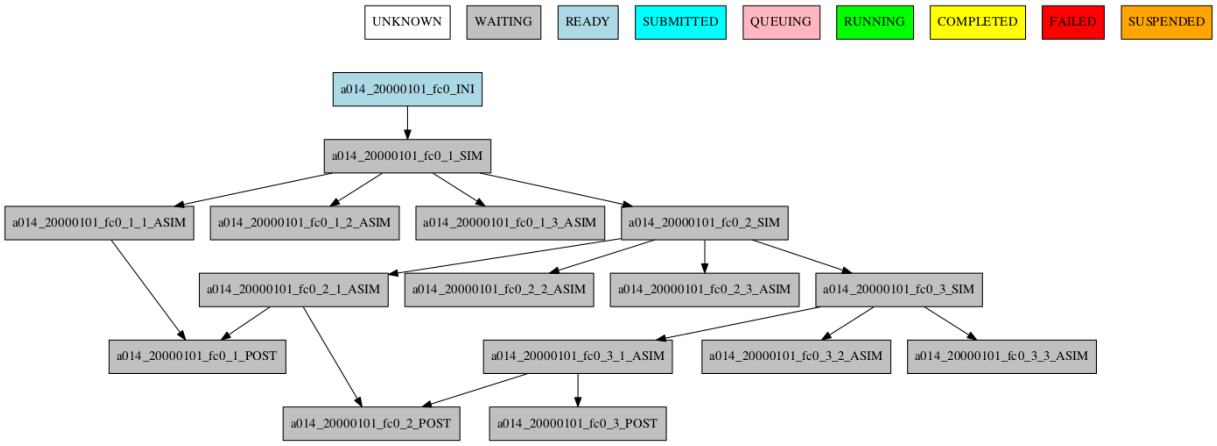
```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[asim]
FILE = asim.sh
DEPENDENCIES = sim
RUNNING = chunk
SPLITS = 3

[post]
```

(continues on next page)

(continued from previous page)

```
FILE = post.sh
RUNNING = chunk
DEPENDENCIES = asim[1] asim[1]+1
```

The resulting workflow can be seen in Figure 5.9



5.9: Example showing the job ASIM divided into 3 parts for each chunk.

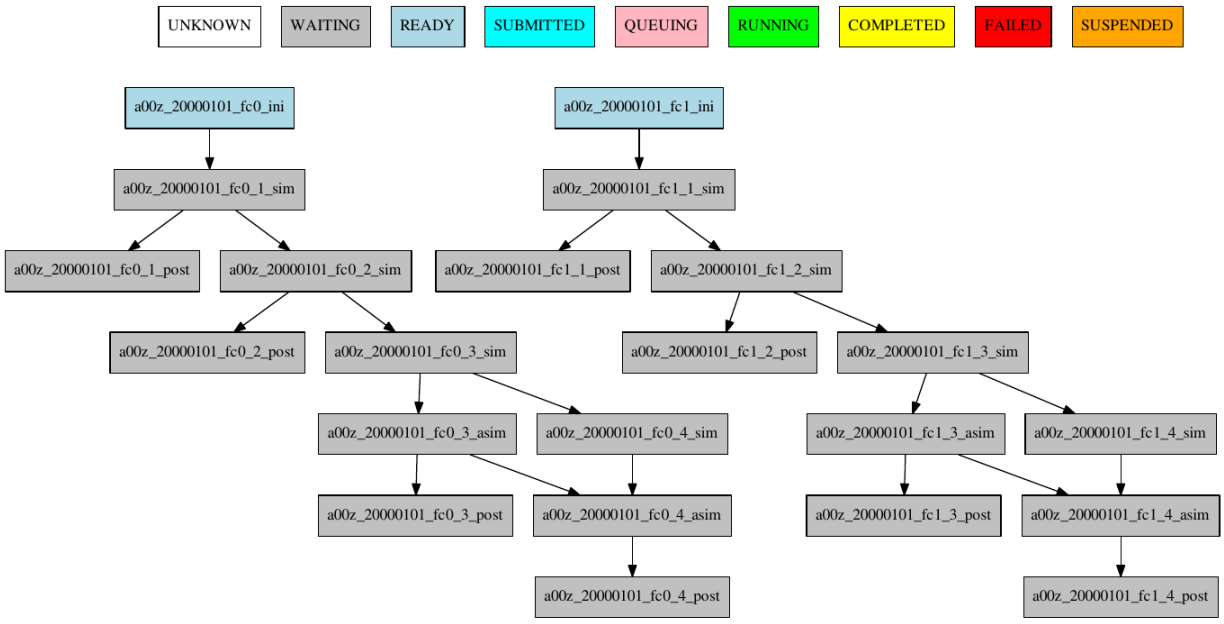
5.7 Job delay

Some times you need a job to be run after a certain number of chunks. For example, you may want to launch the asim job after various chunks have completed. This behaviour can be achieved using the DELAY attribute. You can specify an integer N for this attribute and the job will run only after N chunks.

Hint: This job parameter works with jobs with RUNNING parameter equals to 'chunk'.

```
[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[asim]
FILE = asim.sh
DEPENDENCIES = sim asim-1
RUNNING = chunk
DELAY = 2

[post]
FILE = post.sh
DEPENDENCIES = sim asim
RUNNING = chunk
```

The resulting workflow can be seen in Figure 5.10



5.10: Example showing the asim job starting only from chunk 3.

5.8 Rerun dependencies

Autosubmit has the possibility to rerun some chunks of the experiment without affecting everything else. In this case, autosubmit will automatically rerun all jobs of that chunk. If some of this jobs need another one on the workflow you have to add the RERUN_DEPENDENCIES attribute and specify which jobs to rerun.

It is also usual that you will have some code that it is needed only in the case of a rerun. You can add this jobs to the workflow as usual and set the attribute RERUN_ONLY to true. This jobs will be omitted from the workflow in the normal case, but will appear on the reruns.

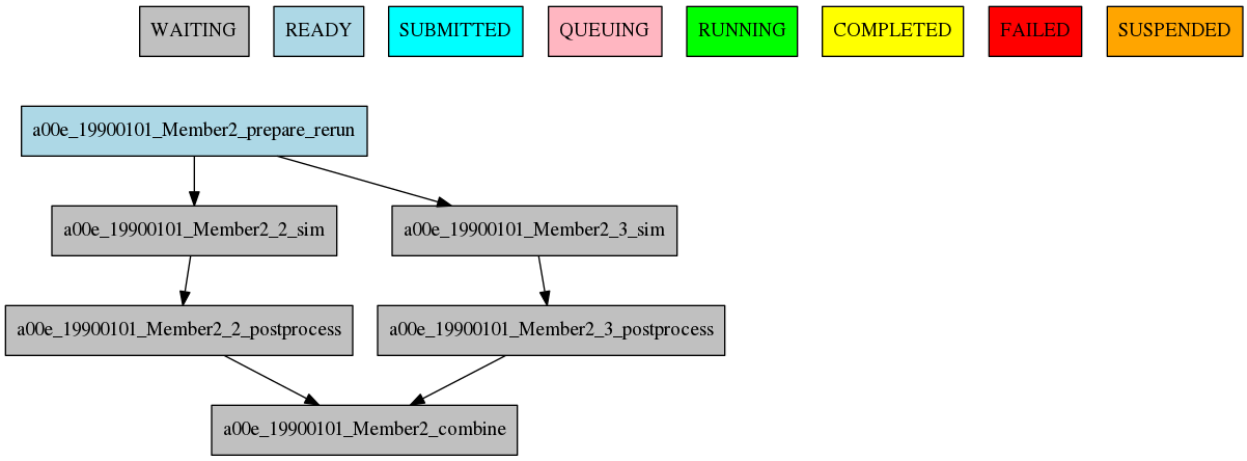
```
[prepare_rerun]
FILE = prepare_rerun.sh
RERUN_ONLY = true
RUNNING = member

[ini]
FILE = ini.sh
RUNNING = member

[sim]
FILE = sim.sh
DEPENDENCIES = ini combine prepare_rerun
RERUN_DEPENDENCIES = combine prepare_rerun
RUNNING = chunk

[postprocess]
FILE = postprocess.sh
DEPENDENCIES = sim
```

(continues on next page)

(continued from previous page)

```
RUNNING = chunk

[combine]
FILE = combine.sh
DEPENDENCIES = postprocess
RUNNING = member
```

The resulting workflow can be seen in Figure 5.11 for a rerun of chunks 2 and 3 of member 2.



5.11: Example showing a rerun workflow for chunks 2 and 3.

CHAPTER 6

Frequent Questions and Answers

The latest version of **Autosubmit** implements a code system that guides you through the process of fixing some of the common problems you might find. Consequently, the **FAQ** section has been replaced by *Error codes and solutions*, where you will find the list of error codes, their descriptions, and solutions.

CHAPTER 7

Troubleshooting

7.1 How to change the job status stopping autosubmit

Review setstatus.

7.2 How to change the job status without stopping autosubmit

Review setstatusno.

7.3 My project parameters are not being substituted in the templates.

Explanation: If there is a duplicated section or option in any other side of autosubmit, including proj files It won't be able to recognize which option pertains to what section in which file.

Solution: Don't repeat section names and parameters names until Autosubmit 4.0 release.

7.4 Unable to recover remote logs files.

Explanation: If there are limitations on the remote platform regarding multiple connections, *Solution*: You can try DISABLE_RECOVERY_THREADS = TRUE under the [platform_name] section in the platform.conf.

7.5 Other possible errors

I see the 'database malformed' error on my experiment log.

Explanation: The latest version of autosubmit uses a database to efficiently track changes in the jobs of your experiment. It might happen that this small database gets corrupted.

Solution: run *autosubmit dbfix expid* where *expid* is the identifier of your experiment. This function will rebuild the database saving as much information as possible (usually all of it).

The pkl file of my experiment is empty but there is a job_list_%expid%_backup.pkl file that seems to be the real one.

Solution: run *autosubmit pklfix expid*, it will restore the *backup* file if possible.

CHAPTER 8

Error codes and solutions

8.1 Experiment Locked - Critical Error 7000

Code | Details | Solution		
7000	Experiment is locked due another instance of Autosubmit using it	Halt other experiment instances //Delete <expid>/tmp/autosubmit.lock

8.2 Database Issues - Critical Error codes [7001-7005]

Code | Details | Solution		
7001	Connection to the db could not be established	Check if database exist
7002	Wrong version	Check system sqlite version
7003	DB doesn't exist	Check if database exist
7004	Can't create a new database	Check your user permissions
7005	AS database is corrupted or locked	Please, open a new issue ASAP. (If you are on BSC environment)

8.2.1 Default Solution

These issues are usually from server side, please, ask first in Autosubmit git if you don't have a custom installation.

8.3 Wrong User Input - Critical Error codes [7010-7030]

Code	Details	Solution
7010	Experiment has been halted in a manual way	
7011	Wrong arguments for an specific command	Check the command section for more info
7012	Insufficient permissions for an specific experiment.	Check if you have enough permissions, experiment exist or specified expid has a typo
7013	Pending commits	You must commit/synchronize pending changes in the experiment proj folder.
7014	Wrong configuration	Check your experiment/conf files, also take a look to the ASLOG/command.log detailed output

8.3.1 Default Solution

These issues are usually mistakes from the user input, check the available logs and git resolved issues. Alternative, you can ask for help to Autosubmit team.

8.4 Platform issues - Critical Error codes. Local [7040-7050] and remote [7050-7060]

Code | Details | Solution		
7040	Invalid experiment pkl/db likely due a local platform failure	Should be recovered automatically, if not check if there is a backup file and do it manually
7041	Weird job status	Weird Job status, try to recover experiment(check the recovery how-to for more info) if this issue persist please, report it to gitlab
7050	Connection can't be established.	Check your experiment platform configuration
7050	Failure after a restart, connection can't be restored.	Check or ask (manually) if the remote platforms have any known issues
7051	Invalid ssh configuration.	Check .ssh/config file. Additionally, Check if you can perform a password less connection to that platform.

8.4.1 Default Solution

Check autosubmit log for detailed information, there will be additional error codes.

8.5 Uncatalogued codes - Critical Error codes [7060+]

Code	Details	Solution
7060	Display issues during monitoring	Use a different output or txt.
7061	Stat command failed	Check Aslogs command output, open a git issue.
7062	Svn issues	Check, in expdef, if url exist.
7063	cp/rsync issues	Check if destination path exist.
7064	Git issues	Check that the proj folder is a well configured git folder. Also, check [GIT] expdef config.
7065	Wrong git configuration	Invalid git url. Check [GIT] expdef config. If issue persists, check if proj folder is a well configured git folder.
7066	Presubmission feature issues	New feature, this message shouldn't be prompt. Please report it to Git.
7067	Historical Database not found	Configure [historicdb] PATH = <file_path>.
7068	Monitor output can't be loaded	Try another output method// Check if the experiment is reachable
7069	Monitor output format invalid	Try another output method

8.5.1 Default Solution

Check autosubmit log for detailed information, there will be additional error codes.

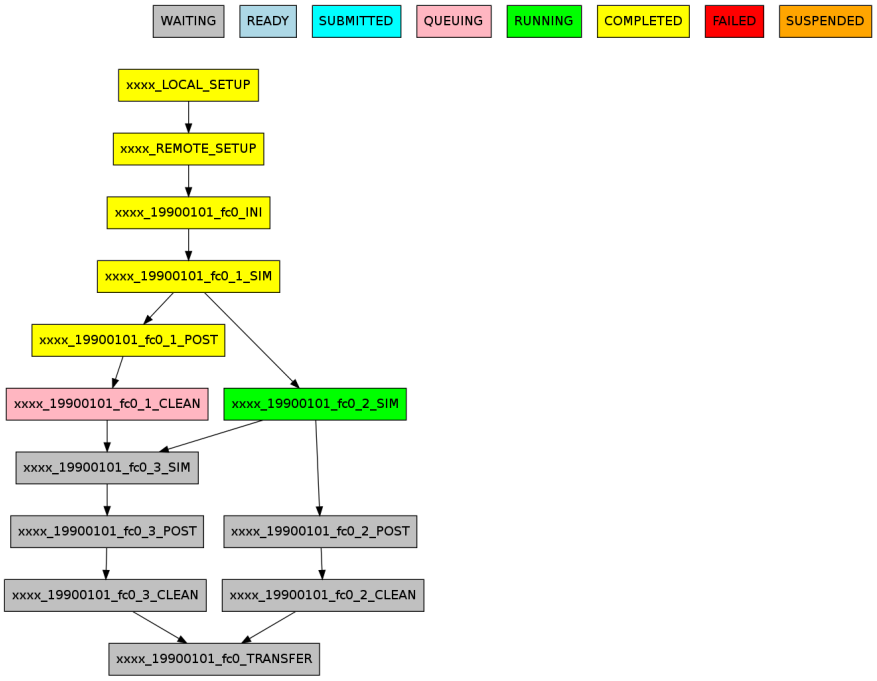
8.6 Minor errors - Error codes [6000+]

Code	Details	Solution
6001	Failed to retrieve log files	Automatically, if there aren't bigger issues
6002	Failed reconection	Automatically, if there aren't bigger issues
6003	Failed connection, wrong configuration	Check your platform.conf file
6004	Input output issues	Automatically, if there aren't bigger issues
6005	Unable to execute the command	Automatically, if there aren't bigger issues
6006	Failed command	Check err output for more info, command worked but some issue was detected
6007	Broken sFTP connection	Automatically, if there aren't bigger issues
6008	Inconsistent/unexpected ,job status	Automatically, if there aren't bigger issues
6009	Failed job checker	Automatically, if there aren't bigger issues
6010	Corrupted job_list using backup	Automatically, if it fails, Perform mv <expid>/pkl/job_list_backup.pkl <expid>/pkl/job_list.pkl
6011	Incorrect mail notifier configuration	Double check your mail configuration on job.conf (job status) and autosubmit.conf (email)
6012	Migrate , archive/unarchive I/O issues	Check migrate how-to configuration
6013	Configuration issues	Check log output for more info
6014	Git Can't clone repository submodel	Check submodule url, perform a refresh
6015	Submission failed	Automatically, if there aren't bigger issues

CHAPTER 9

Developing a project

Autosubmit is used at BSC to run EC-Earth. To do that, a git repository has been created that contains the model source code and the scripts used to run the tasks.



9.1: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

The workflow is defined using seven job types, as shown in the figure above. These job types are:

- Local_setup: prepares a patch for model changes and copies it to HPC.
- Remote_setup: creates a model copy and applies the patch to it.
- Ini: prepares model to start the simulation of one member.

- Sim: runs a simulation chunk (usually 1 to 3 months).
- Post: post-process outputs for one simulation chunk.
- Clean: removes unnecessary outputs from the simulated chunk.
- Transfer: transfers post-processed outputs to definitive storage.

Since Autosubmit 2.2 the user can select the desired source repository for the experiment project and using a given concrete branch is possible. This introduce a better version control system for project and more options to create new experiments based on different developments by the user. The different projects contain the shell script to run, for each job type (local setup, remote setup, ini, sim, post, clean and transfer) that are platform independent. Additionally the user can modify the sources under proj folder. The executable scripts are created at runtime so the modifications on the sources can be done on the fly.

Warning: Autosubmit automatically adds small shell script code blocks in the header and the tailer of your scripts, to control the workflow. Please, remove any exit command in the end of your scripts, e.g. `exit 0`.

Important: For a complete reference on how to develop an EC-Earth project, please have a look in the following wiki page: https://earth.bsc.es/wiki/doku.php?id=models:models

CHAPTER 10

Variables reference

Autosubmit uses a variable substitution system to facilitate the development of the templates. This variables can be used on the template in the form %VARIABLE_NAME%.

10.1 Job variables

This variables are relatives to the current job.

- **TASKTYPE**: type of the job, as given on job configuration file.
- **JOBNAME**: current job full name.
- **FAIL_COUNT**: number of failed attempts to run this job.
- **SDATE**: current startdate.
- **MEMBER**: current member.
- **CHUNK**: current chunk.
- **SPLIT**: current split.
- **DELAY**: current delay.
- **DAY_BEFORE**: day before the startdate
- **Chunk_End_IN_DAYS**: chunk's length in days
- **Chunk_START_DATE**: chunk's start date
- **Chunk_START_YEAR**: chunk's start year
- **Chunk_START_MONTH**: chunk's start month
- **Chunk_START_DAY**: chunk's start day
- **Chunk_START_HOUR**: chunk's start hout
- **Chunk_END_DATE**: chunk's end date

- **Chunk_END_YEAR**: chunk's end year
- **Chunk_END_MONTH**: chunk's end month
- **Chunk_END_DAY**: chunk's end day
- **Chunk_END_HOUR**: chunk's end hour
- **PREV**: days since startdate at the chunk's start
- **Chunk_FIRST**: True if the current chunk is the first, false otherwise.
- **Chunk_LAST**: True if the current chunk is the last, false otherwise.
- **NUMPROC**: Number of processors that the job will use.
- **NUMTHREADS**: Number of threads that the job will use.
- **NUMTASK**: Number of tasks that the job will use.
- **WALLCLOCK**: Number of processors that the job will use.
- **SCRATCH_FREE_SPACE**: Percentage of free space required on the `scratch`.
- **NOTIFY_ON**: Determine the job statuses you want to be notified.
- **WRAPPER**: Wrapper type, None if wrapper is not being used

10.2 Platform variables

This variables are relative to the platoforms defined on the jobs conf. A full set of the next variables are defined for each platform defined on the platforms configuration file, substituting {PLATFORM_NAME} for each platform's name. Also, a suite of variables is defined for the current platform where {PLATFORM_NAME} is substituted by CURRENT.

- **{PLATFORM_NAME}_ARCH**: Platform name
- **{PLATFORM_NAME}_HOST**: Platform url
- **{PLATFORM_NAME}_USER**: Platform user
- **{PLATFORM_NAME}_PROJ**: Platform project
- **{PLATFORM_NAME}_BUDG**: Platform budget
- **{PLATFORM_NAME}_RESERVATION**: You can configure your reservation id for the given platform.
- **{PLATFORM_NAME}_EXCLUSIVITY**: True if you want to request exclusivity nodes.
- **{PLATFORM_NAME}_TYPE**: Platform scheduler type
- **{PLATFORM_NAME}_VERSION**: Platform scheduler version
- **{PLATFORM_NAME}_SCRATCH_DIR**: Platform's scratch folder path
- **{PLATFORM_NAME}_ROOTDIR**: Platform's experiment folder path
- **{PLATFORM_NAME}_CUSTOM_DIRECTIVES**: Platform's custom directives for the resource manager.

Hint: The variables `_USER`, `_PROJ` and `_BUDG` has no value on the LOCAL platform.

Hint: Until now, the variables `_RESERVATION` and `_EXCLUSIVITY` are only available for MN.

It is also defined a suite of variables for the experiment's default platform:

- **HPCARCH**: Default HPC platform name
- **HPCHOST**: Default HPC platform url
- **HPCUSER**: Default HPC platform user
- **HPCPROJ**: Default HPC platform project
- **HPCBUDG**: Default HPC platform budget
- **HPCTYPE**: Default HPC platform scheduler type
- **HPCVERSION**: Default HPC platform scheduler version
- **SCRATCH_DIR**: Default HPC platform scratch folder path
- **HPCROOTDIR**: Default HPC platform experiment's folder path

10.3 Project variables

- **NUMMEMBERS**: number of members of the experiment
- **NUMCHUNKS**: number of chunks of the experiment
- **CHUNKSIZE**: size of each chunk
- **CHUNKSIZEUNIT**: unit of the chuk size. Can be hour, day, month or year.
- **CALENDAR**: calendar used for the experiment. Can be standard or noleap.
- **ROOTDIR**: local path to experiment's folder
- **PROJDIR**: local path to experiment's proj folder

10.4 Performance Metrics

Currently, these variables apply only to the report function of Autosubmit. See report.

- **SYPD**: Simulated years per day.
- **ASYPD**: Actual simulated years per day.
- **RSYPD**: Raw simulated years per day.
- **CHSY**: Core hours per simulated year.
- **JPSY**: Joules per simulated year.
- **Parallelization**: Number of cores requested for the simulation job.

For more information about these metrics please visit:

https://earth.bsc.es/gitlab/wuruchi/autosubmitreact/-/wikis/Performance-Metrics.

CHAPTER 11

Module documentation

11.1 autosubmit

class autosubmit.autosubmit.**Autosubmit**
Interface class for autosubmit.

static archive(*expid*, *noclean=True*, *uncompress=True*)
Archives an experiment: call clean (if experiment is of version 3 or later), compress folder to tar.gz and moves to year's folder

Parameters

- **clean,compress** –
- **expid** (*str*) – experiment identifier

Returns

static change_status(*final*, *final_status*, *job*, *save*)
Set job status to final

Parameters

- **final** –
- **final_status** –
- **job** –

static check(*experiment_id*, *notransitive=False*)
Checks experiment configuration and warns about any detected error or inconsistency.

Parameters **experiment_id** (*str*) – experiment identifier:

static clean(*expid*, *project*, *plot*, *stats*)
Clean experiment's directory to save storage space. It removes project directory and outdated plots or stats.

Parameters

- **create_log_file** (*bool*) – if true, creates log file

- **expid** (*str*) – identifier of experiment to clean
- **project** (*bool*) – set True to delete project directory
- **plot** (*bool*) – set True to delete outdated plots
- **stats** (*bool*) – set True to delete outdated stats

static configure(*advanced*, *database_path*, *database_filename*, *local_root_path*, *platforms_conf_path*, *jobs_conf_path*, *smtp_hostname*, *mail_from*, *machine*, *local*)

Configure several paths for autosubmit: database, local root and others. Can be configured at system, user or local levels. Local level configuration precedes user level and user level precedes system configuration.

Parameters

- **database_path** (*str*) – path to autosubmit database
- **database_filename** (*str*) – database filename
- **local_root_path** (*str*) – path to autosubmit's experiments' directory
- **platforms_conf_path** (*str*) – path to platforms conf file to be used as model for new experiments
- **jobs_conf_path** (*str*) – path to jobs conf file to be used as model for new experiments
- **machine** (*bool*) – True if this configuration has to be stored for all the machine users
- **local** (*bool*) – True if this configuration has to be stored in the local path
- **mail_from** (*str*) –
- **smtp_hostname** (*str*) –

static configure_dialog()

Configure several paths for autosubmit interactively: database, local root and others. Can be configured at system, user or local levels. Local level configuration precedes user level and user level precedes system configuration.

static create(*expid*, *noplot*, *hide*, *output='pdf'*, *group_by=None*, *expand=[]*, *expand_status=[]*, *notransitive=False*, *check_wrappers=False*, *detail=False*)

Creates job list for given experiment. Configuration files must be valid before executing this process.

Parameters

- **expid** (*str*) – experiment identifier
- **noplot** – if True, method omits final plotting of the jobs list. Only needed on large experiments when

plotting time can be much larger than creation time. :type noplot: bool :return: True if successful, False if not :rtype: bool :param hide: hides plot window :type hide: bool :param hide: hides plot window :type hide: bool :param output: plot's file format. It can be pdf, png, ps or svg :type output: str

static database_fix(*expid*)

Database methods. Performs a sql dump of the database and restores it.

Parameters **expid** (*str*) – experiment identifier

Returns

Return type

static delete(*expid*, *force*)

Deletes and experiment from database and experiment's folder

Parameters

- **expid** (*str*) – identifier of the experiment to delete
- **force** (*bool*) – if True, does not ask for confirmation

Returns True if succesful, False if not

Return type bool

static describe(*experiment_id*)
Show details for specified experiment

Parameters **experiment_id** (*str*) – experiment identifier:

static expid(*hpc*, *description*, *copy_id=''*, *dummy=False*, *test=False*, *operational=False*, *root_folder=''*)
Creates a new experiment for given HPC

Parameters

- **operational** (*bool*) – if true, creates an operational experiment
- **hpc** (*str*) – name of the main HPC for the experiment
- **description** (*str*) – short experiment's description.
- **copy_id** (*str*) – experiment identifier of experiment to copy
- **dummy** (*bool*) – if true, writes a default dummy configuration for testing
- **test** – if true, creates an experiment for testing

Returns experiment identifier. If method fails, returns ''.

Return type str

static generate_scripts_andor_wrappers(*as_conf*, *job_list*, *jobs_filtered*, *packages_persistence*, *only_wrappers=False*)

Parameters

- **as_conf** (*AutosubmitConfig() Object*) – Class that handles basic configuration parameters of Autosubmit.
- **job_list** (*JobList() Object*) – Representation of the jobs of the experiment, keeps the list of jobs inside.
- **jobs_filtered** (*List() of Job Objects*) – list of jobs that are relevant to the process.
- **packages_persistence** (*JobPackagePersistence() Object*) – Object that handles local db persistence.
- **only_wrappers** (*Boolean*) – True when coming from Autosubmit.create(). False when coming from Autosubmit.inspect(),

Returns Nothing

Return type

static inspect(*expid*, *lst*, *filter_chunks*, *filter_status*, *filter_section*, *notransitive=False*, *force=False*, *check_wrapper=False*)
Generates cmd files experiment.

Parameters **expid** (*str*) – identifier of experiment to be run

Returns True if run to the end, False otherwise

Return type bool

static install()
Creates a new database instance for autosubmit at the configured path

static migrate(*experiment_id*, *offer*, *pickup*, *only_remote*)
Migrates experiment files from current to other user. It takes mapping information for new user from config files.

Parameters

- **experiment_id** – experiment identifier:
- **pickup** –
- **offer** –
- **only_remote** –

static monitor(*expid*, *file_format*, *lst*, *filter_chunks*, *filter_status*, *filter_section*, *hide*, *txt_only=False*, *group_by=None*, *expand=[]*, *expand_status=[]*, *hide_groups=False*, *notransitive=False*, *check_wrapper=False*, *txt_logfiles=False*, *detail=False*)
Plots workflow graph for a given experiment with status of each job coded by node color. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

- **expid** (*str*) – identifier of the experiment to plot
- **file_format** (*str*) – plot's file format. It can be pdf, png, ps or svg
- **lst** (*str*) – list of jobs to change status
- **filter_chunks** (*str*) – chunks to change status
- **filter_status** (*str*) – current status of the jobs to change status
- **filter_section** (*str*) – sections to change status
- **hide** (*bool*) – hides plot window

static parse_args()
Parse arguments given to an executable and start execution of command given

static pkl_fix(*expid*)
Tries to find a backup of the pkl file and restores it. Verifies that autosubmit is not running on this experiment.

Parameters **expid** (*str*) – experiment identifier

Returns

Return type

static recovery(*expid*, *noplot*, *save*, *all_jobs*, *hide*, *group_by=None*, *expand=[]*, *expand_status=[]*, *notransitive=False*, *no_recover_logs=False*, *detail=False*)
Method to check all active jobs. If COMPLETED file is found, job status will be changed to COMPLETED, otherwise it will be set to WAITING. It will also update the jobs list.

Parameters

- **expid** (*str*) – identifier of the experiment to recover
- **save** (*bool*) – If true, recovery saves changes to the jobs list
- **all_jobs** (*bool*) – if True, it tries to get completed files for all jobs, not only active.

- **hide** (*bool*) – hides plot window

static refresh(*expid*, *model_conf*, *jobs_conf*)
Refresh project folder for given experiment

Parameters

- **model_conf** (*bool*) –
- **jobs_conf** (*bool*) –
- **expid** (*str*) – experiment identifier

static report(*expid*, *template_file_path=''*, *show_all_parameters=False*, *folder_path=''*, *placeholders=False*)
Show report for specified experiment :param expid: experiment identifier: :type str :param template_file_path: template filepath :type str :param show_all_parameters: Write all parameters of the experiment :type bool :param folder_path: Allows to put the report files on another folder :type str

static rerun_recovery(*expid*, *job_list*, *rerun_list*, *as_conf*)
Method to check all active jobs. If COMPLETED file is found, job status will be changed to COMPLETED, otherwise it will be set to WAITING. It will also update the jobs list.

Parameters

- **expid** (*str*) – identifier of the experiment to recover
- **save** (*bool*) – If true, recovery saves changes to the jobs list
- **all_jobs** (*bool*) – if True, it tries to get completed files for all jobs, not only active.
- **hide** (*bool*) – hides plot window

static run_experiment(*expid*, *notransitive=False*, *update_version=False*, *start_time=None*, *start_after=None*, *run_members=None*)
Runs and experiment (submitting all the jobs properly and repeating its execution in case of failure).

Parameters **expid** (*str*) – identifier of experiment to be run

Returns True if run to the end, False otherwise

Return type bool

static set_status(*expid*, *noplot*, *save*, *final*, *lst*, *filter_chunks*, *filter_status*, *filter_section*, *filter_type_chunk*, *hide*, *group_by=None*, *expand=[]*, *expand_status=[]*, *notransitive=False*, *check_wrapper=False*, *detail=False*)
Set status

Parameters

- **expid** (*str*) – experiment identifier
- **save** (*bool*) – if true, saves the new jobs list
- **final** (*str*) – status to set on jobs
- **lst** (*str*) – list of jobs to change status
- **filter_chunks** (*str*) – chunks to change status
- **filter_status** (*str*) – current status of the jobs to change status
- **filter_section** (*str*) – sections to change status
- **hide** (*bool*) – hides plot window

static statistics(*expid*, *filter_type*, *filter_period*, *file_format*, *hide*, *notransitive=False*)
Plots statistics graph for a given experiment. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

- **expid** (*str*) – identifier of the experiment to plot
- **filter_type** – type of the jobs to plot
- **filter_period** – period to plot
- **file_format** (*str*) – plot's file format. It can be pdf, png, ps or svg
- **hide** (*bool*) – hides plot window

static submit_ready_jobs(*as_conf*, *job_list*, *platforms_to_test*, *packages_persistence*, *inspect=False*, *only_wrappers=False*, *hold=False*)
Gets READY jobs and send them to the platforms if there is available space on the queues

Parameters

- **as_conf** (*AutosubmitConfig object*) – autosubmit config object
- **job_list** (*JobList object*) – job list to check
- **platforms_to_test** (*set of Platform Objects, e.g. SgePlatform(), LsfPlatform()*) – platforms used
- **packages_persistence** (*JobPackagePersistence object*) – Handles database per experiment.
- **inspect** (*Boolean*) – True if coming from generate_scripts_andor_wrappers().
- **only_wrappers** (*Boolean*) – True if it comes from create -cw, False if it comes from inspect -cw.

Returns True if at least one job was submitted, False otherwise

Return type Boolean

static test(*expid*, *chunks*, *member=None*, *start_date=None*, *hpc=None*, *branch=None*)
Method to conduct a test for a given experiment. It creates a new experiment for a given experiment with a given number of chunks with a random start date and a random member to be run on a random HPC.

Parameters

- **expid** (*str*) – experiment identifier
- **chunks** (*int*) – number of chunks to be run by the experiment
- **member** (*str*) – member to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **start_date** (*str*) – start date to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **hpc** (*str*) – HPC to be used by the test. If None, it uses a random one from which are defined on the experiment.
- **branch** (*str*) – branch or revision to be used by the test. If None, it uses configured branch.

Returns True if test was succesful, False otherwise

Return type bool

static testcase(*copy_id*, *description*, *chunks=None*, *member=None*, *start_date=None*, *hpc=None*, *branch=None*)
Method to create a test case. It creates a new experiment whose id starts by 't'.

Parameters

- **copy_id** (*str*) – experiment identifier
- **description** (*str*) – test case experiment description
- **chunks** (*int*) – number of chunks to be run by the experiment. If None, it uses configured chunk(s).
- **member** (*str*) – member to be used by the test. If None, it uses configured member(s).
- **start_date** (*str*) – start date to be used by the test. If None, it uses configured start date(s).
- **hpc** (*str*) – HPC to be used by the test. If None, it uses configured HPC.
- **branch** (*str*) – branch or revision to be used by the test. If None, it uses configured branch.

Returns test case id

Return type str

static unarchive(*experiment_id*, *uncompressed=True*)
Unarchives an experiment: uncompress folder from tar.gz and moves to experiments root folder

Parameters **experiment_id** (*str*) – experiment identifier

static update_version(*expid*)
Refresh experiment version with the current autosubmit version :param expid: experiment identifier :type expid: str

autosubmit.autosubmit.**signal_handler**(*signal_received*, *frame*)
Used to handle interrupt signals, allowing autosubmit to clean before exit

Parameters

- **signal_received** –
- **frame** –

autosubmit.autosubmit.**signal_handler_create**(*signal_received*, *frame*)
Used to handle KeyboardInterrumpt signals while the create method is being executed

Parameters

- **signal_received** –
- **frame** –

11.2 autosubmit.config

11.2.1 autosubmit.config.basicConfig

class autosubmit.config.basicConfig.**BasicConfig**
Class to manage configuration for Autosubmit path, database and default values for new experiments

static read()
Reads configuration from .autosubmitrc files, first from /etc, then for user directory and last for current path.

11.2.2 autosubmit.config.config_common

class autosubmit.config.config_common.**AutosubmitConfig**(*expid*, *basic_config*, *parser_factory*)

Bases: object

Class to handle experiment configuration coming from file or database

Parameters **expid** (*str*) – experiment identifier

check_autosubmit_conf()
Checks experiment's autosubmit configuration file.

Returns True if everything is correct, False if it founds any error

Return type bool

check_conf_files(*running_time=False*)
Checks configuration files (autosubmit, experiment jobs and platforms), looking for invalid values, missing required options. Prints results in log

Returns True if everything is correct, False if it finds any error

Return type bool

check_expdef_conf()
Checks experiment's experiment configuration file.

Returns True if everything is correct, False if it founds any error

Return type bool

check_jobs_conf()
Checks experiment's jobs configuration file.

Returns True if everything is correct, False if it founds any error

Return type bool

check_platforms_conf()
Checks experiment's queues configuration file.

check_proj()
Checks project config file

Returns True if everything is correct, False if it founds any error

Return type bool

check_proj_file()
Add a section header to the project's configuration file (if not exists)

experiment_file
Returns experiment's config file name

get_chunk_ini(*default=1*)
Returns the first chunk from where the experiment will start

Parameters **default** –

Returns initial chunk

Return type int

get_chunk_list()
Returns chunk list from experiment's config file

Returns experiment's chunks

Return type list

get_chunk_size(*default=1*)
Chunk Size as defined in the expdef file.

Returns Chunksize, 1 as default.

Return type int

get_chunk_size_unit()
Unit for the chunk length

Returns Unit for the chunk length Options: {hour, day, month, year}

Return type str

get_communications_library()
Returns the communications library from autosubmit's config file. Paramiko by default.

Returns communications library

Return type str

get_copy_remote_logs()
Returns if the user has enabled the logs local copy from autosubmit's config file

Returns if logs local copy

Return type bool

get_current_host(*section*)
Returns the user to be changed from platform config file.

Returns migrate user to

Return type str

get_current_project(*section*)
Returns the project to be changed from platform config file.

Returns migrate user to

Return type str

get_current_user(*section*)
Returns the user to be changed from platform config file.

Returns migrate user to

Return type str

get_custom_directives(*section*)
Gets custom directives needed for the given job type :param section: job type :type section: str :return: custom directives needed :rtype: str

get_date_list()
Returns startdates list from experiment's config file

Returns experiment's startdates

Return type list

get_default_job_type()
Returns the default job type from experiment's config file

Returns default type such as bash, python, r..

Return type str

get_disable_recovery_threads(*section*)
Returns FALSE/TRUE :return: recovery_threads_option :rtype: str

get_export(*section*)
Gets command line for being submitted with :param section: job type :type section: str :return: wallclock time :rtype: str

get_fetch_single_branch()
Returns fetch single branch from experiment's config file Default is -single-branch :return: fetch_single_branch(Y/N) :rtype: boolean

get_file_jobs_conf()
Returns path to project config file from experiment config file

Returns path to project config file

Return type str

get_file_project_conf()
Returns path to project config file from experiment config file

Returns path to project config file

Return type str

get_full_config_as_dict()
Returns full configuration as json object

get_full_config_as_json()
Return config as json object

get_git_project_branch()
Returns git branch from experiment's config file

Returns git branch

Return type str

get_git_project_commit()
Returns git commit from experiment's config file

Returns git commit

Return type str

get_git_project_origin()
Returns git origin from experiment config file

Returns git origin

Return type str

get_git_remote_project_root()
Returns remote machine ROOT PATH

Returns git commit

Return type str

get_jobs_sections()
Returns the list of sections defined in the jobs config file

Returns sections

Return type list

get_local_project_path()
Gets path to origin for local project

Returns path to local project

Return type str

get_mails_to()
Returns the address where notifications will be sent from autosubmit's config file

Returns mail address

Return type [str]

get_max_processors()
Returns max processors from autosubmit's config file

Return type str

get_max_waiting_jobs()
Returns max number of waiting jobs from autosubmit's config file

Returns main platforms

Return type int

get_max_wallclock()
Returns max wallclock

Return type str

get_max_wrapped_jobs()
Returns the maximum number of jobs that can be wrapped together as configured in autosubmit's config file

Returns maximum number of jobs (or total jobs)

Return type int

get_member_list(*run_only=False*)
Returns members list from experiment's config file

Returns experiment's members

Return type list

get_memory(*section*)
Gets memory needed for the given job type :param section: job type :type section: str :return: memory needed :rtype: str

get_memory_per_task(*section*)
Gets memory per task needed for the given job type :param section: job type :type section: str :return: memory per task needed :rtype: str

get_migrate_duplicate(*section*)
Returns the user to change to from platform config file.

Returns migrate user to

Return type str

get_migrate_host_to(*section*)
Returns the host to change to from platform config file.

Returns host_to

Return type str

get_migrate_project_to(*section*)
Returns the project to change to from platform config file.

Returns migrate project to

Return type str

get_migrate_user_to(*section*)
Returns the user to change to from platform config file.

Returns migrate user to

Return type str

get_min_wrapped_jobs()

Returns the minim number of jobs that can be wrapped together as configured in autosubmit's config file

Returns minim number of jobs (or total jobs)

Return type int

get_notifications()
Returns if the user has enabled the notifications from autosubmit's config file

Returns if notifications

Return type string

get_num_chunks()
Returns number of chunks to run for each member

Returns number of chunks

Return type int

get_output_type()
Returns default output type, pdf if none

Returns output type

Return type string

get_parse_two_step_start()
Returns two step start jobs

Returns jobs_list

Return type str

static get_parser(*parser_factory*, *file_path*)
Gets parser for given file

Parameters

- **parser_factory** –
- **file_path** (*str*) – path to file to be parsed

Returns parser

Return type SafeConfigParser

get_platform()
Returns main platforms from experiment's config file

Returns main platforms

Return type str

get_processors(*section*)
Gets processors needed for the given job type :param section: job type :type section: str :return: wallclock time :rtype: str

get_project_destination()
Returns git commit from experiment's config file

Returns git commit

Return type str

get_project_dir()
Returns experiment's project directory

Returns experiment's project directory

Return type str

get_project_type()
Returns project type from experiment config file

Returns project type

Return type str

get_remote_dependencies()
Returns if the user has enabled the PRESUBMISSION configuration parameter from autosubmit's config file

Returns if remote dependencies

Return type bool

get_rerun()
Returns startdates list from experiment's config file

Returns rerurn value

Return type list

get_retrials()
Returns max number of retrials for job from autosubmit's config file

Returns safety sleep time

Return type int

get_safetysleeptime()
Returns safety sleep time from autosubmit's config file

Returns safety sleep time

Return type int

get_scratch_free_space(*section*)
Gets scratch free space needed for the given job type :param section: job type :type section: str :return: percentage of scratch free space needed :rtype: int

get_storage_type()
Returns the communications library from autosubmit's config file. Paramiko by default.

Returns communications library

Return type str

get_submodules_list()
Returns submodules list from experiment's config file Default is –recursive :return: submodules to load :rtype: list

get_svn_project_revision()
Get revision for subversion project

Returns revision for subversion project

Return type str

get_svn_project_url()
Gets subversion project url

Returns subversion project url

Return type str

get_synchronize(*section*)
Gets wallclock for the given job type :param section: job type :type section: str :return: wallclock time :rtype: str

get_tasks(*section*)
Gets tasks needed for the given job type :param section: job type :type section: str :return: tasks (processes) per host :rtype: str

get_threads(*section*)
Gets threads needed for the given job type :param section: job type :type section: str :return: threads needed :rtype: str

get_total_jobs()
Returns max number of running jobs from autosubmit's config file

Returns max number of running jobs

Return type int

get_version()
Returns version number of the current experiment from autosubmit's config file

Returns version

Return type str

get_wallclock(*section*)
Gets wallclock for the given job type :param section: job type :type section: str :return: wallclock time :rtype: str

get_wchunkinc(*section*)
Gets the chunk increase to wallclock :param section: job type :type section: str :return: wallclock increase per chunk :rtype: str

get_wrapper_check_time()
Returns time to check the status of jobs in the wrapper

Returns wrapper check time

Return type int

get_wrapper_export()
Returns modules variable from wrapper

Returns string

Return type string

get_wrapper_jobs()
Returns the jobs that should be wrapped, configured in the autosubmit's config

Returns expression (or none)

Return type string

get_wrapper_machinefiles()
Returns the strategy for creating the machinefiles in wrapper jobs

Returns machinefiles function to use

Return type string

get_wrapper_method()
Returns the method of make the wrapper

Returns method

Return type string

get_wrapper_policy()
Returns what kind of wrapper (VERTICAL, MIXED-VERTICAL, HORIZONTAL, HYBRID, NONE) the user has configured in the autosubmit's config

Returns wrapper type (or none)

Return type string

get_wrapper_queue()
Returns the wrapper queue if not defined, will be the one of the first job wrapped

Returns expression (or none)

Return type string

get_wrapper_type()
Returns what kind of wrapper (VERTICAL, MIXED-VERTICAL, HORIZONTAL, HYBRID, NONE) the user has configured in the autosubmit's config

Returns wrapper type (or none)

Return type string

jobs_file
Returns project's jobs file name

load_parameters()
Load parameters from experiment and autosubmit config files. If experiment's type is not none, also load parameters from model's config file

Returns a dictionary containing tuples [parameter_name, parameter_value]

Return type dict

load_platform_parameters()
Load parameters from platform config files.

Returns a dictionary containing tuples [parameter_name, parameter_value]

Return type dict

load_project_parameters()
Loads parameters from model config file

Returns dictionary containing tuples [parameter_name, parameter_value]

Return type dict

load_section_parameters(*job_list*, *as_conf*, *submitter*)
Load parameters from job config files.

Returns a dictionary containing tuples [parameter_name, parameter_value]

Return type dict

platforms_file
Returns experiment's platforms config file name

Returns platforms config file's name

Return type str

platforms_parser
Returns experiment's platforms parser object

Returns platforms config parser object

Return type SafeConfigParser

project_file
Returns project's config file name

reload()
Creates parser objects for configuration files

set_expid(*exp_id*)
Set experiment identifier in autosubmit and experiment config files

Parameters **exp_id** (*str*) – experiment identifier to store

set_git_project_commit(*as_conf*)
Function to register in the configuration the commit SHA of the git project version. :param as_conf: Configuration class for exteriment :type as_conf: AutosubmitConfig

set_new_host(*section*, *new_host*)
Sets new host for given platform :param new_host: :param section: platform name :type: str

set_new_project(*section*, *new_project*)
Sets new project for given platform :param new_project: :param section: platform name :type: str

set_new_user(*section*, *new_user*)
Sets new user for given platform :param new_user: :param section: platform name :type: str

set_platform(*hpc*)
Sets main platforms in experiment's config file

Parameters **hpc** – main platforms

Type str

set_safetysleeptime(*sleep_time*)
Sets autosubmit's version in autosubmit's config file

Parameters **sleep_time** (*int*) – value to set

set_version(*autosubmit_version*)
Sets autosubmit's version in autosubmit's config file

Parameters **autosubmit_version** (*str*) – autosubmit's version

11.3 autosubmit.database

Module containing functions to manage autosubmit's database.

exception autosubmit.database.db_common.**DbException**(*message*)
Exception class for database errors

autosubmit.database.db_common.**check_db**()
Checks if database file exist

Returns None if exists, terminates program if not

autosubmit.database.db_common.**check_experiment_exists**(*name*, *error_on_inexistence=True*)
Checks if exist an experiment with the given name.

Parameters

- **error_on_inexistence** (*bool*) – if True, adds an error log if experiment does not exists
- **name** (*str*) – Experiment name

Returns If experiment exists returns true, if not returns false

Return type bool

autosubmit.database.db_common.**close_conn**(*conn*, *cursor*)
Commits changes and close connection to database

Parameters

- **conn** (*sqlite3.Connection*) – connection to close
- **cursor** (*sqlite3.Cursor*) – cursor to close

autosubmit.database.db_common.**create_db**(*qry*)
Creates a new database for autosubmit

Parameters **qry** (*str*) – query to create the new database

autosubmit.database.db_common.**delete_experiment**(*experiment_id*)
Removes experiment from database

Parameters **experiment_id** (*str*) – experiment identifier

Returns True if delete is succesful

Return type bool

autosubmit.database.db_common.**get_autosubmit_version**(*expid*)
Get the minimun autosubmit version needed for the experiment

Parameters **expid** (*str*) – Experiment name

Returns If experiment exists returns the autosubmit version for it, if not returns None

Return type str

autosubmit.database.db_common.**last_name_used**(*test=False*, *operational=False*)
Gets last experiment identifier used

Parameters

- **test** (*bool*) – flag for test experiments
- **operational** – flag for operational experiments

Returns last experiment identifier used, 'empty' if there is none

Return type str

autosubmit.database.db_common.**open_conn**(*check_version=True*)
Opens a connection to database

Parameters **check_version** (*bool*) – If true, check if the database is compatible with this autosubmit version

Returns connection object, cursor object

Return type sqlite3.Connection, sqlite3.Cursor

autosubmit.database.db_common.**save_experiment**(*name*, *description*, *version*)
Stores experiment in database

Parameters

- **version** (*str*) –
- **name** (*str*) – experiment's name
- **description** (*str*) – experiment's description

autosubmit.database.db_common.**update_experiment_descrip_version**(*name*, *description=None*, *version=None*)
Updates the experiment's description and/or version

Parameters

- **name** – experiment name (expid)
- **description** – experiment new description
- **version** – experiment autosubmit version

Rtype name str

Rtype description str

Rtype version str

Returns If description has been update, True; otherwise, False.

Return type bool

11.4 autosubmit.git

class autosubmit.git.autosubmit_git.**AutosubmitGit**(*expid*)
Class to handle experiment git repository

Parameters **expid** (*str*) – experiment identifier

static check_commit(*as_conf*)
Function to check uncommited changes

Parameters **as_conf** (*autosubmit.config.AutosubmitConfig*) – experiment configuration

static clean_git(*as_conf*)
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/git directory.

Parameters **as_conf** (*autosubmit.config.AutosubmitConfig*) – experiment configuration

static clone_repository(*as_conf*, *force*, *hpcarch*)
Clones a specified git repository on the project folder

Parameters

- **as_conf** (*autosubmit.config.AutosubmitConfig*) – experiment configuration
- **force** (*bool*) – if True, it will overwrite any existing clone
- **hpcarch** – current main platform

Returns True if clone was successful, False otherwise

11.5 autosubmit.job

Main module for Autosubmit. Only contains an interface class to all functionality implemented on Autosubmit

class autosubmit.job.job.**Job**(*name*, *job_id*, *status*, *priority*)
Class to handle all the tasks with Jobs at HPC. A job is created by default with a name, a jobid, a status and a type. It can have children and parents. The inheritance reflects the dependency between jobs. If Job2 must wait until Job1 is completed then Job2 is a child of Job1. Inversely Job1 is a parent of Job2

Parameters

- **name** (*str*) – job's name
- **jobid** (*int*) – job's identifier
- **status** (Status) – job initial status
- **priority** (*int*) – job's priority

add_parent(**parents*)
Add parents for the job. It also adds current job as a child for all the new parents

Parameters **parents** (*Job) – job's parents to add

check_completion(*default_status=-1*)
Check the presence of *COMPLETED* file. Change status to COMPLETED if *COMPLETED* file exists and to FAILED otherwise. :param default_status: status to set if job is not completed. By default is FAILED :type default_status: Status

check_end_time()
Returns end time from stat file

Returns date and time

Return type str

check_retrials_end_time()
Returns list of end datetime for retrials from total stats file

Returns date and time

Return type list[int]

check_retrials_start_time()
Returns list of start datetime for retrials from total stats file

Returns date and time

Return type list[int]

check_retrials_submit_time()
Returns list of submit datetime for retrials from total stats file

Returns date and time

Return type list[int]

check_running_after(*date_limit*)
Checks if the job was running after the given date :param date_limit: reference date :type date_limit: datetime.datetime :return: True if job was running after the given date, false otherwise :rtype: bool

check_script(*as_conf*, *parameters*, *show_logs=False*)
Checks if script is well formed

Parameters

- **parameters** (*dict*) – script parameters
- **as_conf** (AutosubmitConfig) – configuration file
- **show_logs** (*Bool*) – Display output

Returns true if not problem has been detected, false otherwise

Return type bool

check_start_time()
Returns job's start time

Returns start time

Return type str

check_started_after(*date_limit*)
Checks if the job started after the given date :param date_limit: reference date :type date_limit: datetime.datetime :return: True if job started after the given date, false otherwise :rtype: bool

children
Returns a list containing all children of the job

Returns child jobs

Return type set

compare_by_id(*other*)
Compare jobs by ID

Parameters **other** (Job) – job to compare

Returns comparison result

Return type bool

compare_by_name(*other*)
Compare jobs by name

Parameters **other** (Job) – job to compare

Returns comparison result

Return type bool

compare_by_status(*other*)
Compare jobs by status value

Parameters **other** (Job) – job to compare

Returns comparison result

Return type bool

create_script(*as_conf*)
Creates script file to be run for the job

Parameters **as_conf** (AutosubmitConfig) – configuration object

Returns script's filename

Return type str

delete_child(*child*)
Removes a child from the job

Parameters **child** (Job) – child to remove

delete_parent(*parent*)
Remove a parent from the job

Parameters **parent** (Job) – parent to remove

get_last_retrials()
Returns the retrials of a job, including the last COMPLETED run. The selection stops, and does not include, when the previous COMPLETED job is located or the list of registers is exhausted.

Returns list of list of dates of retrial [submit, start, finish] in datetime format

Return type list of list

has_children()
Returns true if job has any children, else return false

Returns true if job has any children, otherwise return false

Return type bool

has_parents()
Returns true if job has any parents, else return false

Returns true if job has any parent, otherwise return false

Return type bool

inc_fail_count()
Increments fail count

static is_a_completed_retrial(*fields*)
Returns true only if there 4 fields: submit start finish status, and status equals COMPLETED.

is_ancestor(*job*)
Check if the given job is an ancestor :param job: job to be checked if is an ancestor :return: True if job is an ancestor, false otherwise :rtype bool

is_parent(*job*)
Check if the given job is a parent :param job: job to be checked if is a parent :return: True if job is a parent, false otherwise :rtype bool

log_job()
Prints job information in log

long_name
Job's long name. If not setted, returns name

Returns long name

Return type str

parents
Returns parent jobs list

Returns parent jobs

Return type set

platform
Returns the platform to be used by the job. Chooses between serial and parallel platforms

:return HPCPlatform object for the job to use :rtype: HPCPlatform

print_job()
Prints debug information about the job

print_parameters()
Print sjob parameters in log

queue
Returns the queue to be used by the job. Chooses between serial and parallel platforms

:return HPCPlatform object for the job to use :rtype: HPCPlatform

remove_redundant_parents()
Checks if a parent is also an ancestor, if true, removes the link in both directions. Useful to remove redundant dependencies.

total_processors
Number of processors requested by job. Reduces ':' separated format if necessary.

update_content(*as_conf*)
Create the script content to be run for the job

Parameters **as_conf** (*config*) – config

Returns script code

Return type str

update_parameters(*as_conf*, *parameters*, *default_parameters={'M': '%M%', 'M_': '%M_%', 'Y': '%Y%', 'Y_': '%Y_%', 'd': '%d%', 'd_': '%d_%', 'm': '%m%', 'm_': '%m_%'}*)
Refresh parameters value

Parameters

- **default_parameters** (*dict*) –

- **as_conf** (AutosubmitConfig) –
- **parameters** (*dict*) –

update_status(*copy_remote_logs=False*, *failed_file=False*)
Updates job status, checking COMPLETED file if needed

Parameters

- **new_status** – job status retrieved from the platform
- **copy_remote_logs** – should copy remote logs when finished?

Type *Status*

write_end_time(*completed*)
Writes ends date and time to TOTAL_STATS file :param completed: True if job was completed successfully, False otherwise :type completed: bool

write_start_time()
Writes start date and time to TOTAL_STATS file :return: True if succesful, False otherwise :rtype: bool

write_submit_time()
Writes submit date and time to TOTAL_STATS file

class autosubmit.job.job.**WrapperJob**(*name*, *job_id*, *status*, *priority*, *job_list*, *total_wallclock*, *num_processors*, *platform*, *as_config*, *hold*)
Defines a wrapper from a package.

Calls Job constructor.

Parameters

- **name** (*String*) – Name of the Package
- **job_id** (*Integer*) – Id of the first Job of the package
- **status** (*String*) – 'READY' when coming from submit_ready_jobs()
- **priority** (*Integer*) – 0 when coming from submit_ready_jobs()
- **job_list** (*List() of Job() objects*) – List of jobs in the package
- **total_wallclock** (*String Formatted*) – Wallclock of the package
- **num_processors** (*Integer*) – Number of processors for the package
- **platform** (*Platform Object. e.g. EcPlatform()*) – Platform object defined for the package
- **as_config** (*AutosubmitConfig object*) – Autosubmit basic configuration object

class autosubmit.job.job_common.**StatisticsSnippetBash**
Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetEmpty**
Class to handle the statistics snippet of a job. It contains header and footer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetPython**
Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetR**
Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**Status**
Class to handle the status of a job

class autosubmit.job.job_common.**Type**
Class to handle the status of a job

autosubmit.job.job_common.**increase_wallclock_by_chunk**(*current*, *increase*, *chunk*)
Receives the wallclock times an increases it according to a quantity times the number of the current chunk. The result cannot be larger than 48:00. If Chunk = 0 then no increment.

Parameters

- **current** (*str*) – WALLCLOCK HH:MM
- **increase** (*str*) – WCHUNKINC HH:MM
- **chunk** (*int*) – chunk number

Returns HH:MM wallclock

Return type str

autosubmit.job.job_common.**parse_output_number**(*string_number*)
Parses number in format 1.0K 1.0M 1.0G

Parameters **string_number** (*str*) – String representation of number

Returns number in float format

Return type float

class autosubmit.job.job_list.**JobList**(*expid*, *config*, *parser_factory*, *job_list_persistence*)
Class to manage the list of jobs to be run by autosubmit

add_logs(*logs*)
add logs to the current job_list

Parameters **platform** (*HPCPlatform*) – job platform

Returns logs

Return type dict(tuple)

backup_load()
Recreates an stored job list from the persistence

Returns loaded job list object

Return type *JobList*

backup_save()
Persists the job list

check_scripts(*as_conf*)
When we have created the scripts, all parameters should have been substituted. %PARAMETER% handlers not allowed

Parameters **as_conf** (AutosubmitConfig) – experiment configuration

expid
Returns the experiment identifier

Returns experiment's identifier

Return type str

generate(*date_list*, *member_list*, *num_chunks*, *chunk_ini*, *parameters*, *date_format*, *default_retrials*, *default_job_type*, *wrapper_type=None*, *wrapper_jobs=None*, *new=True*, *notransitive=False*, *update_structure=False*, *run_only_members=[]*)
Creates all jobs needed for the current workflow

Parameters

- **default_job_type** (*str*) – default type for jobs
- **date_list** (*list*) – start dates
- **member_list** (*list*) – members
- **num_chunks** (*int*) – number of chunks to run
- **chunk_ini** (*int*) – the experiment will start by the given chunk
- **parameters** (*dict*) – parameters for the jobs
- **date_format** (*str*) – option to format dates
- **default_retrials** (*int*) – default retrials for ech job
- **new** (*bool*) – is it a new generation?
- **wrapper_type** – Type of wrapper defined by the user in **autosubmit_**.conf [wrapper] section.
- **wrapper_jobs** (*String*) – Job types defined in **autosubmit_**.conf [wrapper sections] to be wrapped.

get_active(*platform=None*, *wrapper=False*)
Returns a list of active jobs (In platforms queue + Ready)

Parameters **platform** (*HPCPlatform*) – job platform

Returns active jobs

Return type list

get_all(*platform=None*, *wrapper=False*)
Returns a list of all jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns all jobs

Return type list

get_chunk_list()
Get inner chunk list

Returns chunk list

Return type list

get_completed(*platform=None*, *wrapper=False*)
Returns a list of completed jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns completed jobs

Return type list

get_date_list()
Get inner date list

Returns date list

Return type list

get_failed(*platform=None*, *wrapper=False*)
Returns a list of failed jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns failed jobs

Return type list

get_finished(*platform=None*, *wrapper=False*)
Returns a list of jobs finished (Completed, Failed)

Parameters **platform** (*HPCPlatform*) – job platform

Returns finished jobs

Return type list

get_held_jobs(*platform=None*)
Returns a list of jobs in the platforms (Held)

Parameters **platform** (*HPCPlatform*) – job platform

Returns jobs in platforms

Return type list

get_in_queue(*platform=None*, *wrapper=False*)
Returns a list of jobs in the platforms (Submitted, Running, Queuing, Unknown,Held)

Parameters **platform** (*HPCPlatform*) – job platform

Returns jobs in platforms

Return type list

get_job_by_name(*name*)
Returns the job that its name matches parameter name

Parameters **name** (*str*) – name to look for

Returns found job

Return type job

get_job_list()
Get inner job list

Returns job list

Return type list

get_job_names(*lower_case=False*)
Returns a list of all job names

Parameters **platform** (*HPCPlatform*) – job platform

Returns all jobs

Return type list

get_job_related(*select_jobs_by_name=''*, *select_all_jobs_by_section=''*, *filter_jobs_by_section=''*)

Parameters

- **select_jobs_by_name** – job name
- **select_all_jobs_by_section** – section name
- **filter_jobs_by_section** – section, date , member? , chunk?

Returns jobs_list names

Return type list

get_logs()
Returns a dict of logs by jobs_name jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns logs

Return type dict(tuple)

get_member_list()
Get inner member list

Returns member list

Return type list

get_not_in_queue(*platform=None*, *wrapper=False*)
Returns a list of jobs NOT in the platforms (Ready, Waiting)

Parameters **platform** (*HPCPlatform*) – job platform

Returns jobs not in platforms

Return type list

get_ordered_jobs_by_date_member()
Get the dictionary of jobs ordered according to wrapper's expression divided by date and member

Returns jobs ordered divided by date and member

Return type dict

get_prepared(*platform=None*)
Returns a list of prepared jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns prepared jobs

Return type list

get_queuing(*platform=None*, *wrapper=False*)
Returns a list of jobs queuing

Parameters **platform** (*HPCPlatform*) – job platform

Returns queuedjobs

Return type list

get_ready(*platform=None*, *hold=False*, *wrapper=False*)
Returns a list of ready jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns ready jobs

Return type list

get_running(*platform=None*, *wrapper=False*)
Returns a list of jobs running

Parameters **platform** (*HPCPlatform*) – job platform

Returns running jobs

Return type list

get_skipped(*platform=None*)
Returns a list of prepared jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns prepared jobs

Return type list

get_submitted(*platform=None*, *hold=False*, *wrapper=False*)
Returns a list of submitted jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns submitted jobs

Return type list

get_suspended(*platform=None*, *wrapper=False*)
Returns a list of jobs on unknown state

Parameters **platform** (*HPCPlatform*) – job platform

Returns unknown state jobs

Return type list

get_uncompleted(*platform=None*, *wrapper=False*)
Returns a list of completed jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns completed jobs

Return type list

get_unknown(*platform=None*, *wrapper=False*)
Returns a list of jobs on unknown state

Parameters **platform** (*HPCPlatform*) – job platform

Returns unknown state jobs

Return type list

get_unsubmitted(*platform=None*, *wrapper=False*)
Returns a list of unsummited jobs

Parameters **platform** (*HPCPlatform*) – job platform

Returns all jobs

Return type list

get_waiting(*platform=None*, *wrapper=False*)
Returns a list of jobs waiting

Parameters **platform** (*HPCPlatform*) – job platform

Returns waiting jobs

Return type list

get_waiting_remote_dependencies(*platform_type='slurm'*)
Returns a list of jobs waiting on slurm scheduler

Parameters **platform** (*HPCPlatform*) – job platform

Returns waiting jobs

Return type list

graph
Returns the graph

Returns graph

Return type networkx graph

load()
Recreates an stored job list from the persistence

Returns loaded job list object

Return type *JobList*

static load_file(*filename*)
Recreates an stored joblist from the pickle file

Parameters **filename** (*str*) – pickle file to load

Returns loaded joblist object

Return type *JobList*

parameters
List of parameters common to all jobs :return: parameters :rtype: dict

print_with_status(*statusChange=None*, *nocolor=False*, *existingList=None*)
Returns the string representation of the dependency tree of the Job List

Parameters

- **statusChange** (*List of strings*) – List of changes in the list, supplied in set status
- **nocolor** (*Boolean*) – True if the result should not include color codes
- **existingList** (*List of Job Objects*) – External List of Jobs that will be printed, this excludes the inner list of jobs.

Returns String representation

Return type String

remove_rerun_only_jobs(*notransitive=False*)
Removes all jobs to be run only in reruns

rerun(*chunk_list*, *notransitive=False*, *monitor=False*)
Updates job list to rerun the jobs specified by chunk_list

Parameters **chunk_list** (*str*) – list of chunks to rerun

save()
Persists the job list

sort_by_id()
Returns a list of jobs sorted by id

Returns jobs sorted by ID

Return type list

sort_by_name()
Returns a list of jobs sorted by name

Returns jobs sorted by name

Return type list

sort_by_status()
Returns a list of jobs sorted by status

Returns job sorted by status

Return type list

sort_by_type()
Returns a list of jobs sorted by type

Returns job sorted by type

Return type list

update_from_file(*store_change=True*)
Updates jobs list on the fly from and update file :param store_change: if True, renames the update file to avoid reloading it at the next iteration

update_genealogy(*new=True*, *notransitive=False*, *update_structure=False*)
When we have created the job list, every type of job is created. Update genealogy remove jobs that have no templates :param new: if it is a new job list or not :type new: bool

update_list(*as_conf*, *store_change=True*, *fromSetStatus=False*, *submitter=None*, *first_time=False*)
Updates job list, resetting failed jobs and changing to READY all WAITING jobs with all parents COMPLETED

Parameters **as_conf** (AutosubmitConfig) – autosubmit config object

Returns True if job status were modified, False otherwise

Return type bool

11.6 autosubmit.monitor

class autosubmit.monitor.monitor.**Monitor**
Class to handle monitoring of Jobs at HPC.

static clean_plot(*expid*)
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all plots except last two.

Parameters **expid** (*str*) – experiment's identifier

static clean_stats(*expid*)
Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all stats' plots except last two.

Parameters **expid** (*str*) – experiment's identifier

static color_status(*status*)
Return color associated to given status

Parameters **status** (Status) – status

Returns color

Return type str

create_tree_list(*expid*, *joblist*, *packages*, *groups*, *hide_groups=False*)
Create graph from joblist

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (JobList) – joblist to plot

Returns created graph

Return type pydotplus.Dot

generate_output(*expid*, *joblist*, *path*, *output_format='pdf'*, *packages=None*, *show=False*, *groups={}*, *hide_groups=False*, *job_list_object=None*)

Plots graph for joblist and stores it in a file

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*List of Job objects*) – list of jobs to plot
- **output_format** (*str (png, pdf, ps)*) – file format for plot
- **show** (*bool*) – if true, will open the new plot with the default viewer
- **job_list_object** (*JobList object*) – Object that has the main txt generation method

generate_output_stats(*expid*, *joblist*, *output_format='pdf'*, *period_ini=None*, *period_fi=None*, *show=False*)

Plots stats for joblist and stores it in a file

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (JobList) – joblist to plot
- **output_format** (*str (png, pdf, ps)*) – file format for plot
- **period_ini** (*datetime*) – initial datetime of filtered period
- **period_fi** (*datetime*) – final datetime of filtered period
- **show** (*bool*) – if true, will open the new plot with the default viewer

generate_output_txt(*expid*, *joblist*, *path*, *classictxt=False*, *job_list_object=None*)

Function that generates a representation of the jobs in a txt file :param expid: experiment's identifier :type expid: str :param joblist: experiment's list of jobs :type joblist: list :param job_list_object: Object that has the main txt generation method :type job_list_object: JobList object

static get_general_stats(*expid*)

Returns all the options in the sections of the %expid%_GENERAL_STATS

Parameters **expid** (*str*) – experiment's identifier

Returns list of tuples (section, ''), (option, value), (option, value), (section, ''), (option, value), ...

Return type list

11.7 autosubmit.platform

class autosubmit.platforms.ecplatform.**EcPlatform**(*expid*, *name*, *config*, *scheduler*)

Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage queues with ecaccess

Parameters

- **expid** (*str*) – experiment's identifier
- **scheduler** (*str (pbs, loadleveler)*) – scheduler to use

connect()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

delete_file(*filename*)
Deletes a file from this platform

Parameters **filename** (*str*) – file name

Returns True if successful or file does no exists

Return type bool

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_file(*filename*, *must_exist=True*, *relative_path=''*, *ignore_log=False*, *wrapper_failed=False*)
Copies a file from the current platform to experiment's tmp folder

Parameters

- **filename** (*str*) – file name
- **must_exist** (*bool*) – If True, raises an exception if file can not be copied
- **relative_path** (*str*) – path inside the tmp folder

Returns True if file is copied successfully, false otherwise

Return type bool

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script
- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

jobs_in_queue()
Returns empty list because ecacces does not support this command

Returns empty list

Return type list

move_file(*src*, *dest*, *must_exist=False*)
Moves a file on the platform (includes .err and .out) :param src: source name :type src: str :param dest: destination name :param must_exist: ignore if file exist or not :type dest: str

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

restore_connection()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

send_command(*command*, *ignore_log=False*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*filename*, *check=True*)
Sends a local file to the platform :param filename: name of the file to send :type filename: str

test_connection()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

update_cmds()
Updates commands for platforms

class autosubmit.platforms.lsfplatform.**LsfPlatform**(*expid*, *name*, *config*)
Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage jobs to host using LSF scheduler

Parameters **expid** (*str*) – experiment's identifier

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_submit_cmd(*job_script*, *job*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script
- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

update_cmds()
Updates commands for platforms

class autosubmit.platforms.pbsplatform.**PBSPlatform**(*expid*, *name*, *config*, *version*)
Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage jobs to host using PBS scheduler

Parameters

- **expid** (*str*) – experiment's identifier
- **version** (*str*) – scheduler version

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_submit_cmd(*job_script*, *job*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script
- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

update_cmds()
Updates commands for platforms

class autosubmit.platforms.sgeplatform.**SgePlatform**(*expid*, *name*, *config*)
Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage jobs to host using SGE scheduler

Parameters **expid** (*str*) – experiment's identifier

connect()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_submit_cmd(*job_script*, *job*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script
- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

restore_connection()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

test_connection()
In this case, it does nothing because connection is established for each command

Returns True

Return type bool

update_cmds()
Updates commands for platforms

class autosubmit.platforms.slurmplatform.**SlurmPlatform**(*expid*, *name*, *config*)
Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage jobs to host using SLURM scheduler

Parameters **expid** (*str*) – experiment's identifier

get_checkAlljobs_cmd(*jobs_id*)
Returns command to check jobs status on remote platforms

Parameters

- **jobs_id** – id of jobs to check
- **job_id** – str

Returns command to check job status

Return type str

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script

- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*outputlines*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

parse_Alljobs_output(*output*, *job_id*)
Parses check jobs command output so it can be interpreted by autosubmit :param output: output to parse :param job_id: select the job to parse :type output: str :return: job status :rtype: str

parse_job_finish_data(*output*, *packed*)
Parses the context of the sacct query to SLURM for a single job. Only normal jobs return submit, start, finish, joules, ncpus, nnodes.

When a wrapper has finished, capture finish time.

Parameters

- **output** (*str*) – The sacct output
- **job_id** (*int*) – Id in SLURM for the job
- **packed** (*bool*) – true if job belongs to package

Returns submit, start, finish, joules, ncpus, nnodes, detailed_data

Return type int, int, int, int, int, int, json object (str)

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

submit_Script(*hold=False*)
Sends a Submit file Script, execute it in the platform and retrieves the Jobs_ID of all jobs at once.

Parameters **job** (autosubmit.job.job.Job) – job object

Returns job id for submitted jobs

Return type list(str)

update_cmds()
Updates commands for platforms

class autosubmit.platforms.locplatform.**LocalPlatform**(*expid*, *name*, *config*)
Bases: autosubmit.platforms.paramiko_platform.ParamikoPlatform

Class to manage jobs to localhost

Parameters **expid** (*str*) – experiment's identifier

check_file_exists(*src*, *wrapper_failed=False*)
Moves a file on the platform :param src: source name :type src: str :param dest: destination name :param must_exist: ignore if file exist or not :type dest: str

connect()
Creates ssh connection to host

Returns True if connection is created, False otherwise

Return type bool

delete_file(*filename*, *del_cmd=False*)
Deletes a file from this platform

Parameters **filename** (*str*) – file name

Returns True if successful or file does no exists

Return type bool

get_checkjob_cmd(*job_id*)
Returns command to check job status on remote platforms

Parameters

- **job_id** – id of job to check
- **job_id** – int

Returns command to check job status

Return type str

get_file(*filename*, *must_exist=True*, *relative_path=''*, *ignore_log=False*, *wrapper_failed=False*)
Copies a file from the current platform to experiment's tmp folder

Parameters

- **filename** (*str*) – file name
- **must_exist** (*bool*) – If True, raises an exception if file can not be copied
- **relative_path** (*str*) – path inside the tmp folder

Returns True if file is copied successfully, false otherwise

Return type bool

get_logs_files(*exp_id*, *remote_logs*)
Overriding the parent's implementation. Do nothing because the log files are already in the local platform (redundancy).

Parameters

- **exp_id** (*str*) – experiment id
- **remote_logs** ((*str, str*)) – names of the log files

get_mkdir_cmd()
Gets command to create directories on HPC

Returns command to create directories on HPC

Return type str

get_ssh_output()
Gets output from last command executed

Returns output from last command

Return type str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)
Get command to add job to scheduler

Parameters

- **job_type** –
- **job_script** – path to job script
- **job_script** – str
- **hold** – submit a job in a held status
- **hold** – boolean
- **export** – modules that should've downloaded
- **export** – string

Returns command to submit job to platforms

Return type str

get_submitted_job_id(*output*)
Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

move_file(*src*, *dest*, *must_exist=False*)
Moves a file on the platform (includes .err and .out) :param src: source name :type src: str :param dest: destination name :param must_exist: ignore if file exist or not :type dest: str

parse_job_output(*output*)
Parses check job command output so it can be interpreted by autosubmit

Parameters **output** (*str*) – output to parse

Returns job status

Return type str

send_command(*command*, *ignore_log=False*)
Sends given command to HPC

Parameters **command** (*str*) – command to send

Returns True if executed, False if failed

Return type bool

send_file(*filename*)
Sends a local file to the platform :param filename: name of the file to send :type filename: str

test_connection()
Test if the connection is still alive, reconnect if not.

update_cmds()
Updates commands for platforms

CHAPTER 12

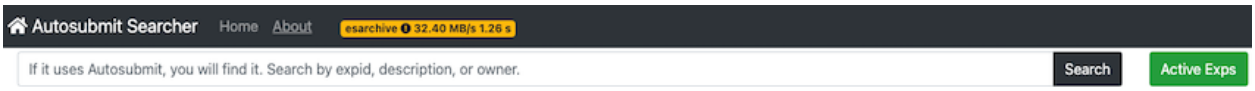
Autosubmit GUI

12.1 Autosubmit GUI Main Page

Inside the Barcelona Supercomputing Internal Network you can find the latest version of Autosubmit GUI deployed for BSC users. It can be accessed by following the url http://bscesweb04.bsc.es/autosubmitapp/ or https://earth.bsc.es/autosubmitapp/. This is a graphic user interface that allows you to easily monitor your experiments and those of your colleagues. This Web App introduces many useful features for experiment monitoring, and we are continuously improving it.

Note: The Web App can also be accessed through the VPN Client provided by BSC.

When you enter the site, you will be presented with the following page:

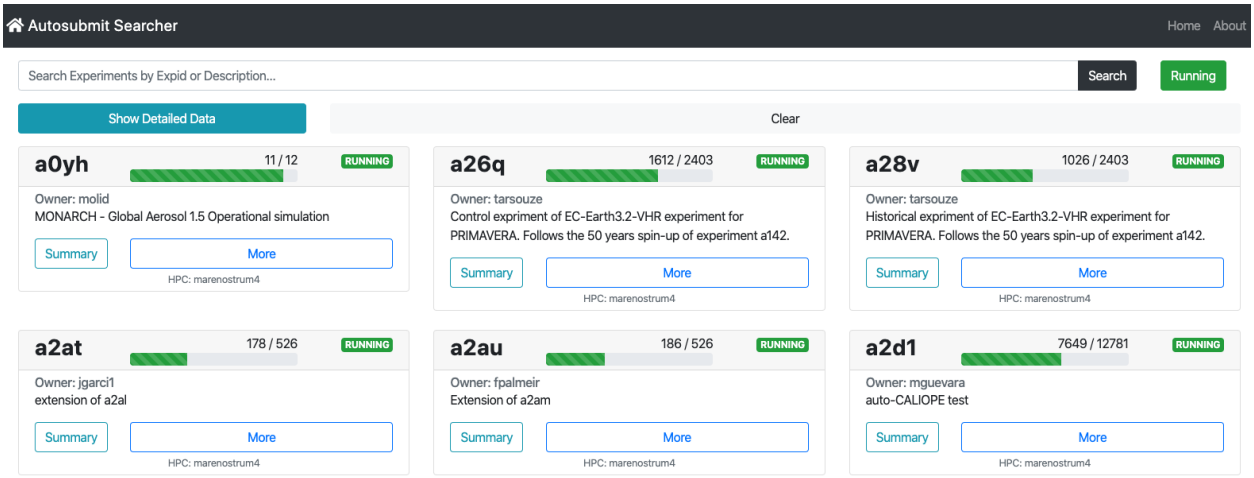


12.1: Welcome page

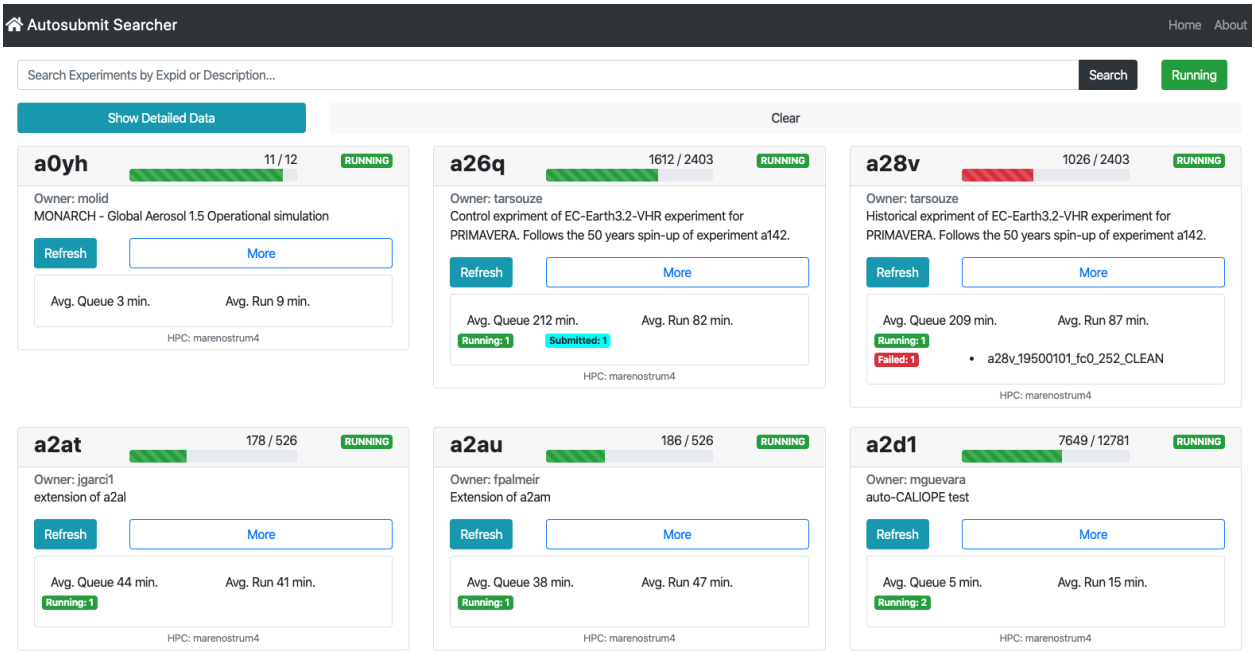
Here you can search for any ongoing or past experiment by typing some text in the Search input box and pressing **Search**: the search engine will look for coincidences between your input string and any of the description, owner or name of the experiment fields. The results will be shown below ordered by status, experiments `RUNNING` will be shown in the first rows. You can also click on the `Running` button, and all the experiments that are currently running will be listed. The results will look like:

If you click on `Show Detailed Data`, summary data for each experiment (result) will be loaded. These are data details from the experiment run, useful to see its status at a glance. Progress bars and status will use different colors to highlight the important information.

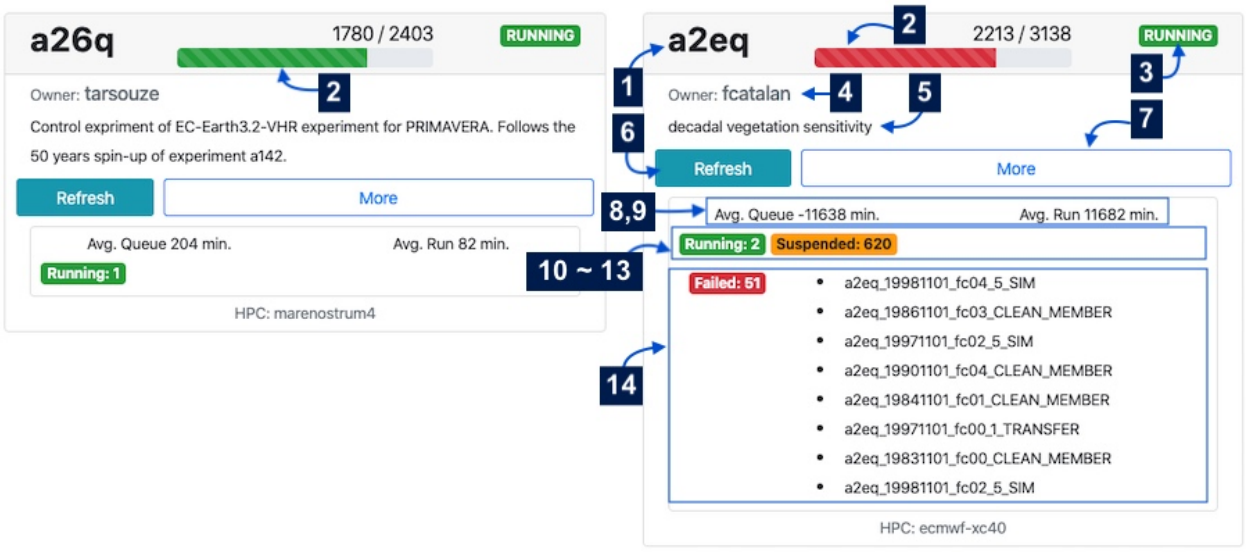
For each experiment, you see the following data:



12.2: Search Result



12.3: Search Result plus Detailed Data

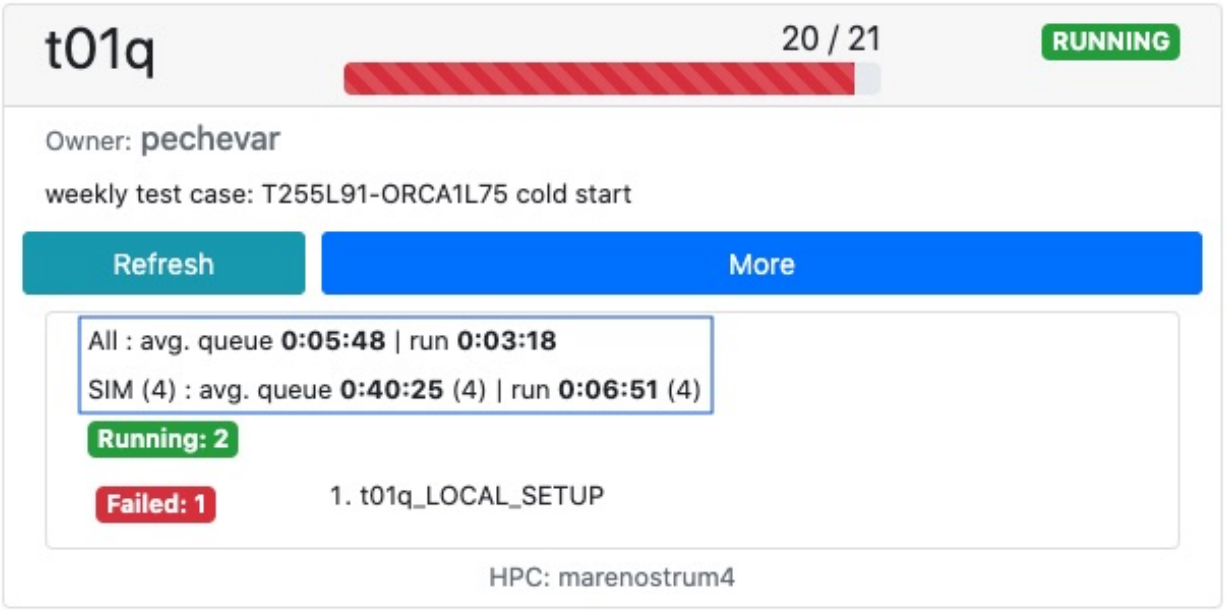


12.4: Description of Detailed Data

1. *Experiment Name*
2. *Progress Bar*: Shows completed jobs / total jobs. It turns red when there are **failed** jobs in the experiment, but **Show Detailed Data** should have been requested.
3. *Experiment Status*: *RUNNING* or *NOT RUNNING*.
4. *Owner*
5. *Experiment Description*
6. *Refresh button*: It will say *Summary* when the detailed data has not been requested. If it says *Summary* and you click on it, it will load datailed data for that experiment, otherwise it will refresh the existing detailed data.
7. *More button*: Opens the **Experiment Page**.
8. *Average Queue Time* for all jobs.
9. *Average Run Time* for all jobs.
10. *Number of Running Jobs*
11. *Number of Queuing Jobs*
12. *Number of Submitted Jobs*
13. *Number of Suspended Jobs*
14. *Number of Failed Jobs*: If there are Failed jobs, a list of the names of those jobs will be displayed.

In experiments that include `SIM` jobs, you will also see the average queuing and running time for these jobs. In the latest version the time format has been updated to `HH:mm:ss`. The text for the `SIM` average follows the format `avg. queue HH:mm:ss (M) | run HH:mm:ss (N)` where `M` is the number of jobs considered for the `avg. queue` calculation and `N` is the number of jobs considered for `run` calculation.

After clicking on the **MORE** button, you will be presented with the **Experiment Page**, which is the main view that Autosubmit provides. These are its main components:



12.5: Average Times Feature

12.1.1 Experiment Information

This component offers the main information about your experiment.

At the top left you see the `Autosubmit Searcher` home link that will take you back to the *Autosubmit GUI Main Page*, next to it you see the `Home` link that serves the same purpose, then you see the `About` link that takes you to a page with important information about the application (including the link to this documentation). Then you see the experiment name and `status`, which is updated every 5 minutes. Next, you see the *run history* button, this button opens a panel that shows information about previous runs of the experiment, only works for experiments running the latest version of Autosubmit. Then, you see the `esarchive status` badge, it shows information about the current status of the *esarchive* file system.

At the bottom you see some relevant metadata, including the `branch` of the `model` that was used in the experiment, the `HPC name` targeted by the experiment, the owner, the `version` that this experiment us running on, the `DB` version of Autosubmit, and the number of jobs in the experiment.

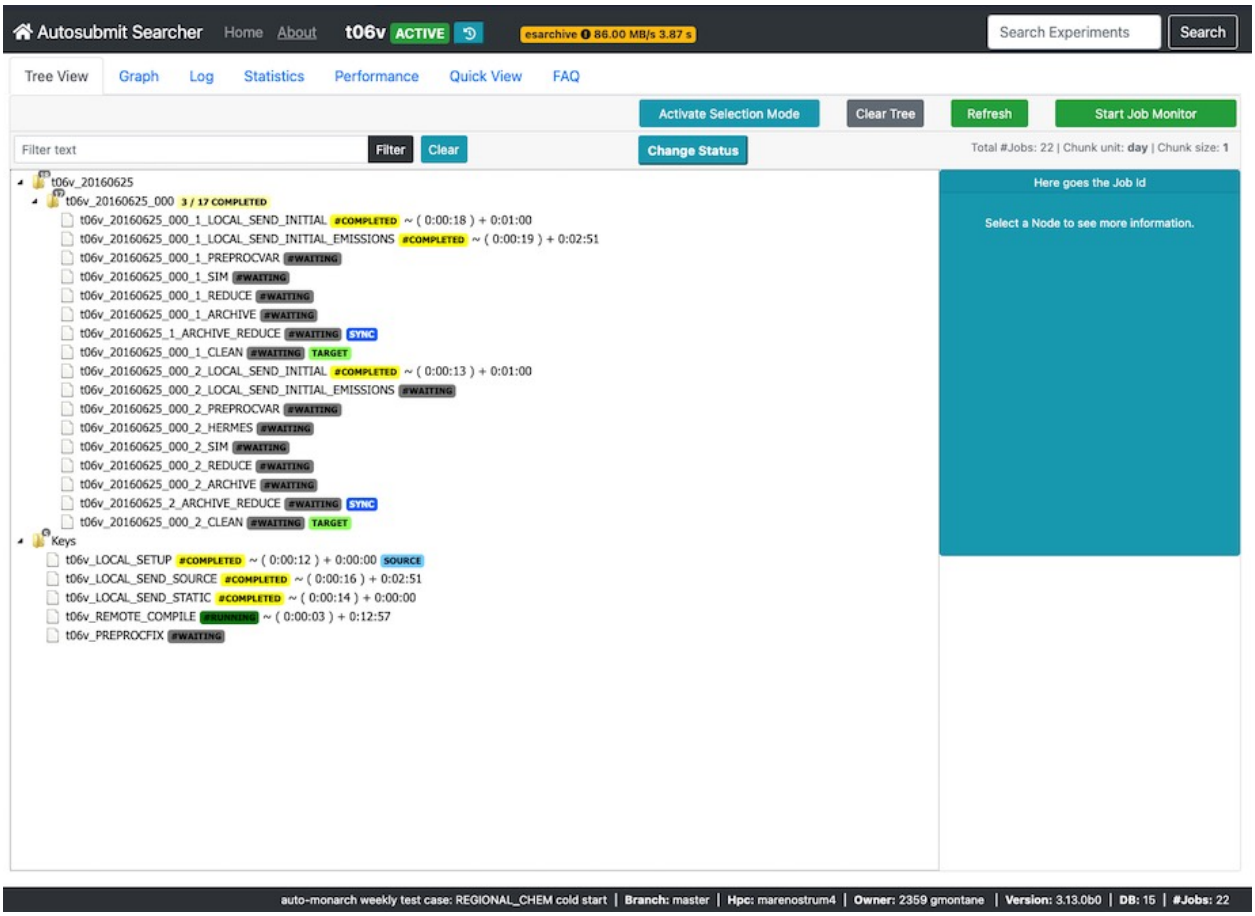
On the center you see the *Tree Representation*, which is loaded automatically when you open this page.

12.1.2 Tree Representation

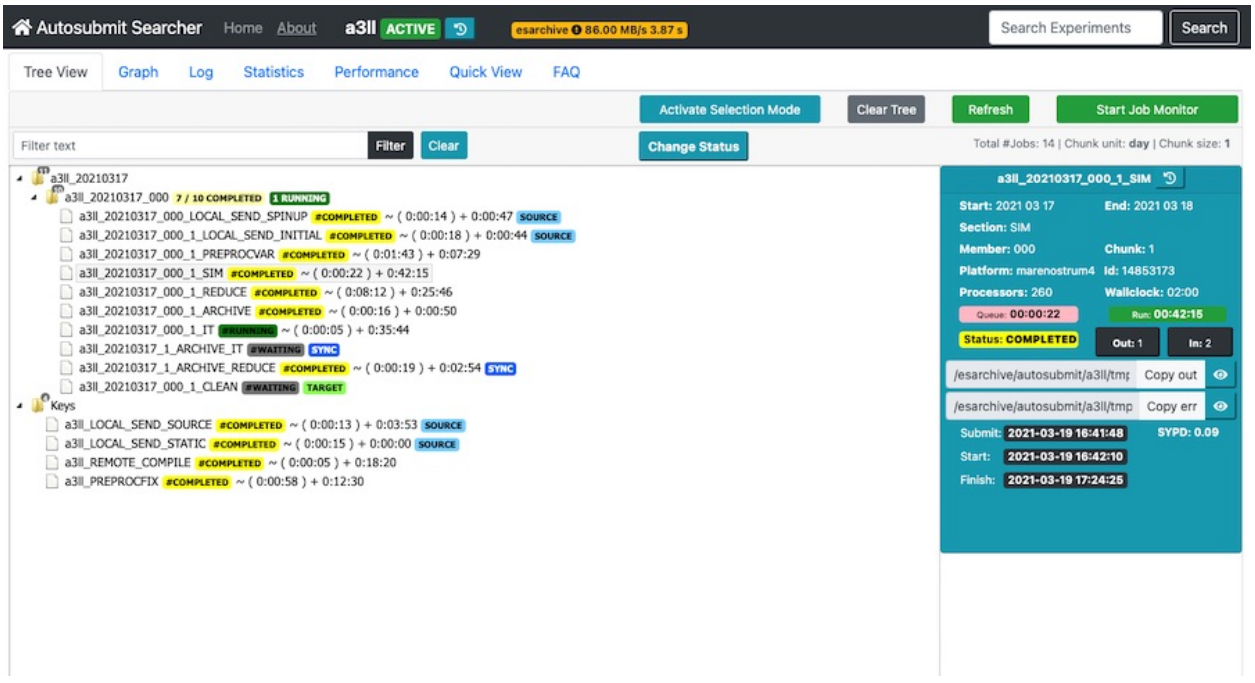
The Tree Representation offers a structured view of the experiment.

The view is organized in groups by `date`, and `date-member`. Each group has a folder icon, and next to the icon you can find the progress of the group as `completed / total` jobs (when all the jobs in a group have been completed, a check symbol will appear); then, an indicator of how many jobs inside that group are **RUNNING**, **QUEUING**, or have **FAILED**. Furthermore, if wrappers exist in the experiment, independent groups will be added for each wrapper that will contain the list of jobs included in the corresponding wrapper. This implies that a job can be repeated: once inside its `date-member` group and once in its wrapper group.

Inside each group you will find the list of jobs that belong to that group. The jobs are shown following this format: *job name* + # *job status* + (+ *queuing time* +) + *running time*. Jobs that belong to a wrapper have also a badge with the code of the wrapper.



12.6: Experiment Information



12.7: Experiment Tree Representation

When you click on a Job, you can see on the right panel (**Selection Panel**) the following information:

- *Start*: Starting date.
- *End*: Ending date.
- *Section*: Also known as job type.
- *Member*
- *Chunk*
- *Platform*: Remote platform.
- *Id*: Id in the remote platform.
- *Processors*: Number of processors required by the job.
- *Wallclock*: Time requested by the job.
- *Queue*: Time spent in queue, in minutes.
- *Run*: Time spent running, in minutes.
- *Status*: Job status.
- *Out*: Button that opens a list of jobs that depend on the one selected.
- *In*: Button that opens a list of jobs on which the selected job depends.
- *out path*: Path to the .out log file.
- *err path*: Path to the .err log file.
- *Submit*: Submit time of the job (If applicable).
- *Start*: Start time of the job (If applicable).
- *Finish*: Finish time of the job (If applicable).

Important: Next to the **out** and **err** paths, you see the a `Copy out/err` button that copies the path to your clipboard. Then you see an `eye symbol` button, that when clicked will show that last 150 lines of the **out/err** file.

Selection

When you click on a job in the tree view, a `Change Status` button will appear in the top bar, if you click, you will be presented with the option to generate a change status command that can be run on autosubmit, or to generate a format that can be used to change the status of the job while the experiment is running.

You can select many jobs at the same time by maintaining `CTRL` pressed and clicking on the jobs, then the generated command will include all these jobs.

Monitoring

If the experiment status is **RUNNING**, you will see a button called **Refresh** at the top right corner. This button will update the information of the jobs in the tree if necessary. Next to this button, you will see the button **Start Job Monitor**. When you click on it, a live **Job Monitor** will be initialized and the status of the jobs and wrappers will be queried every minute, any change will be updated in the **Tree View**. Also, if the **Job Monitor** is running, the detected changes will be listed in a panel **Monitor Panel** below the **Selection Panel**. You can stop this process by clicking on the button **Stop Job Monitor**.

The button **Clear Tree View** will clear the Tree Representation. It is also a valid way to refresh the Tree View.

Filter

At the top left you can find the **Filter text** input box. Insert any string and the list will show only those jobs whose description coincides with that string. For example `#COMPLETED` will show only completed jobs, `Wrapped` will show only those jobs that belong to a wrapper, `_fc0_` will show only those jobs that belong to the fc0 member. Press **Clear** to reset the filter. On the right side of this bar, you will see the total number of jobs, and the chunk unit used in the experiment.

Advanced Filter

It is possible to use the keychar `*` to separate keywords in the name of the job, in order. For example:

- `1850*fc0*_1_`: List all the jobs that have the string `1850` and then at least 1 occurrence of the string `fc0` and then at least 1 occurrence of the string `_1_`. This will effectively list all the jobs for the DATE that starts with `1850` for the member `fc0` and the chunk `_1_`.
- `000*_5`: List all the jobs that have the string `000` followed by at least one occurrence of the string `_5`. This will effectively list all the jobs that have member `000` and chunk number that starts with the digit `5`.
- `000*_5*PREPROCVAR`: It will also add the filter for jobs of type `PREPROCVAR`.

As you might infer, the logic is fairly straightforward: Start your string with the word or part of the word you are looking for, then add `*` and the word or part of the word that follows, and so on. The algorithm will split your string by `*` and then search for each part in order, once it finds the part in the title of the job, it takes a substring of the job title to not repeat the next search in the same string, it continues looking for the next part in the new reduced string, and so on.

You can extend this funcionality considering that date, member, section, chunk names start with the symbol _ and finish with the same symbol.

Important: This view is designed to show a structured view of your experiment, if you want a more dependency oriented view that shows better the execution sequence of your jobs, you can refer to *Graph Representation*.

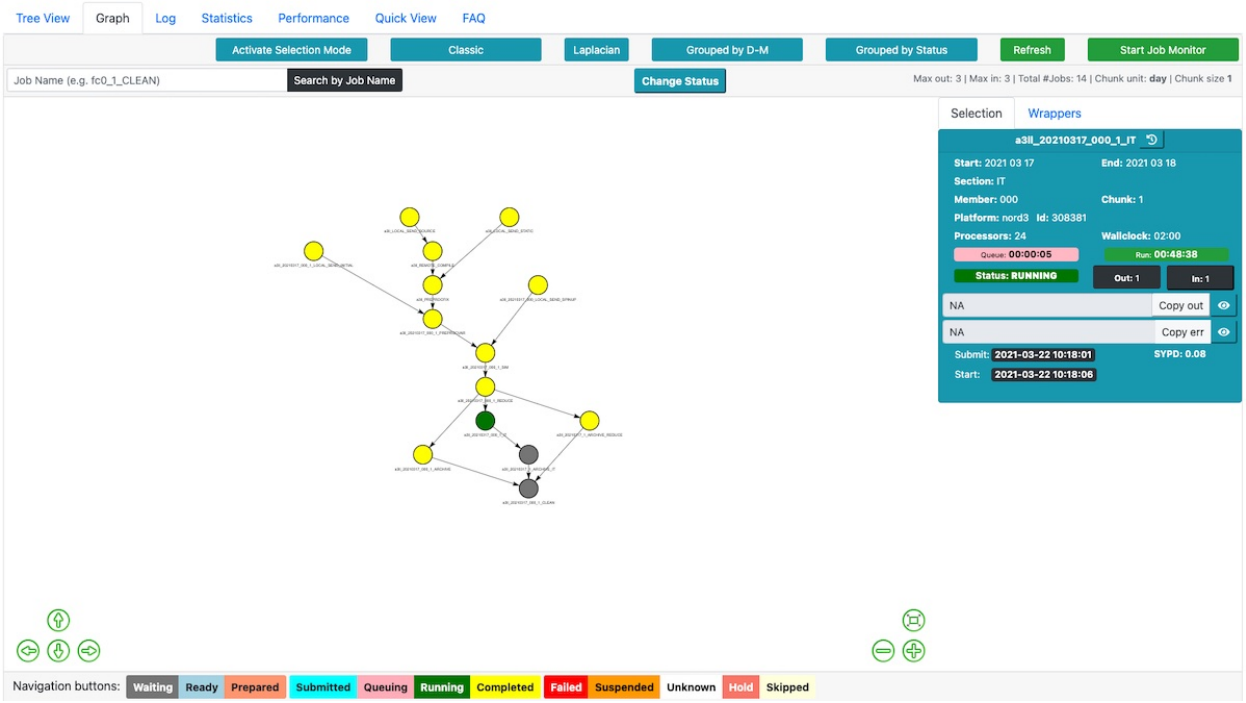
12.1.3 Graph Representation

The Graph Representation of the experiment offers a dependency oriented view.

This view offers a graph representation of the experiments where a node represents a job and an edge represents a directed dependency relationship between nodes. To open it you must click on the button `Classic`, which is the basic representation that uses either `GraphViz` or an heuristic approach depending on experiment complexity; we explain the other options later.

Once the graph representation is loaded, it will focus on a relevant node according to some established rules. The color of each node represents the status of the job it represents: you can see a color guide at the bottom of the page in the form of buttons. If you click in any of those buttons, the graph will focus on the last node with that status, except in the case of `WAITING` where the graph will focus on the first one. You can navigate the graph in this way, but there are other navigation buttons at the left and right corners of the graph canvas. You can also use your mouse or trackpad to navigate the graph, zoom in or zoom out. Below each node you can see the `job name` of the job it represents.

Important: For some experiments you will get a well distributed and generally good looking graph representation, for others you get a more straightforward representation. It depends on the size and dependency complexity of your experiments, not all experiments can be modeled as a good looking graph in reasonable time.



12.8: Experiment Graph Representation

When you click on a node, you can see on the right panel (**Selection Panel**) the following information:

- *Start*: Starting date.
- *End*: Ending date.
- *Section*: Also known as job type.
- *Member*
- *Chunk*
- *Platform*: Remote platform.
- *Id*: Id in the remote platform.
- *Processors*: Number of processors required by the job.
- *Wallclock*: Time requested by the job.
- *Queue*: Time spent in queue, in minutes.
- *Run*: Time spent running, in minutes.
- *Status*: Job status.
- *Out*: Button that opens a list of jobs that depend on the one selected.
- *In*: Button that opens a list of jobs on which the selected job depends.
- *out path*: Path to the .out log file.
- *err path*: Path to the .err log file.
- *Submit*: Submit time of the job (If applicable).
- *Start*: Start time of the job (If applicable).

- *Finish*: Finish time of the job (If applicable).

Important: Next to the **out** and **err** paths, you see the a `Copy out/err` button that copies the path to your clipboard. Then you see an `eye symbol` button, that when clicked will show that last 150 lines of the **out/err** file.

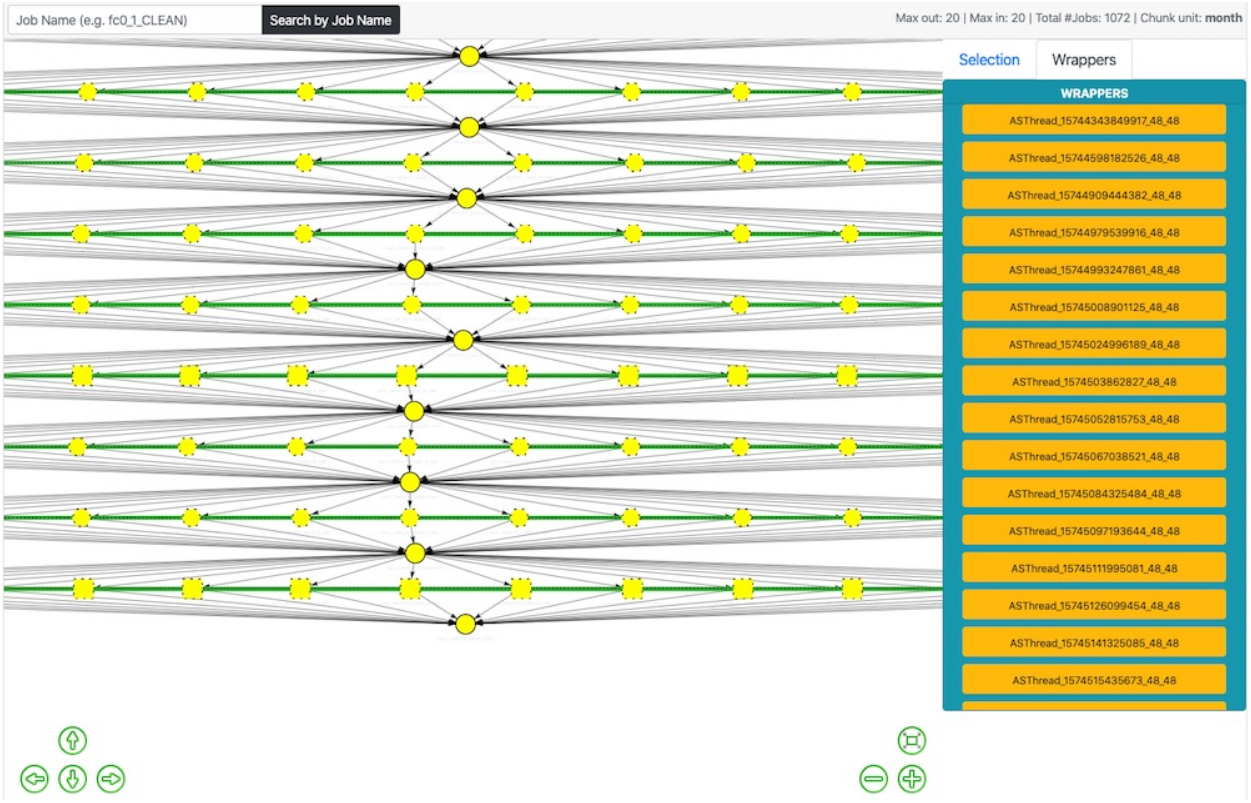
Selection

When you click on a node in the tree view, a `Change Status` button will appear in the top bar, if you click, you will be presented with the option to generate a change status command that can be run on autosubmit, or to generate a format that can be used to change the status of the job while the experiment is running.

You can select many nodes at the same time by maintaining `CTRL` pressed and clicking on the nodes, then the generated command will include all these jobs.

Wrappers Representation

Wrappers are an important feature of Autosubmit, and as such, it should be possible to visualize them in the graph representation.



12.9: Wrapper Graph Representation

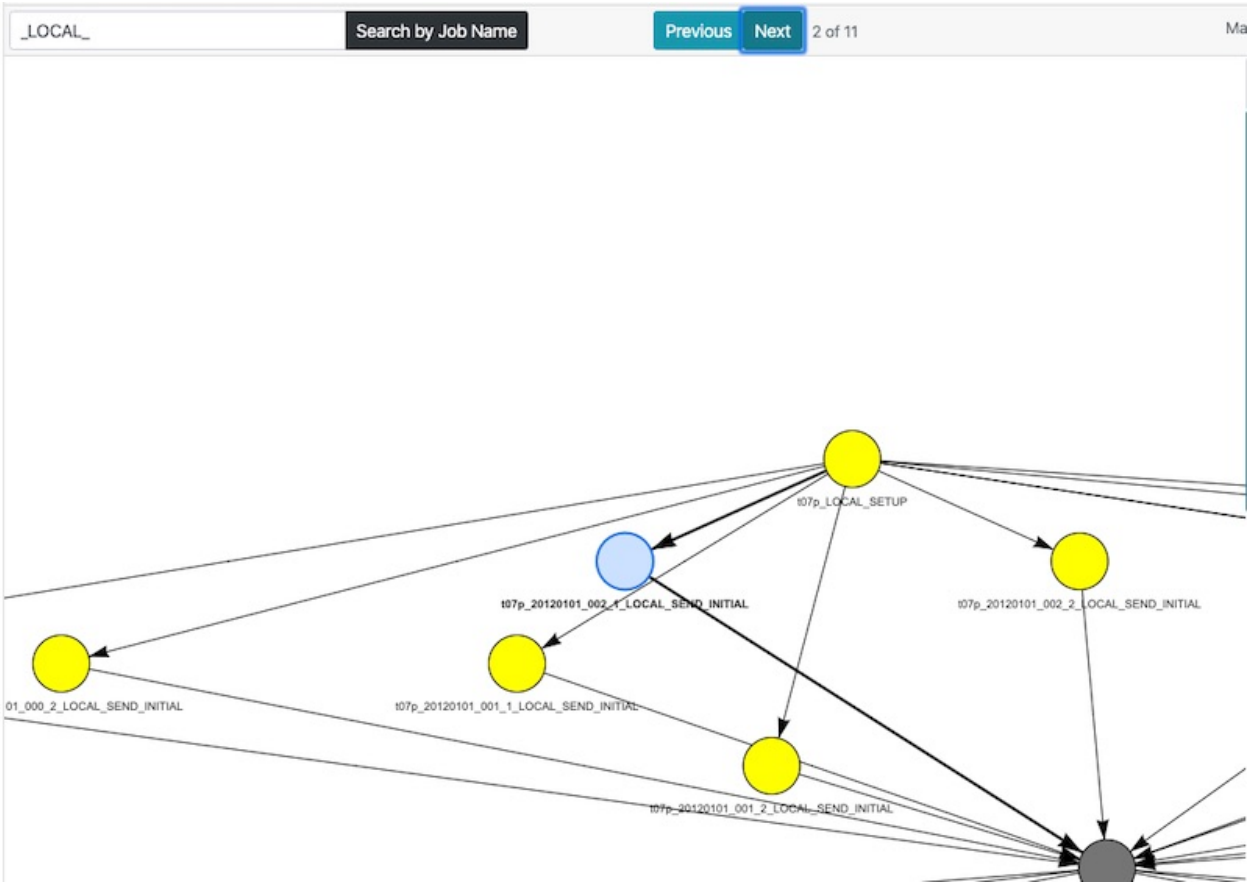
Wrappers are represented by nodes that have dashed border, hexagon or square shape (no difference between them), and that share green background edges. On the right side of the graph you can find the **Wrappers Tab** and it will display a list of the existing wrappers as buttons. If you click on any of these buttons, the nodes that belong to that wrapper will be highlighted.

Monitoring

If the experiment is `RUNNING` you will see at the top right corner the button `Start Job Monitor`. When you click on it, a live **Job Monitor** will be initialized and the status of the jobs and wrappers will be queried every minute, any change will be updated in the graph. Also, if the **Job Monitor** is running, the detected changes will be listed in a panel **Monitor Panel** below the **Selection Panel**. You can stop this process by clicking on the button **Stop Job Monitor**.

Important: While this is a good option to monitor the progress of your experiment, you can also use the *Autosubmit Log*.

Job Search



12.10: Job Search in Graph

On top of the graph you will see an input text box following by the button `Seach by Job Name`. Insert into that box the string that you want to find and the engine will build an internal list of those jobs whose name coincides with that string. For example `_LOCAL_` will show only jobs whose title contain the that string. Buttons `Previous` and `Next` will appear and next to them the number of jobs that coincide with your search, you can use these buttons to traverse the graph highlighting the nodes included in the resulting internal list.

Grouped by Date Member

By clicking on the button `Grouped by D-M` you get a graph representation where the nodes are clustered by date and member. For example, if your experiment has only one starting date and one member, then you will have only one cluster in this view. These clusters are represented by rectangular boxes whose color gives a general idea of the status of the jobs inside it.

Important: You can double click on any cluster to "open" it, meaning that the nodes that belong to that cluster will be freed and positioned individually.

Grouped by Status

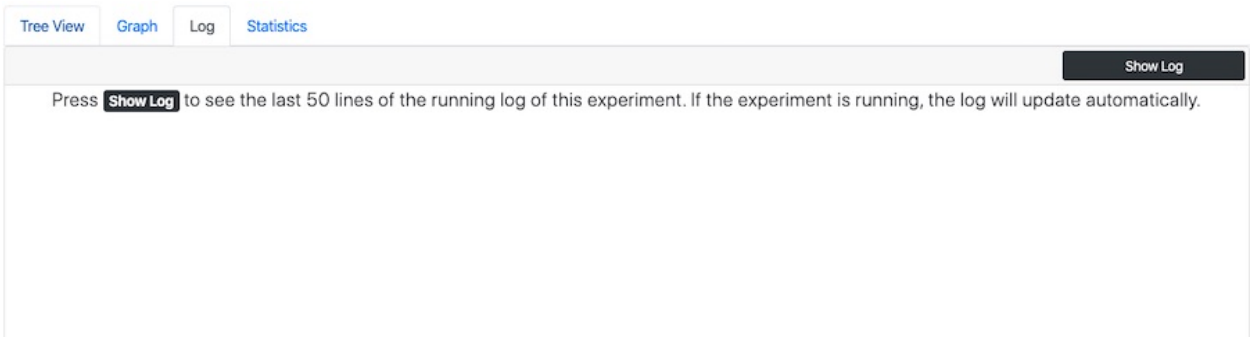
By clicking on the button `Grouped by Status` you get a graph representation where the nodes are clustered by status into 3 clusters: `WAITING`, `COMPLETED`, and `SUSPENDED`. Same rules mentioned for **Grouped by Date Member** apply.

Laplacian

By clicking on the button `Laplacian` you get a graph representation where the `(x,y)` coordinates of each node are calculated based on the second and third smallest eigenvector of the Graph Laplacian. All functionality is supported.

12.1.4 Autosubmit Log

When you click on the `Log` tab, you will see the button `Show Log`:



12.11: Experiment Log

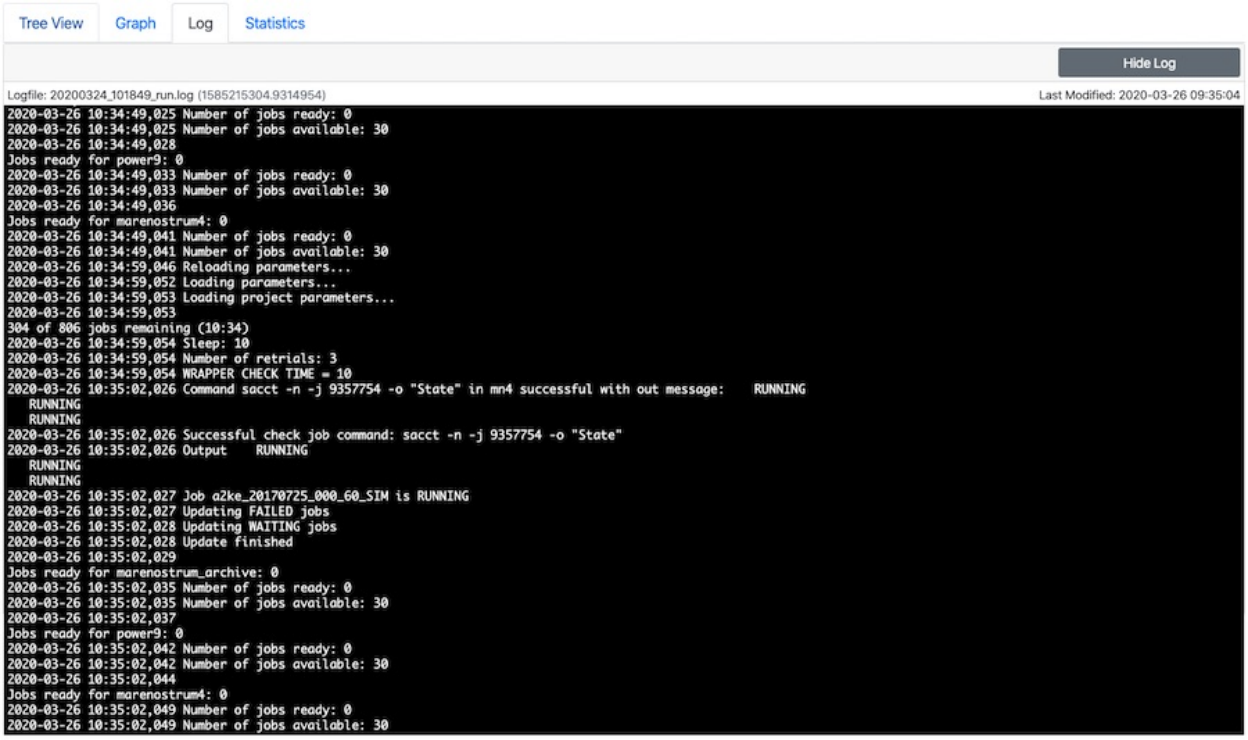
Important: The main Autosubmit log is usually stored in the folder `/tmp/` of your experiment, and this is the first path the system will scan.

When you click on the `Show Log` button, the last 150 lines of the log will be displayed:

At the top of the log you will see the name of the log file that is being displayed along with the timestamp of the last time the log was requested, and to the right you see this timestamp in `datetime` format.

If the experiment is currently running, the log will be updated periodically to keep you up with recent updates in the experiment execution. It is possible to scroll this view.

If you click on `Hide Log` the log will be cleared and the periodic updates will stop.



12.12: Experiment Log Open

12.1.5 Performance Metrics

The Performance Metrics tabs offers a set of metrics that try to measure the efficiency of the simulation performed, and other aspects of it.

On the left you have the values of the main performance metrics that apply to the experiment. Then, on the right, you see the list of jobs considered for this calculation with their data, also, `SYPD` and `ASYPD` are calculated individually for these jobs. This list is scrollable.

You can also access a `Show warnings` button that opens a list of important information that might affect the calculation of the metrics. You can click again on this button to close the list.

Further information about the metrics is included in the tab.

12.1.6 Autosubmit Statistics

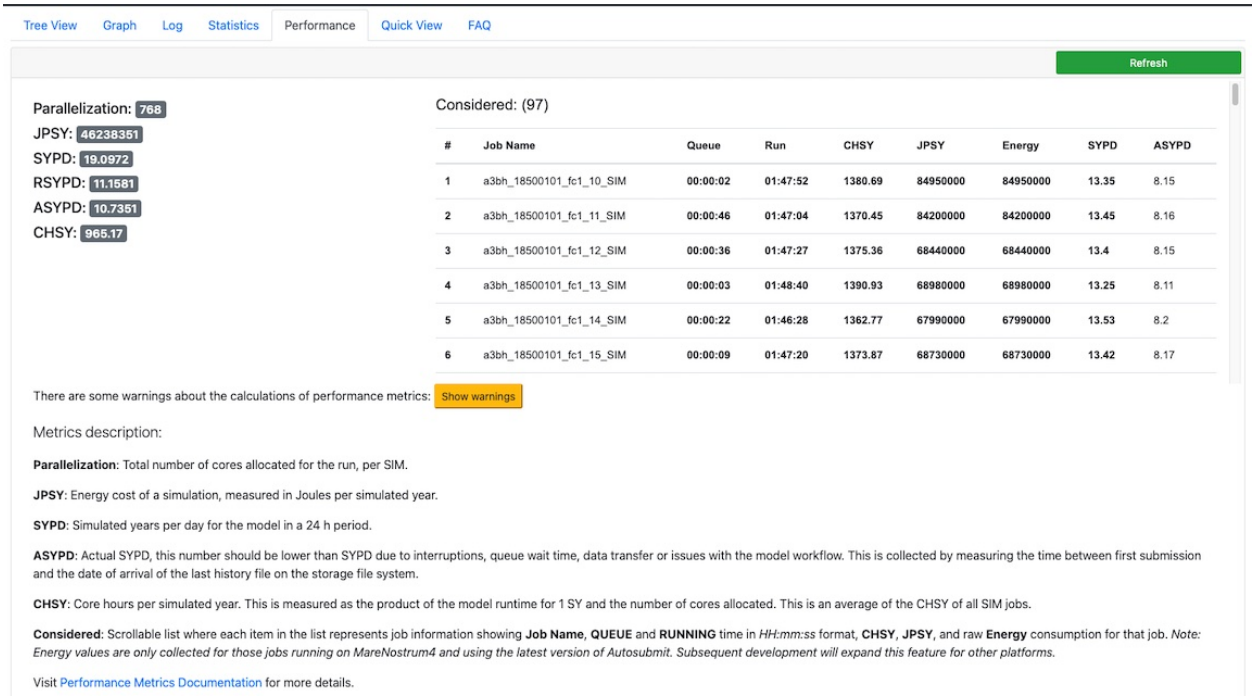
When you click on the `Statistics` tab, you will see two input boxes: `Section` and `Hours`, followed by the button `Get Statistics`:

There is also a brief explanation of the input fields and expected result. Basically, `Section` allows you to narrow your search to certain job types, and `Hours` allows you to set a time limit to look into the past in hours.

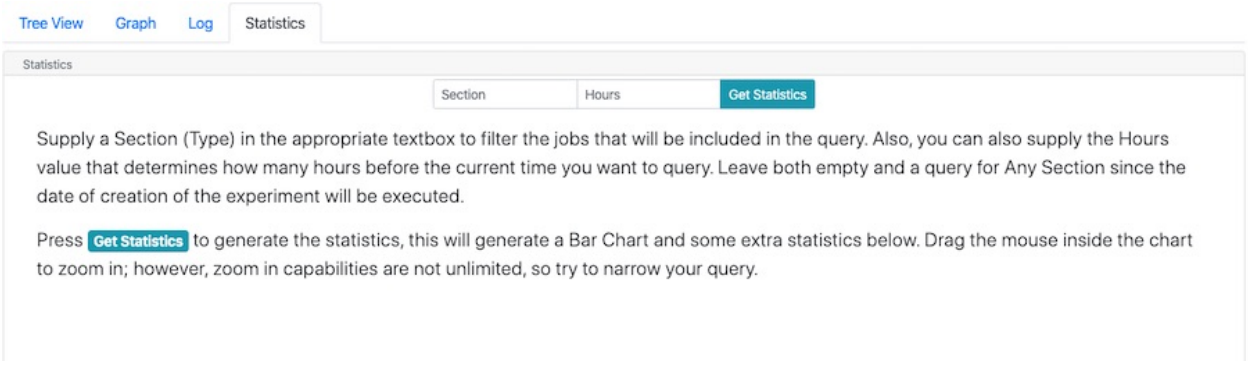
In this example we have queried 3 hours into the past:

Click on the button `Clear Statistics` to clear the results and submit another query.

Important: For more details about Autosubmit statistics, refer to: autoStatistics.



12.13: Performance Metrics Tab



12.14: Experiment Statistics

12.15: Experiment Statistics (Last 3 hours)

Important: To improve response times, Autosubmit GUI will try to store the dependency structure of your experiment in a database filed called `structure_expid.db` where `expid` is the name of your experiment. This file will be located in `/esarchive/autosubmit/expid/pkl/`.

Python Module Index

a

autosubmit.autosubmit, 47
autosubmit.config.basicConfig, 53
autosubmit.config.config_common, 54
autosubmit.database.db_common, 63
autosubmit.git.autosubmit_git, 64
autosubmit.job.job, 65
autosubmit.job.job_common, 69
autosubmit.job.job_list, 70
autosubmit.monitor.monitor, 76
autosubmit.platforms.ecplatform, 77
autosubmit.platforms.locplatform, 84
autosubmit.platforms.lsfplatform, 80
autosubmit.platforms.pbsplatform, 81
autosubmit.platforms.sgeplatform, 82
autosubmit.platforms.slurmplatform, 83

Index

A

add_logs() (*autosubmit.job.job_list.JobList method*), 70
add_parent() (*autosubmit.job.job.Job method*), 65
archive() (*autosubmit.autosubmit.Autosubmit static method*), 47
Autosubmit (*class in autosubmit.autosubmit*), 47
autosubmit.autosubmit (*module*), 47
autosubmit.config.basicConfig (*module*), 53
autosubmit.config.config_common (*module*), 54
autosubmit.database.db_common (*module*), 63
autosubmit.git.autosubmit_git (*module*), 64
autosubmit.job.job (*module*), 65
autosubmit.job.job_common (*module*), 69
autosubmit.job.job_list (*module*), 70
autosubmit.monitor.monitor (*module*), 76
autosubmit.platforms.ecplatform (*module*), 77
autosubmit.platforms.locplatform (*module*), 84
autosubmit.platforms.lsfplatform (*module*), 80
autosubmit.platforms.pbsplatform (*module*), 81
autosubmit.platforms.sgeplatform (*module*), 82
autosubmit.platforms.slurmplatform (*module*), 83
AutosubmitConfig (*class in autosubmit.config.config_common*), 54
AutosubmitGit (*class in autosubmit.git.autosubmit_git*), 64

B

backup_load() (*autosubmit.job.job_list.JobList method*), 70
backup_save() (*autosubmit.job.job_list.JobList method*), 70
BasicConfig (*class in autosubmit.config.basicConfig*), 53

C

change_status() (*autosubmit.autosubmit.Autosubmit static method*), 47
check() (*autosubmit.autosubmit.Autosubmit static method*), 47
check_autosubmit_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_commit() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 65
check_completion() (*autosubmit.job.job.Job method*), 65
check_conf_files() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_db() (*in module autosubmit.database.db_common*), 63
check_end_time() (*autosubmit.job.job.Job method*), 65
check_expdef_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_experiment_exists() (*in module autosubmit.database.db_common*), 63
check_file_exists() (*autosubmit.platforms.locplatform.LocalPlatform method*), 84
check_jobs_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_platforms_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_proj() (*autosubmit.config.config_common.AutosubmitConfig*

method), 54
check_proj_file() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
check_retrials_end_time() (*autosubmit.job.job.Job method*), 65
check_retrials_start_time() (*autosubmit.job.job.Job method*), 66
check_retrials_submit_time() (*autosubmit.job.job.Job method*), 66
check_running_after() (*autosubmit.job.job.Job method*), 66
check_script() (*autosubmit.job.job.Job method*), 66
check_scripts() (*autosubmit.job.job_list.JobList method*), 70
check_start_time() (*autosubmit.job.job.Job method*), 66
check_started_after() (*autosubmit.job.job.Job method*), 66
children (*autosubmit.job.job.Job attribute*), 66
clean() (*autosubmit.autosubmit.Autosubmit static method*), 47
clean_git() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 65
clean_plot() (*autosubmit.monitor.monitor.Monitor static method*), 76
clean_stats() (*autosubmit.monitor.monitor.Monitor static method*), 76
clone_repository() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 65
close_conn() (*in module autosubmit.database.db_common*), 63
color_status() (*autosubmit.monitor.monitor.Monitor static method*), 76
compare_by_id() (*autosubmit.job.job.Job method*), 66
compare_by_name() (*autosubmit.job.job.Job method*), 66
compare_by_status() (*autosubmit.job.job.Job method*), 67
configure() (*autosubmit.autosubmit.Autosubmit static method*), 48
configure_dialog() (*autosubmit.autosubmit.Autosubmit static method*), 48
connect() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
connect() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
connect() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
create() (*autosubmit.autosubmit.Autosubmit static method*), 48
create_db() (*in module autosubmit.database.db_common*), 63
create_script() (*autosubmit.job.job.Job method*), 67
create_tree_list() (*autosubmit.monitor.monitor.Monitor method*), 76

D

database_fix() (*autosubmit.autosubmit.Autosubmit static method*), 48
DbException, 63
delete() (*autosubmit.autosubmit.Autosubmit static method*), 48
delete_child() (*autosubmit.job.job.Job method*), 67
delete_experiment() (*in module autosubmit.database.db_common*), 63
delete_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
delete_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
delete_parent() (*autosubmit.job.job.Job method*), 67
describe() (*autosubmit.autosubmit.Autosubmit static method*), 49

E

EcPlatform (*class in autosubmit.platforms.ecplatform*), 77
experiment_file (*autosubmit.config.config_common.AutosubmitConfig attribute*), 54
expid (*autosubmit.job.job_list.JobList attribute*), 70
expid() (*autosubmit.autosubmit.Autosubmit static method*), 49

G

generate() (*autosubmit.job.job_list.JobList method*), 70
generate_output() (*autosubmit.monitor.monitor.Monitor method*), 77
generate_output_stats() (*autosubmit.monitor.monitor.Monitor method*), 77
generate_output_txt() (*autosubmit.monitor.monitor.Monitor method*), 77
generate_scripts_andor_wrappers() (*autosubmit.autosubmit.Autosubmit static method*), 49

get_active() (*autosubmit.job.job_list.JobList method*), 71
get_all() (*autosubmit.job.job_list.JobList method*), 71
get_autosubmit_version() (*in module autosubmit.database.db_common*), 63
get_checkAlljobs_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 83
get_checkjob_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
get_checkjob_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
get_checkjob_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
get_checkjob_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
get_checkjob_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
get_checkjob_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 83
get_chunk_ini() (*autosubmit.config.config_common.AutosubmitConfig method*), 54
get_chunk_list() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_chunk_list() (*autosubmit.job.job_list.JobList method*), 71
get_chunk_size() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_chunk_size_unit() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_communications_library() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_completed() (*autosubmit.job.job_list.JobList method*), 71
get_copy_remote_logs() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_current_host() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_current_project() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_current_user() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_custom_directives() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_date_list() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_date_list() (*autosubmit.job.job_list.JobList method*), 71
get_default_job_type() (*autosubmit.config.config_common.AutosubmitConfig method*), 55
get_disable_recovery_threads() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_export() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_failed() (*autosubmit.job.job_list.JobList method*), 71
get_fetch_single_branch() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
get_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
get_file_jobs_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_file_project_conf() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_finished() (*autosubmit.job.job_list.JobList method*), 72
get_full_config_as_dict() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_full_config_as_json() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_general_stats() (*autosubmit.monitor.monitor.Monitor static method*), 77
get_git_project_branch() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_git_project_commit() (*autosubmit.config.config_common.AutosubmitConfig*

method), 56
get_git_project_origin() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_git_remote_project_root() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_held_jobs() (*autosubmit.job.job_list.JobList method*), 72
get_in_queue() (*autosubmit.job.job_list.JobList method*), 72
get_job_by_name() (*autosubmit.job.job_list.JobList method*), 72
get_job_list() (*autosubmit.job.job_list.JobList method*), 72
get_job_names() (*autosubmit.job.job_list.JobList method*), 72
get_job_related() (*autosubmit.job.job_list.JobList method*), 72
get_jobs_sections() (*autosubmit.config.config_common.AutosubmitConfig method*), 56
get_last_retrials() (*autosubmit.job.job.Job method*), 67
get_local_project_path() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_logs() (*autosubmit.job.job_list.JobList method*), 73
get_logs_files() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
get_mails_to() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_max_processors() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_max_waiting_jobs() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_max_wallclock() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_max_wrapped_jobs() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_member_list() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_member_list() (*autosubmit.job.job_list.JobList method*), 73
get_memory() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_memory_per_task() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_migrate_duplicate() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_migrate_host_to() (*autosubmit.config.config_common.AutosubmitConfig method*), 57
get_migrate_project_to() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_migrate_user_to() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_min_wrapped_jobs() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_mkdir_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
get_mkdir_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
get_mkdir_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
get_mkdir_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
get_mkdir_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
get_mkdir_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 83
get_not_in_queue() (*autosubmit.job.job_list.JobList method*), 73
get_notifications() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_num_chunks() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_ordered_jobs_by_date_member() (*autosubmit.job.job_list.JobList method*), 73
get_output_type() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_parse_two_step_start() (*autosubmit.config.config_common.AutosubmitConfig method*), 58
get_parser() (*autosub-*

mit.config.config_common.AutosubmitConfig static method), 58
get_platform() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_prepared() (*autosubmit.job.job_list.JobList method*), 73
get_processors() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_project_destination() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_project_dir() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_project_type() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_queuing() (*autosubmit.job.job_list.JobList method*), 73
get_ready() (*autosubmit.job.job_list.JobList method*), 73
get_remote_dependencies() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_rerun() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_retrials() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_running() (*autosubmit.job.job_list.JobList method*), 73
get_safetysleeptime() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_scratch_free_space() (*autosubmit.config.config_common.AutosubmitConfig method*), 59
get_skipped() (*autosubmit.job.job_list.JobList method*), 73
get_ssh_output() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
get_ssh_output() (*autosubmit.platforms.locplatform.LocalPlatform method*), 85
get_storage_type() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_submit_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 78
get_submit_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
get_submit_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
get_submit_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
get_submit_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
get_submit_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 83
get_submitted() (*autosubmit.job.job_list.JobList method*), 74
get_submitted_job_id() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
get_submitted_job_id() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
get_submitted_job_id() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
get_submitted_job_id() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
get_submitted_job_id() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
get_submitted_job_id() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84
get_submodules_list() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_suspended() (*autosubmit.job.job_list.JobList method*), 74
get_svn_project_revision() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_svn_project_url() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_synchronize() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_tasks() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_threads() (*autosubmit.config.config_common.AutosubmitConfig*

method), 60
get_total_jobs() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_uncompleted() (*autosubmit.job.job_list.JobList method*), 74
get_unknown() (*autosubmit.job.job_list.JobList method*), 74
get_unsubmitted() (*autosubmit.job.job_list.JobList method*), 74
get_version() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_waiting() (*autosubmit.job.job_list.JobList method*), 74
get_waiting_remote_dependencies() (*autosubmit.job.job_list.JobList method*), 74
get_wallclock() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_wchunkinc() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_wrapper_check_time() (*autosubmit.config.config_common.AutosubmitConfig method*), 60
get_wrapper_export() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_jobs() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_machinefiles() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_method() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_policy() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_queue() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
get_wrapper_type() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
graph (*autosubmit.job.job_list.JobList attribute*), 75

H

has_children() (*autosubmit.job.job.Job method*), 67
has_parents() (*autosubmit.job.job.Job method*), 67

I

inc_fail_count() (*autosubmit.job.job.Job method*), 67
increase_wallclock_by_chunk() (*in module autosubmit.job.job_common*), 70
inspect() (*autosubmit.autosubmit.Autosubmit static method*), 49
install() (*autosubmit.autosubmit.Autosubmit static method*), 50
is_a_completed_retrial() (*autosubmit.job.job.Job static method*), 67
is_ancestor() (*autosubmit.job.job.Job method*), 67
is_parent() (*autosubmit.job.job.Job method*), 68

J

Job (*class in autosubmit.job.job*), 65
JobList (*class in autosubmit.job.job_list*), 70
jobs_file (*autosubmit.config.config_common.AutosubmitConfig attribute*), 61
jobs_in_queue() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79

L

last_name_used() (*in module autosubmit.database.db_common*), 64
load() (*autosubmit.job.job_list.JobList method*), 75
load_file() (*autosubmit.job.job_list.JobList static method*), 75
load_parameters() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
load_platform_parameters() (*autosubmit.config.config_common.AutosubmitConfig method*), 61
load_project_parameters() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
load_section_parameters() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
LocalPlatform (*class in autosubmit.platforms.locplatform*), 84
log_job() (*autosubmit.job.job.Job method*), 68
long_name (*autosubmit.job.job.Job attribute*), 68
LsfPlatform (*class in autosubmit.platforms.lsfplatform*), 80

M

migrate() (*autosubmit.autosubmit.Autosubmit static method*), 50
Monitor (*class in autosubmit.monitor.monitor*), 76

monitor() (*autosubmit.autosubmit.Autosubmit static method*), 50
move_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
move_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86

O

open_conn() (*in module autosubmit.database.db_common*), 64

P

parameters (*autosubmit.job.job_list.JobList attribute*), 75
parents (*autosubmit.job.job.Job attribute*), 68
parse_Alljobs_output() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84
parse_args() (*autosubmit.autosubmit.Autosubmit static method*), 50
parse_job_finish_data() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84
parse_job_output() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
parse_job_output() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
parse_job_output() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
parse_job_output() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
parse_job_output() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 82
parse_job_output() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84
parse_output_number() (*in module autosubmit.job.job_common*), 70
PBSPlatform (*class in autosubmit.platforms.pbsplatform*), 81
pkl_fix() (*autosubmit.autosubmit.Autosubmit static method*), 50
platform (*autosubmit.job.job.Job attribute*), 68
platforms_file (*autosubmit.config.config_common.AutosubmitConfig attribute*), 62
platforms_parser (*autosubmit.config.config_common.AutosubmitConfig attribute*), 62
print_job() (*autosubmit.job.job.Job method*), 68
print_parameters() (*autosubmit.job.job.Job method*), 68
print_with_status() (*autosubmit.job.job_list.JobList method*), 75
project_file (*autosubmit.config.config_common.AutosubmitConfig attribute*), 62

Q

queue (*autosubmit.job.job.Job attribute*), 68

R

read() (*autosubmit.config.basicConfig.BasicConfig static method*), 53
recovery() (*autosubmit.autosubmit.Autosubmit static method*), 50
refresh() (*autosubmit.autosubmit.Autosubmit static method*), 51
reload() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
remove_redundant_parents() (*autosubmit.job.job.Job method*), 68
remove_rerun_only_jobs() (*autosubmit.job.job_list.JobList method*), 75
report() (*autosubmit.autosubmit.Autosubmit static method*), 51
rerun() (*autosubmit.job.job_list.JobList method*), 75
rerun_recovery() (*autosubmit.autosubmit.Autosubmit static method*), 51
restore_connection() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
restore_connection() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 83
run_experiment() (*autosubmit.autosubmit.Autosubmit static method*), 51

S

save() (*autosubmit.job.job_list.JobList method*), 75
save_experiment() (*in module autosubmit.database.db_common*), 64
send_command() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
send_command() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86

send_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
send_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
set_expid() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_git_project_commit() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_new_host() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_new_project() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_new_user() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_platform() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_safetysleeptime() (*autosubmit.config.config_common.AutosubmitConfig method*), 62
set_status() (*autosubmit.autosubmit.Autosubmit static method*), 51
set_version() (*autosubmit.config.config_common.AutosubmitConfig method*), 63
SgePlatform (*class in autosubmit.platforms.sgeplatform*), 82
signal_handler() (*in module autosubmit.autosubmit*), 53
signal_handler_create() (*in module autosubmit.autosubmit*), 53
SlurmPlatform (*class in autosubmit.platforms.slurmplatform*), 83
sort_by_id() (*autosubmit.job.job_list.JobList method*), 75
sort_by_name() (*autosubmit.job.job_list.JobList method*), 75
sort_by_status() (*autosubmit.job.job_list.JobList method*), 76
sort_by_type() (*autosubmit.job.job_list.JobList method*), 76
statistics() (*autosubmit.autosubmit.Autosubmit static method*), 51
StatisticsSnippetBash (*class in autosubmit.job.job_common*), 69
StatisticsSnippetEmpty (*class in autosubmit.job.job_common*), 69
StatisticsSnippetPython (*class in autosubmit.job.job_common*), 69
StatisticsSnippetR (*class in autosubmit.job.job_common*), 69
Status (*class in autosubmit.job.job_common*), 69
submit_ready_jobs() (*autosubmit.autosubmit.Autosubmit static method*), 52
submit_Script() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84

T

test() (*autosubmit.autosubmit.Autosubmit static method*), 52
test_connection() (*autosubmit.platforms.ecplatform.EcPlatform method*), 79
test_connection() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
test_connection() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 83
testcase() (*autosubmit.autosubmit.Autosubmit static method*), 52
total_processors (*autosubmit.job.job.Job attribute*), 68
Type (*class in autosubmit.job.job_common*), 69

U

unarchive() (*autosubmit.autosubmit.Autosubmit static method*), 53
update_cmds() (*autosubmit.platforms.ecplatform.EcPlatform method*), 80
update_cmds() (*autosubmit.platforms.locplatform.LocalPlatform method*), 86
update_cmds() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 80
update_cmds() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 81
update_cmds() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 83
update_cmds() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 84
update_content() (*autosubmit.job.job.Job method*), 68

update_experiment_descrip_version() (*in module autosubmit.database.db_common*), 64
update_from_file() (*autosubmit.job.job_list.JobList method*), 76
update_genealogy() (*autosubmit.job.job_list.JobList method*), 76
update_list() (*autosubmit.job.job_list.JobList method*), 76
update_parameters() (*autosubmit.job.job.Job method*), 68
update_status() (*autosubmit.job.job.Job method*), 69
update_version() (*autosubmit.autosubmit.Autosubmit static method*), 53

W

WrapperJob (*class in autosubmit.job.job*), 69
write_end_time() (*autosubmit.job.job.Job method*), 69
write_start_time() (*autosubmit.job.job.Job method*), 69
write_submit_time() (*autosubmit.job.job.Job method*), 69