

---

# **autosubmit Documentation**

***Release 3.9.1***

**Domingo Manubens - Joan López**

**Jul 20, 2022**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What is Autosubmit ? . . . . .	1
1.2	Why is Autosubmit needed ? . . . . .	1
1.3	How does Autosubmit work ? . . . . .	2
<b>2</b>	<b>Tutorial</b>	<b>5</b>
2.1	Quick start guide . . . . .	5
<b>3</b>	<b>Installation</b>	<b>17</b>
3.1	How to install . . . . .	17
3.2	How to configure . . . . .	18
<b>4</b>	<b>Usage</b>	<b>21</b>
4.1	Command list . . . . .	21
4.2	Tutorials (How to) . . . . .	22
<b>5</b>	<b>Defining the workflow</b>	<b>23</b>
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
<b>6</b>	<b>Frequent Questions and Answers</b>	<b>33</b>
<b>7</b>	<b>Troubleshooting</b>	<b>35</b>
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	Error on create caused by a configuration parsing error . . . . .	35
7.6	Other possible errors . . . . .	36
<b>8</b>	<b>Error codes and solutions</b>	<b>37</b>
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
<b>9</b>	<b>Developing a project</b>	<b>41</b>
<b>10</b>	<b>Variables reference</b>	<b>43</b>
10.1	Job variables	43
10.2	Platform variables	44
10.3	Project variables	45
10.4	Performance Metrics	45
<b>11</b>	<b>Module documentation</b>	<b>47</b>
11.1	autosubmit	47
11.2	autosubmit.config	54
11.3	autosubmit.database	64
11.4	autosubmit.git	66
11.5	autosubmit.job	66
11.6	autosubmit.monitor	79
11.7	autosubmit.platform	80
<b>12</b>	<b>Autosubmit GUI</b>	<b>91</b>
12.1	Autosubmit GUI Main Page	91
	<b>Python Module Index</b>	<b>107</b>
	<b>Index</b>	<b>109</b>

### 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:

GitLab:	<a href="https://earth.bsc.es/gitlab/es/autosubmit">https://earth.bsc.es/gitlab/es/autosubmit</a>
Mail:	<a href="mailto:support-autosubmit@bsc.es">support-autosubmit@bsc.es</a>

### 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
```

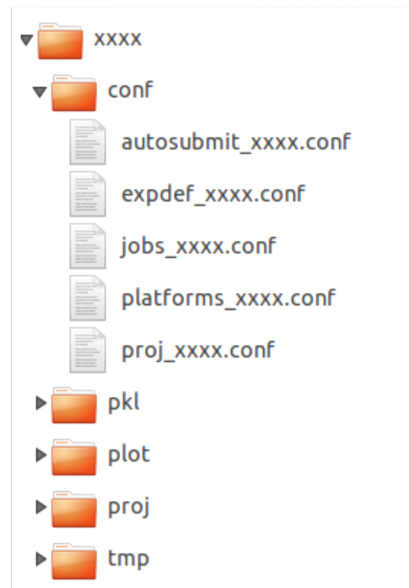
### 1.3.1 Experiment creation

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.

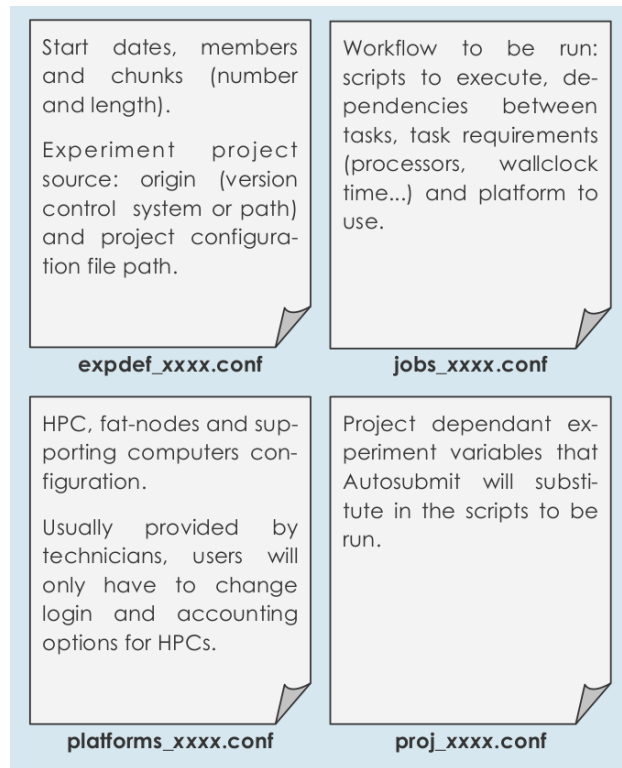
This command assigns a unique four character identifier (xxxx, names starting from a letter, the other three characters) to the experiment and creates a new folder in experiments repository with structure shown in Figure 1.1.



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:

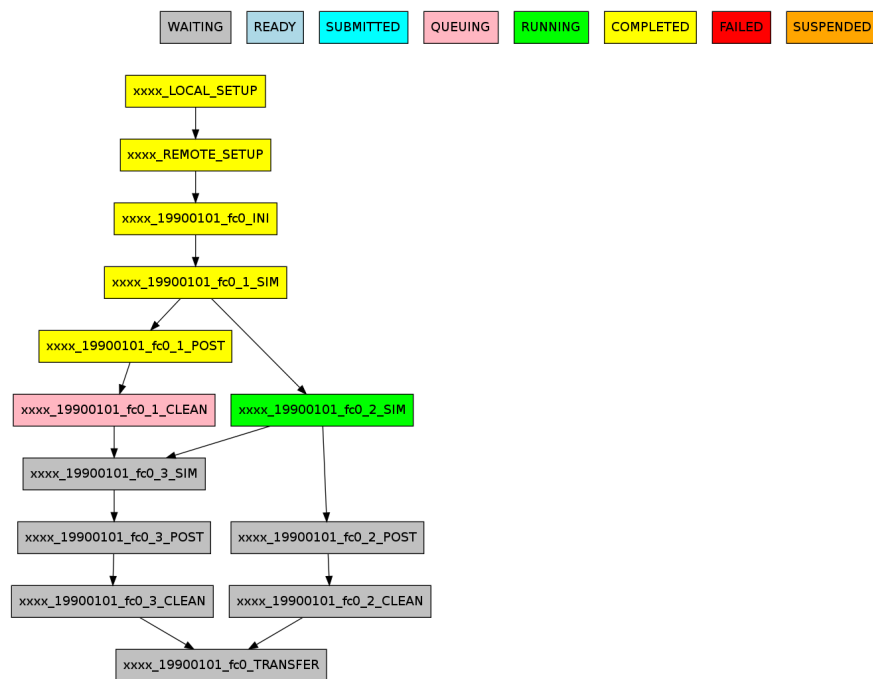
```
autosubmit run xxxx
```

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 from `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.



## 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 abbreviated syntax 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 jobs to rerun
RERUN_JOBLIST =

[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 omitted from workflow.
## Path relative to the project directory
# FILE =
## Platform to execute the job. If not specified, defaults to HPCARCH in expdef_
↳file.
## LOCAL is always defined and refers to current machine
# PLATFORM =
## Queue to add the job to. If not specified, uses PLATFORM default.
# QUEUE =
## Defines dependencies from job as a list of parents jobs separated 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 startdate, 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
## 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
## Enables hyperthreading. If not specified, defaults to false.
# HYPERTHREADING = false
```

(continues on next page)

(continued from previous page)

```

## Tasks number to be submitted to the HPC. If not specified, defaults to 1.
# Tasks = 1
## 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
## Allows to put a delay between retries, of retrials if a job fails. If not
↳specified, it will be static
# DELAY_RETRY_TIME = 11
# DELAY_RETRY_TIME = +11 # will wait 11,22,33,44...
# DELAY_RETRY_TIME = *11 # will wait 11,110,1110,11110...
## 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

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

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

```

(continues on next page)

(continued from previous page)

**[TRANSFER]**

```
FILE = TRANSFER.sh
PLATFORM = LOCAL
DEPENDENCIES = CLEAN
RUNNING = member
```

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

```
# Example platform with all options specified

## Platform name
# [PLATFORM]
## 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]**

(continues on next page)

(continued from previous page)

```
# Queue type. Options: ps, SGE, LSF, SLURM, PBS, eceaccess
TYPE = SGE
HOST = ithaca
PROJECT = cfu
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
## Allows to put a delay between retries, of retrials if a job fails. If not_
↳specified, it will be static
# DELAY_RETRY_TIME = 11
# DELAY_RETRY_TIME = +11 # will wait 11,22,33,44...
# DELAY_RETRY_TIME = *11 # will wait 11,110,1110,11110...
# 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 have 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 rlp1, 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)
```

(continues on next page)



(continued from previous page)

```

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
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
NFIYR = 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 =

```

(continues on next page)

(continued from previous page)

```

# 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]
# 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 =

```

(continues on next page)

(continued from previous page)

```

# 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 =
# 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 =

```

(continues on next page)

(continued from previous page)

```
# 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
```

### 3.1 How to install

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

- Pre-requisties: bash, python2, sqlite3, git-scm > 1.8.2, subversion, dialog, curl, python-tk, python2-dev, graphviz >= 2.41, pip2

**Important:** (SYSTEM) Graphviz version must be >= 2.38 except 2.40(not working). You can check the version using `dot -v`.

- Python dependencies: argparse, python-dateutil, pyparsing, numpy, pydotplus, matplotlib, paramiko, python2-pythondialog, portalocker, requests, typing

**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 bscsautosubmit01 bscsautosubmit02 # Add localhost only if you
↳are not on esarchive system

```

Now you are ready to use Autosubmit !

Example:

```
.. code-block:: ini
```

```

# Update repositories apt update
# Avoid interactive stuff export DEBIAN_FRONTEND=noninteractive
# Dependencies apt install wget curl python2 python-tk python2-dev graphviz -y -q
# Additional dependencies related with pycrypto apt install build-essential libssl-dev libffi-dev -y -q
# Download get pip script and launch it wget https://bootstrap.pypa.io/pip/2.7/get-pip.py python2 get-
pip.py
# Install autosubmit using pip pip2 install autosubmit
# Check that we can execute autosubmit commands autosubmit -h
# Configure autosubmit configure
# Install autosubmit install
# Get expid autosubmit expid -H TEST -d "Test exp."
# Create with -np # Since it was a new install the expid will be a000 autosubmit create a000 -np

```





## 4.1 Command list

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

<b>-report</b>	extract experiment parameters
<b>-updateversion</b>	Updates the Autosubmit version of your experiment with the current version of the module you are using
<b>-dbfix</b>	Fixes the database malformed error in the historical database of your experiment
<b>-pklfix</b>	Fixed the blank.pkl error of your experiment
<b>-updatedescrip</b>	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

---

## 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 `DEPENDENCIES` 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.

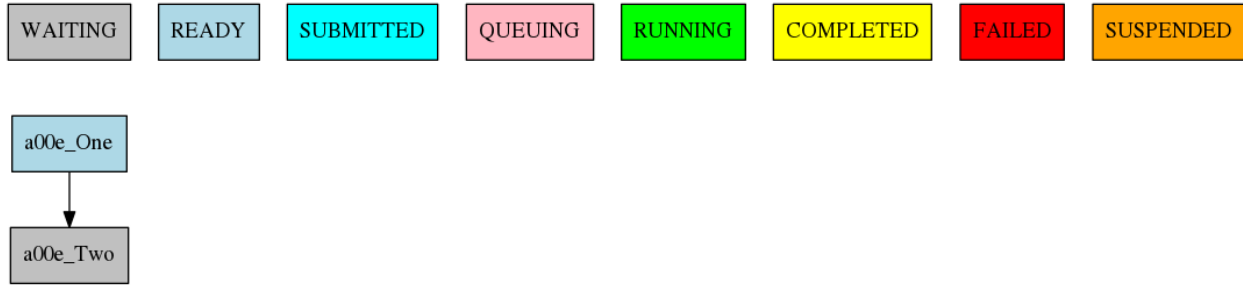
```
[One]
FILE = one.sh

[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.

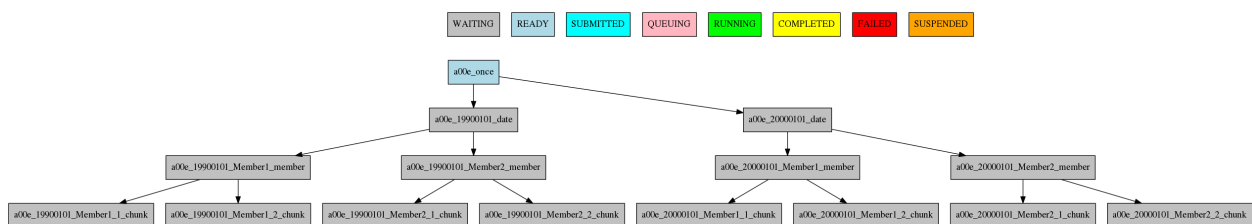
```
[once]
FILE = Once.sh

[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

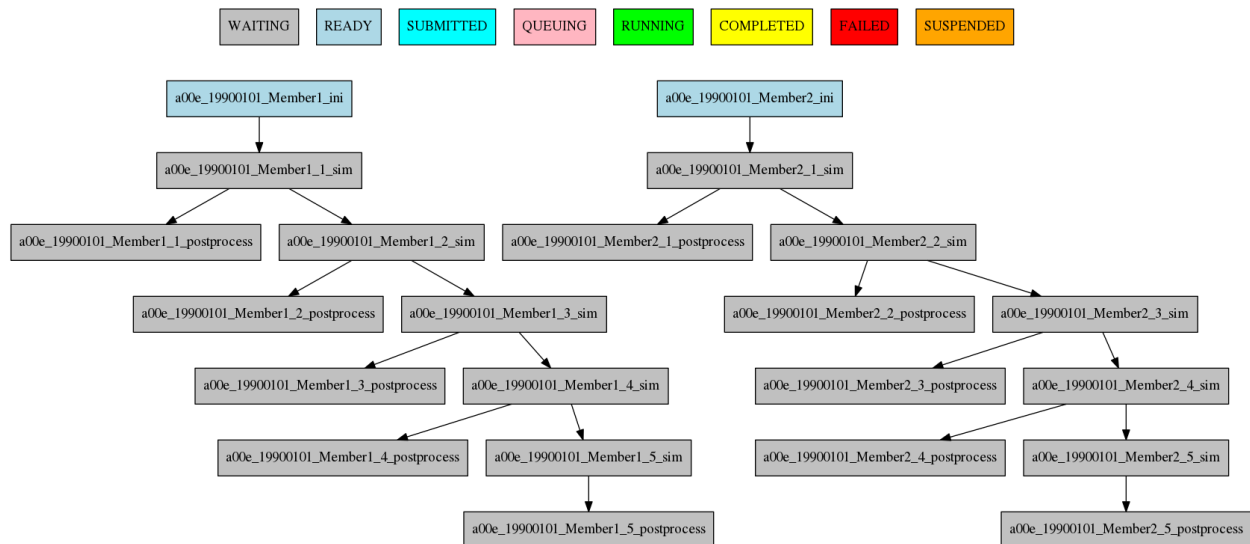
```
[ini]
FILE = ini.sh
RUNNING = member

[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.

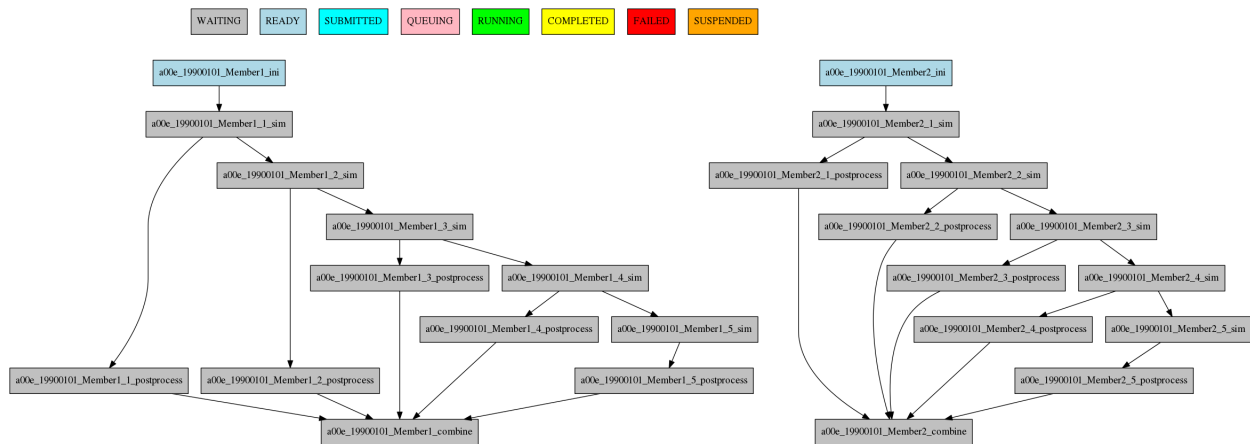
```
[ini]
FILE = ini.sh
RUNNING = member

[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

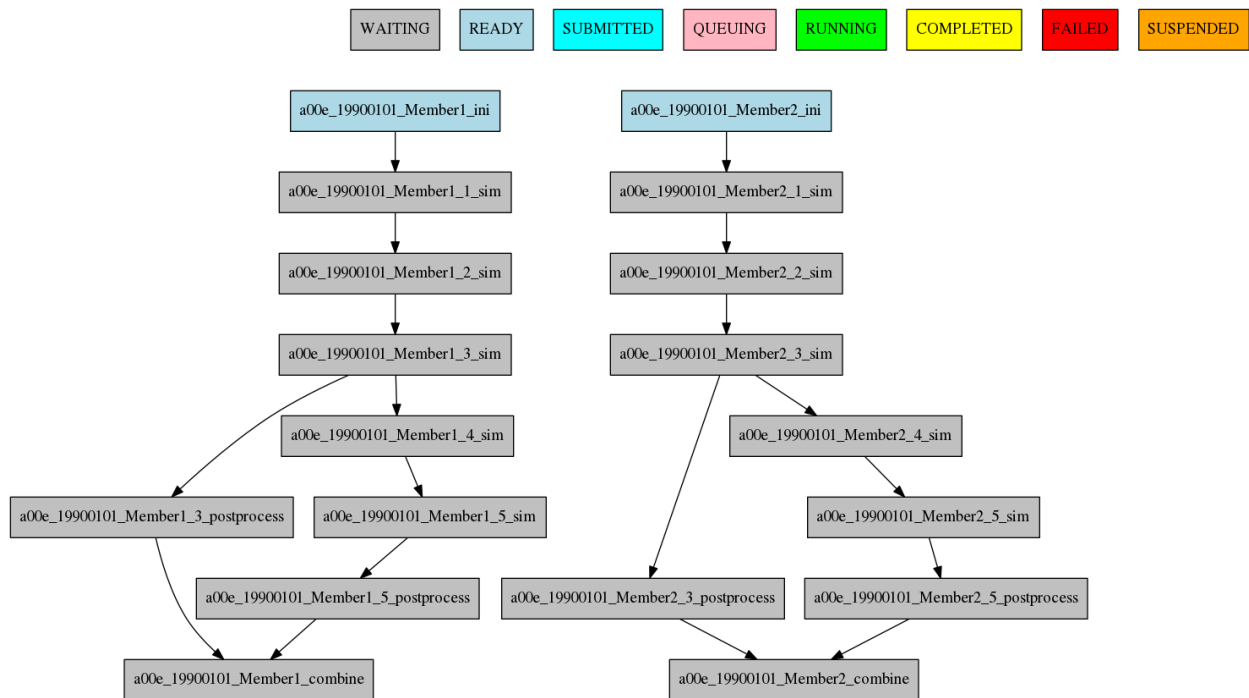
[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[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.

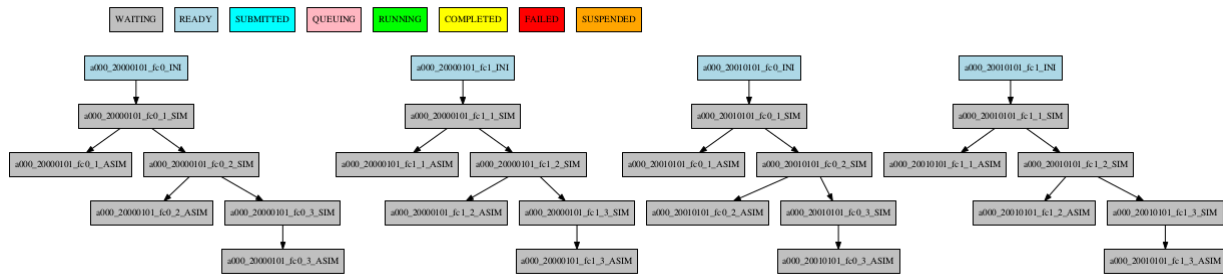
**Hint:** This job parameter works with jobs with RUNNING parameter equals to ‘chunk’.

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

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

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

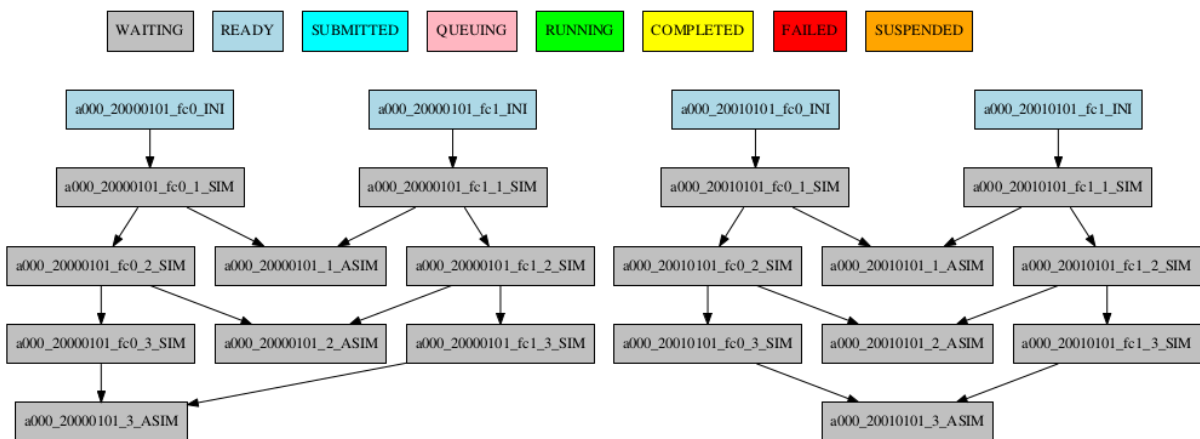
The resulting workflow can be seen in Figure 5.6



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

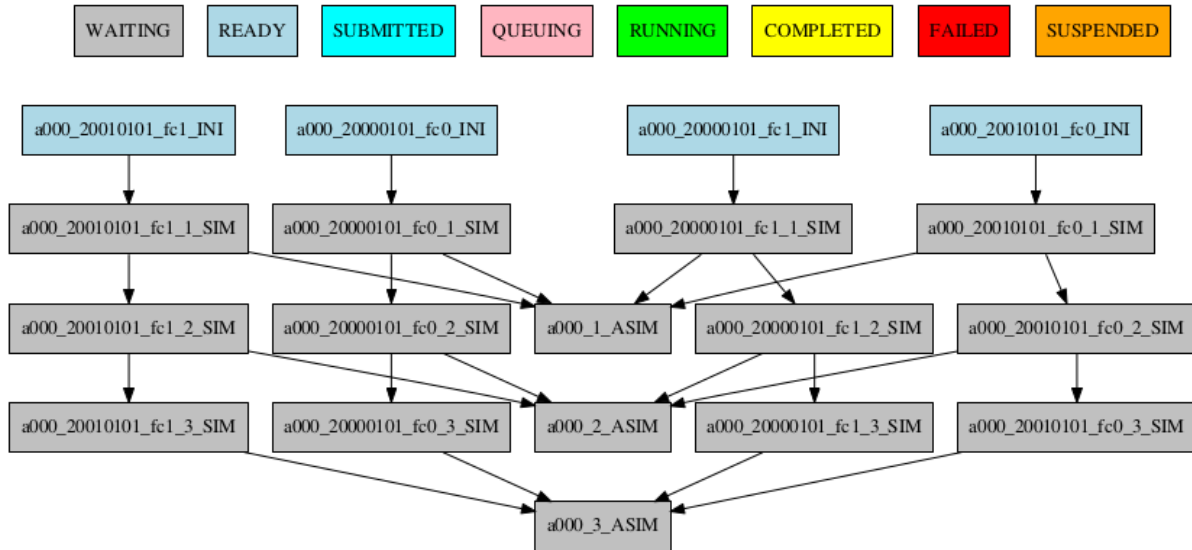


5.7: Example showing dependencies between chunk jobs running with member synchronize.



```
[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)

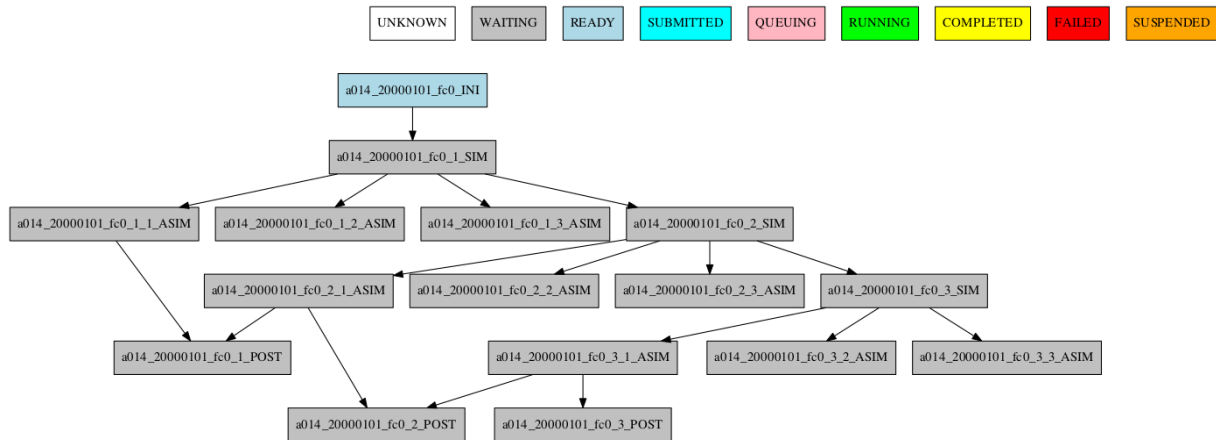
(continued from previous page)

```

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

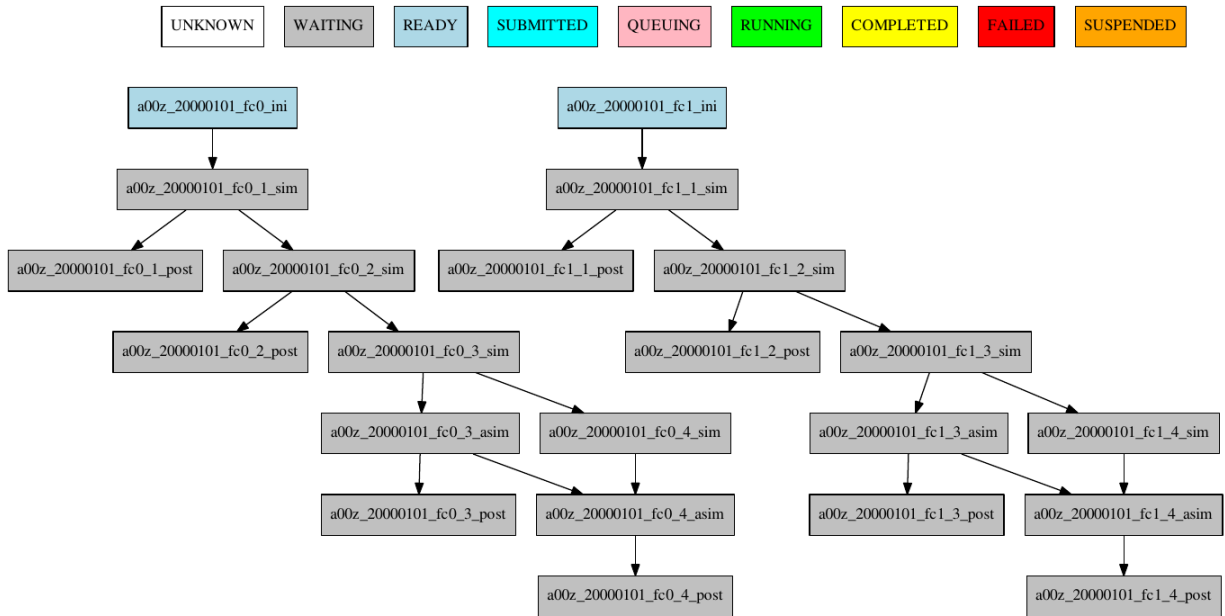
[sim]
FILE = sim.sh
DEPENDENCIES = ini sim-1
RUNNING = chunk

[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.



## 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.



### 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 Error on create caused by a configuration parsing error

When running create you can come across an error similar to:

```
[ERROR] Trace: '%' must be followed by '%' or '(', found: u'%HPCROOTDIR%/remoteconfig/
↳%CURRENT_ARCH%_launcher.sh'
```

The important part of this error is the message '%' must be followed by '%'. It indicated that the source of the error is the `configparser` library. This library is included in the python common libraries, so you shouldn't have any other version of it installed in your environment. Execute `pip list`, if you see `configparser` in the list, then run `pip uninstall configparser`. Then, try to create your experiment again.

## 7.6 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.



## 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.
7052	Scheduler is not installed or correctly configured.	Check if there is a scheduler installed in the remote machine.

### 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
7070	Bug in code	Contact us via git/e-mail output

### 8.5.1 Default Solution

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

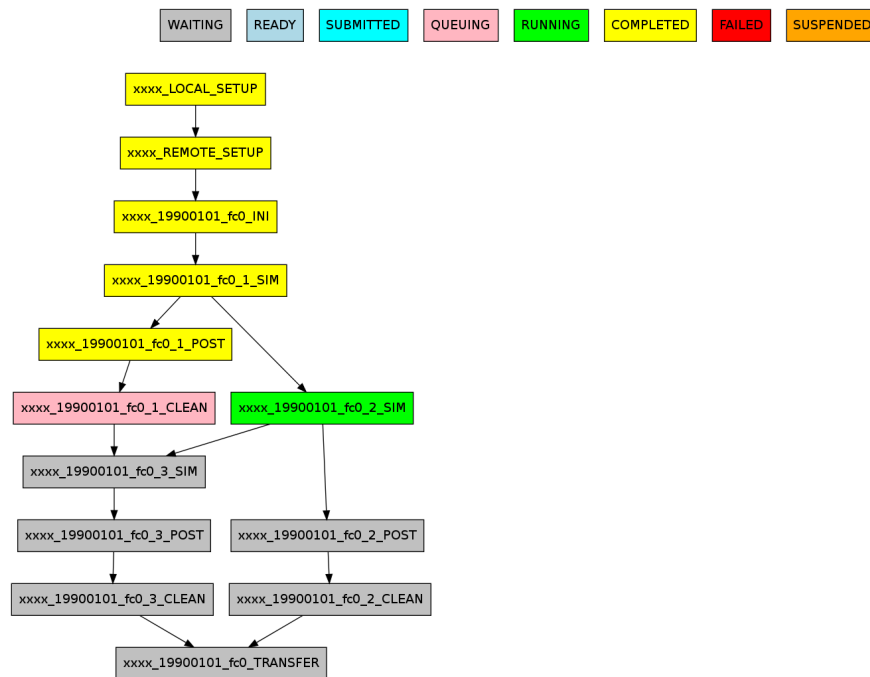
---

## 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 reconnection	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

## 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 introduces 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.

<p><b>Warning:</b> 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. <code>exit 0</code>.</p>
--

---

**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>

---

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.
- **NUMTASKS**: Number of tasks that the job will use.
- **HYPERTHREADING**: Detects if hyperthreading is enabled or not.
- **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 platofrms 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
- **HPVERSION:** 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 chunk 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>.



## 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*, *nottransitive*=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
- **txt\_only**(*bool*) – workflow will only be written as text
- **group\_by**(*bool*) – workflow will only be written as text
- **expand**(*str*) – Filtering of jobs for it's visualization
- **expand\_status**(*str*) – Filtering of jobs for it's visualization
- **hide\_groups**(*bool*) – Simplified workflow illustration by encapsulating the jobs.
- **notransitive**(*bool*) – workflow will only be written as text
- **check\_wrapper**(*bool*) – Shows a preview of how the wrappers will look
- **notransitive** – Some dependencies will be omitted
- **detail**(*bool*) – better text format representation but more expensive

**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=[]*, *nottransitive=False*, *no\_recover\_logs=False*, *detail=False*, *force=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
- **force** (*bool*) – Allows to restore the workflow even if there are running jobs

**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*, *nottransitive=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=[]*, *nottransitive=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
- **notransitive** – Reduces workflow linkage complexity

**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 KeyboardInterrupt 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 finds 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 finds any error

**Return type** `bool`

**check\_jobs\_conf**()

Checks experiment's jobs configuration file.

**Returns** True if everything is correct, False if it finds 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 finds 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\_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\_delay\_retry\_time** ()  
 Returns delay time from autosubmit's config file  
**Returns** safety sleep time  
**Return type** int

**get\_dependencies** (*section='None'*)  
 Returns dependencies list from jobs config file  
**Returns** experiment's members  
**Return type** list

**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\_extensible\_wallclock** (*wrapper\_section\_name='wrapper'*)  
 Gets extend\_wallclock for the given wrapper :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\_hyperthreading(*section*)**

Gets hyperthreading status :param section: job type :type section: str :return: true/false/none :rtype: 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** (*wrapper\_section\_name*='wrapper')

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\_max\_wrapped\_jobs\_horizontal** (*wrapper\_section\_name*='wrapper')

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\_max\_wrapped\_jobs\_vertical** (*wrapper\_section\_name*='wrapper')

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** (*wrapper\_section\_name='wrapper'*)

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\_min\_wrapped\_jobs\_horizontal** (*wrapper\_section\_name='wrapper'*)

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\_min\_wrapped\_jobs\_vertical** (*wrapper\_section\_name='wrapper'*)

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\_notifications** ()

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

**Returns** if notifications

**Return type** string

**get\_notifications\_crash** ()

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** rerun value

**Return type** list

**get\_rerun\_jobs** ()

Returns rerun jobs

**Returns** jobs\_list

**Return type** str

**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** (*wrapper\_section\_name='wrapper'*)

Returns time to check the status of jobs in the wrapper

**Returns** wrapper check time

**Return type** int

**get\_wrapper\_export** (*wrapper\_section\_name='wrapper'*)

Returns modules variable from wrapper

**Returns** string

**Return type** string

**get\_wrapper\_jobs** (*wrapper\_section\_name='wrapper'*)

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

**Returns** expression (or none)

**Return type** string

**get\_wrapper\_machinefiles** (*wrapper\_section\_name='wrapper'*)

Returns the strategy for creating the machinefiles in wrapper jobs

**Returns** machinefiles function to use

**Return type** string

**get\_wrapper\_method** (*wrapper\_section\_name='wrapper'*)

Returns the method of make the wrapper

**Returns** method

**Return type** string

**get\_wrapper\_multi** ()

return the section name of the wrappers

**Returns** wrapper section list

**Return type** string

**get\_wrapper\_policy** (*wrapper\_section\_name='wrapper'*)

Returns what kind of policy (flexible, strict, mixed ) the user has configured in the autosubmit's config

**Returns** wrapper type (or none)

**Return type** string

**get\_wrapper\_queue** (*wrapper\_section\_name='wrapper'*)

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\_retrials** (*wrapper\_section\_name='wrapper'*)

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

**Returns** safety sleep time

**Return type** int

**get\_wrapper\_type** (*wrapper\_section\_name='wrapper'*)

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

**Returns** wrapper type (or none)

**Return type** string

**get\_x11** (*section*)

Active X11 for this section :param section: job type :type section: str :return: false/true :rtype: str

**get\_x11\_jobs** ()

Returns the jobs that should support x11, configured in the autosubmit's config

**Returns** expression (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\_exp\_id**(*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 experiment :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. Anti-lock version.

**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. Anti-lock version.

**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 minimum autosubmit version needed for the experiment. Anti-lock version.

**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. Anti-lock version.

**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. Anti-lock version.

**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. Anti-lock 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 uncommitted 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: date-time.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

**children\_names\_str**

Comma separated list of children's names

**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.

**status\_str**

String representation of the current status

**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, enabled=False)**

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(enabled=False)**

Writes start date and time to TOTAL\_STATS file :return: True if succesful, False otherwise :rtype: bool

**write\_submit\_time(enabled=False, hold=False)**

Writes submit date and time to TOTAL\_STATS file. It doesn't write if hold == True.

**write\_total\_stat\_by\_retries(total\_stats, first\_retrial=False)**

Writes all data to TOTAL\_STATS file :param total\_stats: data gathered by the wrapper :type completed: str

**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 (*version='2'*)

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 and 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={}, new=True, notransitive=False, update\_structure=False, run\_only\_members=[], show\_log=True*)

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 each 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\_delayed** (*platform=None*)

Returns a list of delayed jobs

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

**Returns** delayed jobs

**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="", fil-  
ter\_jobs\_by\_section="", two\_step\_start=True*)

**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\_jobs\_by\_section** (*section\_list*)

Returns the job that its name matches parameter section

**Parameters** **name** – name to look for

**Returns** found job

**Return type** job

**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** (*section*)

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 skipped jobs

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

**Returns** skipped 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\_uncompleted\_and\_not\_waiting** (*platform=None, wrapper=False*)

Returns a list of completed jobs and waiting

**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 unsubmitted 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(nottransitive=False)**

Removes all jobs to be run only in reruns

### **rerun(job\_list\_unparsed, monitor=False)**

Updates job list to rerun the jobs specified by a job list

**Parameters** **chunk\_list** (*str*) – list of chunks to rerun

### **static retrieve\_packages(BasicConfig, expid, current\_jobs=None)**

Retrieves dictionaries that map the collection of packages in the experiment

#### **Parameters**

- **BasicConfig** (*Configuration Object*) – Basic configuration
- **expid** (*String*) – Experiment Id
- **current\_jobs** (*list*) – list of names of current jobs

**Returns** job to package, package to jobs, package to package\_id, package to symbol

**Return type** Dictionary(Job Object, Package), Dictionary(Package, List of Job Objects), Dictionary(String, String), Dictionary(String, String)

**static retrieve\_times** (*status\_code, name, tmp\_path, make\_exception=False, job\_times=None, seconds=False, job\_data\_collection=None*)

Retrieve job timestamps from database. :param status\_code: Code of the Status of the job :type status\_code: Integer :param name: Name of the job :type name: String :param tmp\_path: Path to the tmp folder of the experiment :type tmp\_path: String :param make\_exception: flag for testing purposes :type make\_exception: Boolean :param job\_times: Detail from as\_times.job\_times for the experiment :type job\_times: Dictionary Key: job name, Value: 5-tuple (submit time, start time, finish time, status, detail id) :return: minutes the job has been queuing, minutes the job has been running, and the text that represents it :rtype: int, int, str

**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, queue\_time\_fixes=None*)

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, classic\_txt=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. Options with values larger than GENERAL\_STATS\_OPTION\_MAX\_LENGTH characters are not added.

**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 not exist

**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*, *x11=False*)  
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*, *x11=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*, *x11=False*)

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*, *x11=False*)

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, x11=False*)

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*, *x11=False*)

Parses submit command output to extract job id :param output: output to parse :type output: str :return: job id :rtype: str

**open\_submit\_script** ()

Opens Submit script file

**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*, *x11=False*)

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, x11=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:

Autosubmit Searcher
Home About

Search Experiments by Expid or Description...
Search Running

Show Detailed Data Clear

**a0yh**
11 / 12
RUNNING

Owner: molid  
MONARCH - Global Aerosol 1.5 Operational simulation

Summary More

HPC: marenostrum4

**a26q**
1612 / 2403
RUNNING

Owner: tarsouze  
Control exprimet of EC-Earth3.2-VHR experiment for PRIMAVERA. Follows the 50 years spin-up of experiment a142.

Summary More

HPC: marenostrum4

**a28v**
1026 / 2403
RUNNING

Owner: tarsouze  
Historical exprimet of EC-Earth3.2-VHR experiment for PRIMAVERA. Follows the 50 years spin-up of experiment a142.

Summary More

HPC: marenostrum4

**a2at**
178 / 526
RUNNING

Owner: jgarci1  
extension of a2al

Summary More

HPC: marenostrum4

**a2au**
186 / 526
RUNNING

Owner: fpalmeir  
Extension of a2am

Summary More

HPC: marenostrum4

**a2d1**
7649 / 12781
RUNNING

Owner: mguevara  
auto-CALIOPE test

Summary More

HPC: marenostrum4

## 12.2: Search Result

Autosubmit Searcher
Home About

Search Experiments by Expid or Description...
Search Running

Show Detailed Data Clear

**a0yh**
11 / 12
RUNNING

Owner: molid  
MONARCH - Global Aerosol 1.5 Operational simulation

Refresh More

Avg. Queue 3 min. Avg. Run 9 min.

HPC: marenostrum4

**a26q**
1612 / 2403
RUNNING

Owner: tarsouze  
Control exprimet of EC-Earth3.2-VHR experiment for PRIMAVERA. Follows the 50 years spin-up of experiment a142.

Refresh More

Avg. Queue 212 min. Avg. Run 82 min.

Running: 1 Submitted: 1

HPC: marenostrum4

**a28v**
1026 / 2403
RUNNING

Owner: tarsouze  
Historical exprimet of EC-Earth3.2-VHR experiment for PRIMAVERA. Follows the 50 years spin-up of experiment a142.

Refresh More

Avg. Queue 209 min. Avg. Run 87 min.

Running: 1 Failed: 1

- a28v\_19500101\_fc0\_252\_CLEAN

HPC: marenostrum4

**a2at**
178 / 526
RUNNING

Owner: jgarci1  
extension of a2al

Refresh More

Avg. Queue 44 min. Avg. Run 41 min.

Running: 1

HPC: marenostrum4

**a2au**
186 / 526
RUNNING

Owner: fpalmeir  
Extension of a2am

Refresh More

Avg. Queue 38 min. Avg. Run 47 min.

Running: 1

HPC: marenostrum4

**a2d1**
7649 / 12781
RUNNING

Owner: mguevara  
auto-CALIOPE test

Refresh More

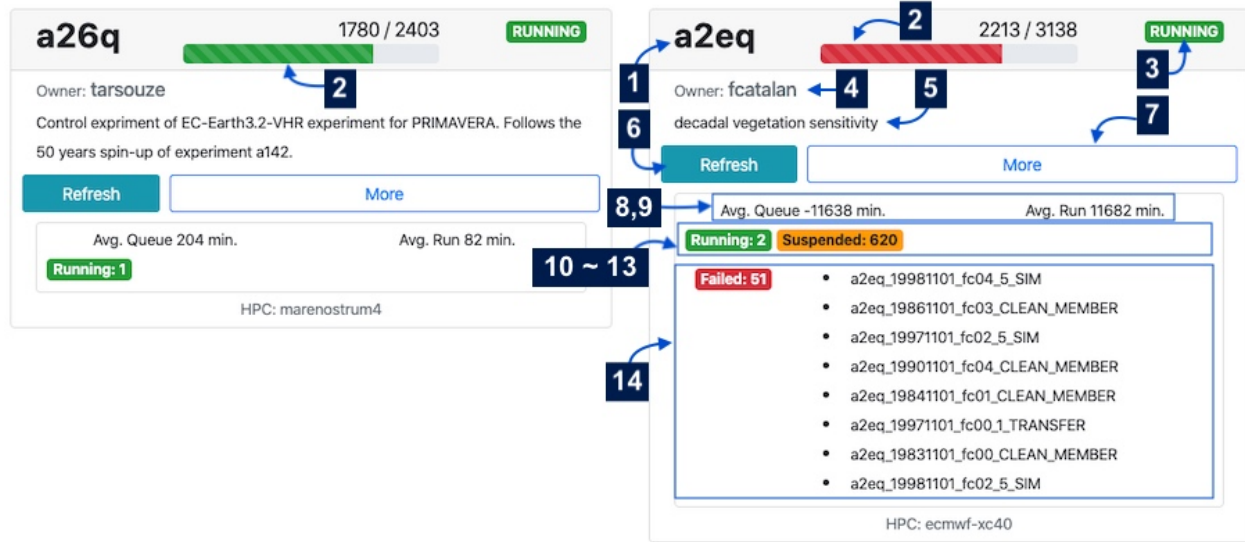
Avg. Queue 5 min. Avg. Run 15 min.

Running: 2

HPC: marenostrum4

## 12.3: Search Result plus Detailed Data



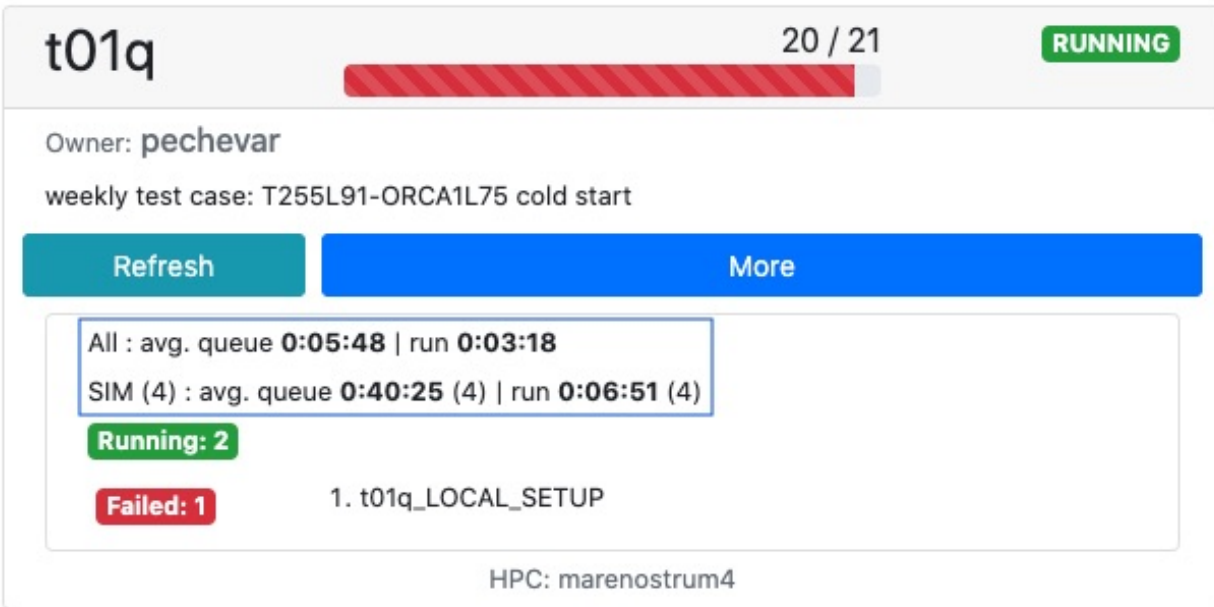


12.4: Description of Detailed Data

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 detailed 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 is 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.

Autosubmit Searcher Home About **t06v** ACTIVE esarchive 86.00 MB/s 3.87 s Search Experiments Search

Tree View Graph Log Statistics Performance Quick View FAQ

Filter text Filter Clear Change Status Total #Jobs: 22 | Chunk unit: day | Chunk size: 1

Activate Selection Mode Clear Tree Refresh Start Job Monitor

Here goes the Job id  
Select a Node to see more information.

t06v\_20160625  
 t06v\_20160625\_000 3 / 17 COMPLETED  
 t06v\_20160625\_000\_1\_LOCAL\_SEND\_INITIAL #COMPLETED ~ ( 0:00:18 ) + 0:01:00  
 t06v\_20160625\_000\_1\_LOCAL\_SEND\_INITIAL\_EMISSIONS #COMPLETED ~ ( 0:00:19 ) + 0:02:51  
 t06v\_20160625\_000\_1\_PREPROCVAR #WAITING  
 t06v\_20160625\_000\_1\_SIM #WAITING  
 t06v\_20160625\_000\_1\_REDUCE #WAITING  
 t06v\_20160625\_000\_1\_ARCHIVE #WAITING  
 t06v\_20160625\_1\_ARCHIVE\_REDUCE #WAITING SYNC  
 t06v\_20160625\_000\_1\_CLEAN #WAITING TARGET  
 t06v\_20160625\_000\_2\_LOCAL\_SEND\_INITIAL #COMPLETED ~ ( 0:00:13 ) + 0:01:00  
 t06v\_20160625\_000\_2\_LOCAL\_SEND\_INITIAL\_EMISSIONS #WAITING  
 t06v\_20160625\_000\_2\_PREPROCVAR #WAITING  
 t06v\_20160625\_000\_2\_HERMES #WAITING  
 t06v\_20160625\_000\_2\_SIM #WAITING  
 t06v\_20160625\_000\_2\_REDUCE #WAITING  
 t06v\_20160625\_000\_2\_ARCHIVE #WAITING  
 t06v\_20160625\_2\_ARCHIVE\_REDUCE #WAITING SYNC  
 t06v\_20160625\_000\_2\_CLEAN #WAITING TARGET  
 Keys  
 t06v\_LOCAL\_SETUP #COMPLETED ~ ( 0:00:12 ) + 0:00:00 SOURCE  
 t06v\_LOCAL\_SEND\_SOURCE #COMPLETED ~ ( 0:00:16 ) + 0:02:51  
 t06v\_LOCAL\_SEND\_STATIC #COMPLETED ~ ( 0:00:14 ) + 0:00:00  
 t06v\_REMOTE\_COMPILE #RUNNING ~ ( 0:00:03 ) + 0:12:57  
 t06v\_PREPROCPIX #WAITING

auto-monarch weekly test case: REGIONAL\_CHEM cold start | Branch: master | Hpc: marenostrum4 | Owner: 2359 gmontane | Version: 3.13.0b0 | DB: 15 | #Jobs: 22

## 12.6: Experiment Information

Autosubmit Searcher Home About **a3ll** ACTIVE esarchive 86.00 MB/s 3.87 s Search Experiments Search

Tree View Graph Log Statistics Performance Quick View FAQ

Filter text Filter Clear Change Status Total #Jobs: 14 | Chunk unit: day | Chunk size: 1

Activate Selection Mode Clear Tree Refresh Start Job Monitor

a3ll\_20210317  
 a3ll\_20210317\_000 7 / 10 COMPLETED 1 RUNNING  
 a3ll\_20210317\_000\_LOCAL\_SEND\_SPINUP #COMPLETED ~ ( 0:00:14 ) + 0:00:47 SOURCE  
 a3ll\_20210317\_000\_1\_LOCAL\_SEND\_INITIAL #COMPLETED ~ ( 0:00:18 ) + 0:00:44 SOURCE  
 a3ll\_20210317\_000\_1\_PREPROCVAR #COMPLETED ~ ( 0:01:43 ) + 0:07:29  
 a3ll\_20210317\_000\_1\_SIM #COMPLETED ~ ( 0:00:22 ) + 0:42:15  
 a3ll\_20210317\_000\_1\_REDUCE #COMPLETED ~ ( 0:08:12 ) + 0:25:46  
 a3ll\_20210317\_000\_1\_ARCHIVE #COMPLETED ~ ( 0:00:16 ) + 0:00:50  
 a3ll\_20210317\_000\_1\_IT #RUNNING ~ ( 0:00:05 ) + 0:35:44  
 a3ll\_20210317\_1\_ARCHIVE\_IT #WAITING SYNC  
 a3ll\_20210317\_1\_ARCHIVE\_REDUCE #COMPLETED ~ ( 0:00:19 ) + 0:02:54 SYNC  
 a3ll\_20210317\_000\_1\_CLEAN #WAITING TARGET  
 Keys  
 a3ll\_LOCAL\_SEND\_SOURCE #COMPLETED ~ ( 0:00:13 ) + 0:03:53 SOURCE  
 a3ll\_LOCAL\_SEND\_STATIC #COMPLETED ~ ( 0:00:15 ) + 0:00:00 SOURCE  
 a3ll\_REMOTE\_COMPILE #COMPLETED ~ ( 0:00:05 ) + 0:18:20  
 a3ll\_PREPROCPIX #COMPLETED ~ ( 0:00:58 ) + 0:12:30

a3ll\_20210317\_000\_1\_SIM  
 Start: 2021 03 17 End: 2021 03 18  
 Section: SIM  
 Member: 000 Chunk: 1  
 Platform: marenostrum4 Id: 14853173  
 Processors: 260 Wallclock: 02:00  
 Queue: 00:00:22 Run: 00:42:15  
 Status: COMPLETED Out: 1 In: 2  
 /esarchive/autosubmit/a3ll/tmp Copy out  
 /esarchive/autosubmit/a3ll/tmp Copy err  
 Submit: 2021-03-19 16:41:48 SYPD: 0.09  
 Start: 2021-03-19 16:42:10  
 Finish: 2021-03-19 17:24:25

## 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 functionality 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.

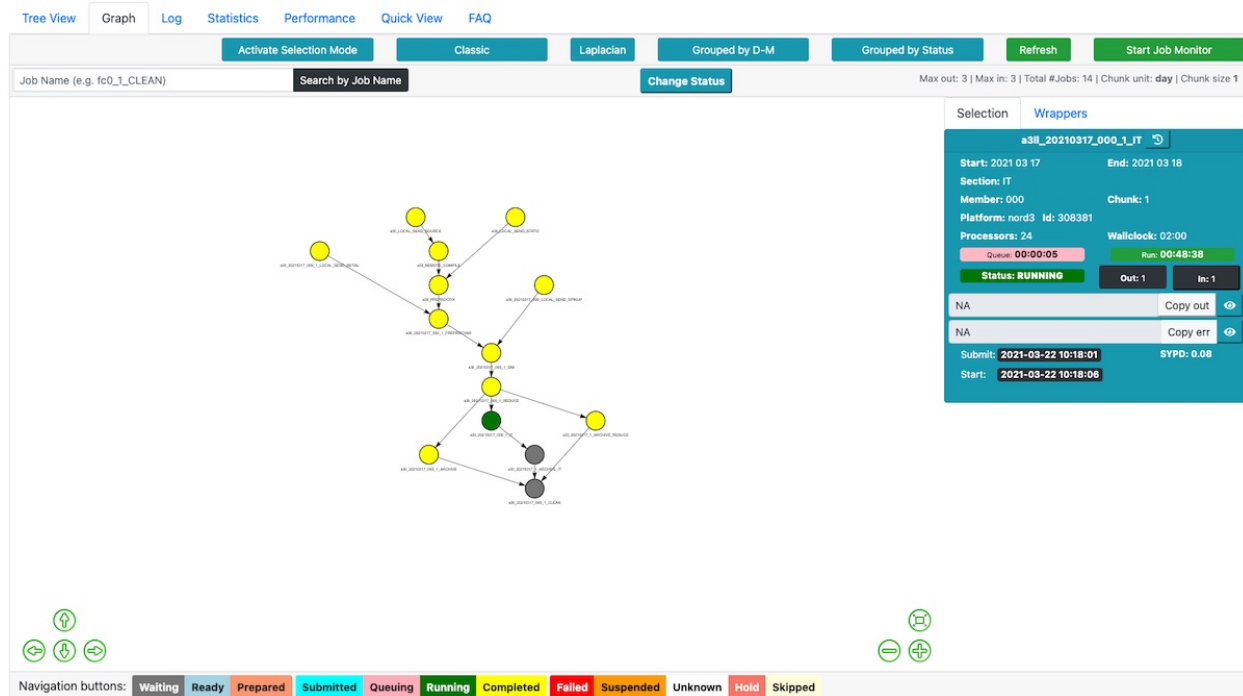
This view offers a graph representation of the experiments where a node represents a job and an edge represents a directed dependency relationship between nodes. To open it you must click on the button `Classic`, which is the basic representation that uses either `GraphViz` or an heuristic approach depending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

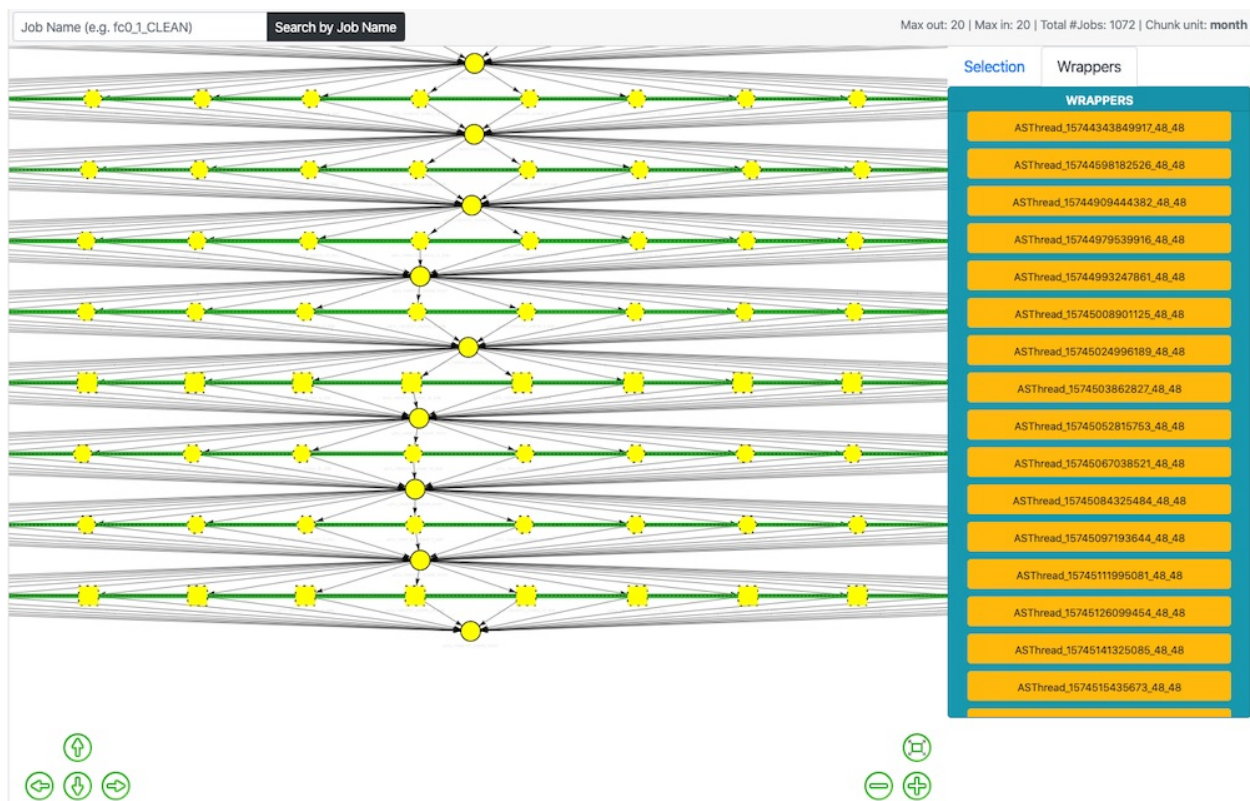
## Selection

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

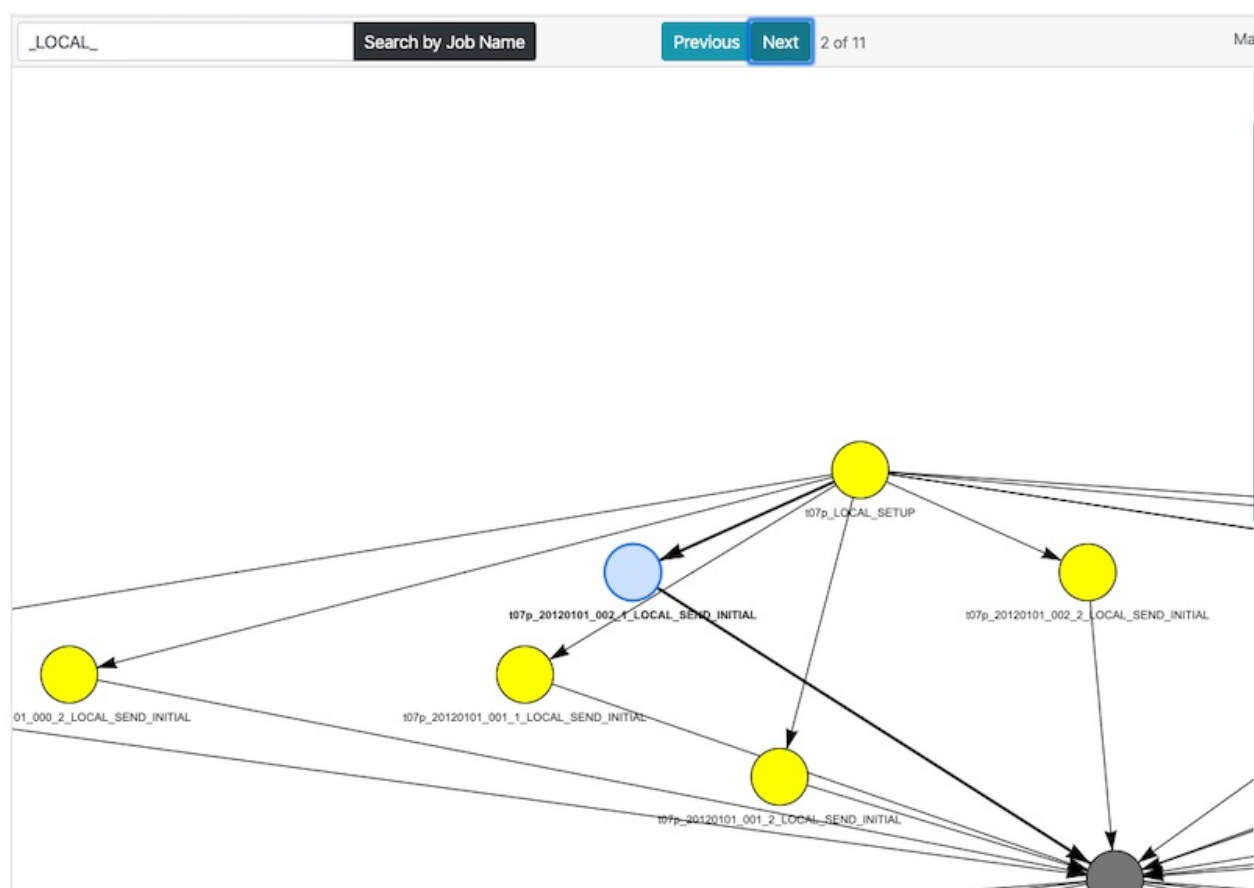
Wrappers are represented by nodes that have dashed border, hexagon or square shape (no difference between them), and that share green background edges. On the right side of the graph you can find the **Wrappers Tab** and it will display a list of the existing wrappers as buttons. If you click on any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

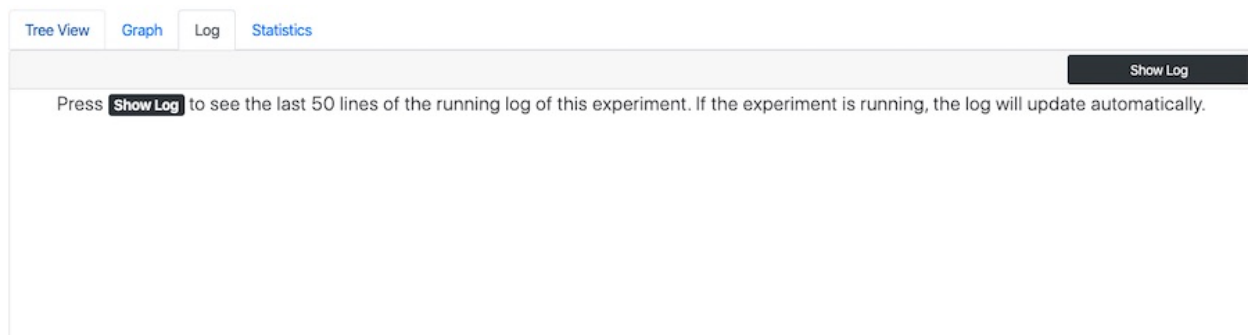
By clicking on the button `Grouped by Status` you get a graph representation where the nodes are clustered by status into 3 clusters: `WAITING`, `COMPLETED`, and `SUSPENDED`. Same rules mentioned for **Grouped by Date Member** apply.

## 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.

```

Tree View Graph Log Statistics
Hide Log
Logfile: 20200324.101849_run.log (1585215304.9314954) Last Modified: 2020-03-26 09:35:04
2020-03-26 10:34:49,025 Number of jobs ready: 0
2020-03-26 10:34:49,025 Number of jobs available: 30
2020-03-26 10:34:49,028
Jobs ready for power9: 0
2020-03-26 10:34:49,033 Number of jobs ready: 0
2020-03-26 10:34:49,033 Number of jobs available: 30
2020-03-26 10:34:49,036
Jobs ready for marenosturm4: 0
2020-03-26 10:34:49,041 Number of jobs ready: 0
2020-03-26 10:34:49,041 Number of jobs available: 30
2020-03-26 10:34:59,046 Reloading parameters...
2020-03-26 10:34:59,052 Loading parameters...
2020-03-26 10:34:59,053 Loading project parameters...
2020-03-26 10:34:59,053
304 of 806 jobs remaining (10:34)
2020-03-26 10:34:59,054 Sleep: 10
2020-03-26 10:34:59,054 Number of retries: 3
2020-03-26 10:34:59,054 WRAPPER CHECK TIME = 10
2020-03-26 10:35:02,026 Command sacct -n -j 9357754 -o "State" in mn4 successful with out message: RUNNING
RUNNING
RUNNING
2020-03-26 10:35:02,026 Successful check job command: sacct -n -j 9357754 -o "State"
2020-03-26 10:35:02,026 Output RUNNING
RUNNING
RUNNING
2020-03-26 10:35:02,027 Job a2ke_20170725_000_60_SIM is RUNNING
2020-03-26 10:35:02,027 Updating FAILED jobs
2020-03-26 10:35:02,028 Updating WAITING jobs
2020-03-26 10:35:02,028 Update finished
2020-03-26 10:35:02,029
Jobs ready for marenosturm_archive: 0
2020-03-26 10:35:02,035 Number of jobs ready: 0
2020-03-26 10:35:02,035 Number of jobs available: 30
2020-03-26 10:35:02,037
Jobs ready for power9: 0
2020-03-26 10:35:02,042 Number of jobs ready: 0
2020-03-26 10:35:02,042 Number of jobs available: 30
2020-03-26 10:35:02,044
Jobs ready for marenosturm4: 0
2020-03-26 10:35:02,049 Number of jobs ready: 0
2020-03-26 10:35:02,049 Number of jobs available: 30

```

## 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`.

---

Tree View Graph Log Statistics Performance Quick View FAQ

Refresh

Parallelization: 768  
 JPSY: 46238361  
 SYPD: 19.0972  
 RSYD: 11.1681  
 ASYPD: 10.7351  
 CHSY: 965.17

Considered: (97)

#	Job Name	Queue	Run	CHSY	JPSY	Energy	SYPD	ASYPD
1	a3bh_18500101_fc1_10_SIM	00:00:02	01:47:52	1380.69	84950000	84950000	13.35	8.15
2	a3bh_18500101_fc1_11_SIM	00:00:46	01:47:04	1370.45	84200000	84200000	13.45	8.16
3	a3bh_18500101_fc1_12_SIM	00:00:36	01:47:27	1375.36	68440000	68440000	13.4	8.15
4	a3bh_18500101_fc1_13_SIM	00:00:03	01:48:40	1390.93	68980000	68980000	13.25	8.11
5	a3bh_18500101_fc1_14_SIM	00:00:22	01:46:28	1362.77	67990000	67990000	13.53	8.2
6	a3bh_18500101_fc1_15_SIM	00:00:09	01:47:20	1373.87	68730000	68730000	13.42	8.17

There are some warnings about the calculations of performance metrics: [Show warnings](#)

Metrics description:

**Parallelization:** Total number of cores allocated for the run, per SIM.

**JPSY:** Energy cost of a simulation, measured in Joules per simulated year.

**SYPD:** Simulated years per day for the model in a 24 h period.

**ASYPD:** Actual SYPD, this number should be lower than SYPD due to interruptions, queue wait time, data transfer or issues with the model workflow. This is collected by measuring the time between first submission and the date of arrival of the last history file on the storage file system.

**CHSY:** Core hours per simulated year. This is measured as the product of the model runtime for 1 SY and the number of cores allocated. This is an average of the CHSY of all SIM jobs.

**Considered:** Scrollable list where each item in the list represents job information showing **Job Name**, **QUEUE** and **RUNNING** time in *HH:mm:ss* format, **CHSY**, **JPSY**, and raw **Energy** consumption for that job. *Note: Energy values are only collected for those jobs running on MareNostrum4 and using the latest version of Autosubmit. Subsequent development will expand this feature for other platforms.*

Visit [Performance Metrics Documentation](#) for more details.

## 12.13: Performance Metrics Tab

Tree View Graph Log Statistics

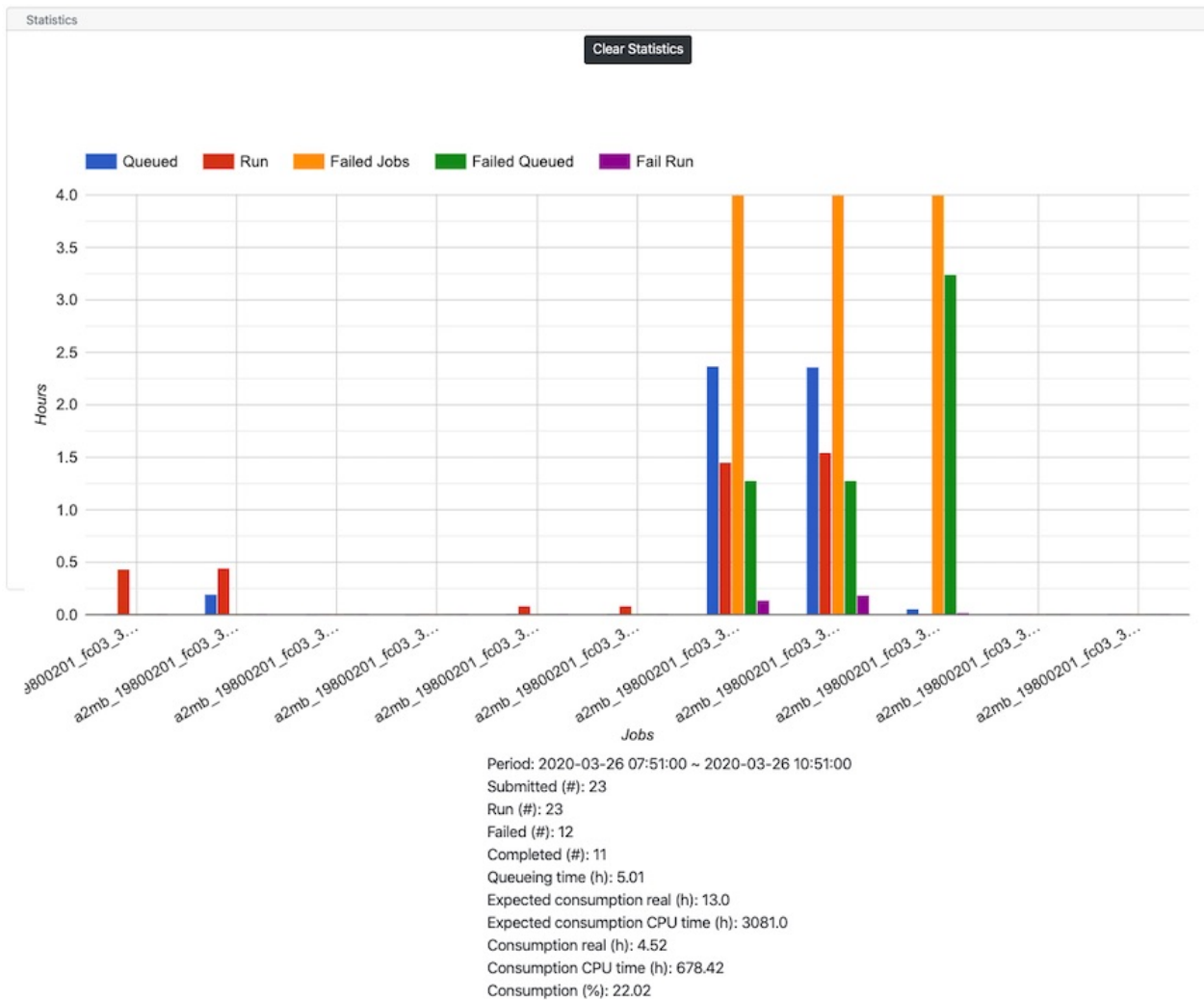
Statistics

Section Hours [Get Statistics](#)

Supply a Section (Type) in the appropriate textbox to filter the jobs that will be included in the query. Also, you can also supply the Hours value that determines how many hours before the current time you want to query. Leave both empty and a query for Any Section since the date of creation of the experiment will be executed.

Press [Get Statistics](#) to generate the statistics, this will generate a Bar Chart and some extra statistics below. Drag the mouse inside the chart to zoom in; however, zoom in capabilities are not unlimited, so try to narrow your query.

## 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 file called `structure_expid.db` where `expid` is the name of your experiment. This file will be located in `/esarchive/autosubmit/expid/pkl/`.

---



### a

- `autosubmit.autosubmit`, 47
- `autosubmit.config.basicConfig`, 54
- `autosubmit.config.config_common`, 54
- `autosubmit.database.db_common`, 64
- `autosubmit.git.autosubmit_git`, 66
- `autosubmit.job.job`, 66
- `autosubmit.job.job_common`, 71
- `autosubmit.job.job_list`, 71
- `autosubmit.monitor.monitor`, 79
- `autosubmit.platforms.ecplatform`, 80
- `autosubmit.platforms.locplatform`, 87
- `autosubmit.platforms.lsfplatform`, 82
- `autosubmit.platforms.pbsplatform`, 83
- `autosubmit.platforms.sgeplatform`, 84
- `autosubmit.platforms.slurmplatform`, 85





## A

`add_logs()` (*autosubmit.job.job\_list.JobList* method), 71  
`add_parent()` (*autosubmit.job.job.Job* method), 67  
`archive()` (*autosubmit.autosubmit.Autosubmit* static method), 47  
`Autosubmit` (class in *autosubmit.autosubmit*), 47  
`autosubmit.autosubmit` (module), 47  
`autosubmit.config.basicConfig` (module), 54  
`autosubmit.config.config_common` (module), 54  
`autosubmit.database.db_common` (module), 64  
`autosubmit.git.autosubmit_git` (module), 66  
`autosubmit.job.job` (module), 66  
`autosubmit.job.job_common` (module), 71  
`autosubmit.job.job_list` (module), 71  
`autosubmit.monitor.monitor` (module), 79  
`autosubmit.platforms.ecplatform` (module), 80  
`autosubmit.platforms.locplatform` (module), 87  
`autosubmit.platforms.lsfplatform` (module), 82  
`autosubmit.platforms.pbsplatform` (module), 83  
`autosubmit.platforms.sgeplatform` (module), 84  
`autosubmit.platforms.slurmplatform` (module), 85  
`AutosubmitConfig` (class in *autosubmit.config.config\_common*), 54  
`AutosubmitGit` (class in *autosubmit.git.autosubmit\_git*), 66

## B

`backup_load()` (*autosubmit.job.job\_list.JobList* method), 72  
`backup_save()` (*autosubmit.job.job\_list.JobList* method), 72

`BasicConfig` (class in *autosubmit.config.basicConfig*), 54

## 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), 66  
`check_completion()` (*autosubmit.job.job.Job* method), 67  
`check_conf_files()` (*autosubmit.config.config\_common.AutosubmitConfig* method), 54  
`check_db()` (in module *autosubmit.database.db\_common*), 64  
`check_end_time()` (*autosubmit.job.job.Job* method), 67  
`check_expdef_conf()` (*autosubmit.config.config\_common.AutosubmitConfig* method), 54  
`check_experiment_exists()` (in module *autosubmit.database.db\_common*), 64  
`check_file_exists()` (*autosubmit.platforms.locplatform.LocalPlatform* method), 87  
`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), 55

check\_retrials\_end\_time() (autosubmit.job.job.Job method), 67

check\_retrials\_start\_time() (autosubmit.job.job.Job method), 67

check\_retrials\_submit\_time() (autosubmit.job.job.Job method), 67

check\_running\_after() (autosubmit.job.job.Job method), 67

check\_script() (autosubmit.job.job.Job method), 67

check\_scripts() (autosubmit.job.job\_list.JobList method), 72

check\_start\_time() (autosubmit.job.job.Job method), 68

check\_started\_after() (autosubmit.job.job.Job method), 68

children (autosubmit.job.job.Job attribute), 68

children\_names\_str (autosubmit.job.job.Job attribute), 68

clean() (autosubmit.autosubmit.Autosubmit static method), 47

clean\_git() (autosubmit.mit.git.autosubmit\_git.AutosubmitGit static method), 66

clean\_plot() (autosubmit.monitor.monitor.Monitor static method), 79

clean\_stats() (autosubmit.monitor.monitor.Monitor static method), 79

clone\_repository() (autosubmit.mit.git.autosubmit\_git.AutosubmitGit static method), 66

close\_conn() (in module autosubmit.database.db\_common), 65

color\_status() (autosubmit.monitor.monitor.Monitor static method), 79

compare\_by\_id() (autosubmit.job.job.Job method), 68

compare\_by\_name() (autosubmit.job.job.Job method), 68

compare\_by\_status() (autosubmit.job.job.Job method), 68

configure() (autosubmit.autosubmit.Autosubmit static method), 48

configure\_dialog() (autosubmit.autosubmit.Autosubmit static method), 48

connect() (autosubmit.mit.platforms.ecplatform.EcPlatform method), 80

connect() (autosub-

mit.platforms.locplatform.LocalPlatform method), 87

connect() (autosubmit.mit.platforms.sgeplatform.SgePlatform method), 84

create() (autosubmit.autosubmit.Autosubmit static method), 48

create\_db() (in module autosubmit.database.db\_common), 65

create\_script() (autosubmit.job.job.Job method), 68

create\_tree\_list() (autosubmit.monitor.monitor.Monitor method), 79

## D

database\_fix() (autosubmit.autosubmit.Autosubmit static method), 48

DbException, 64

delete() (autosubmit.autosubmit.Autosubmit static method), 48

delete\_child() (autosubmit.job.job.Job method), 68

delete\_experiment() (in module autosubmit.database.db\_common), 65

delete\_file() (autosubmit.mit.platforms.ecplatform.EcPlatform method), 80

delete\_file() (autosubmit.mit.platforms.locplatform.LocalPlatform method), 87

delete\_parent() (autosubmit.job.job.Job method), 68

describe() (autosubmit.autosubmit.Autosubmit static method), 49

## E

EcPlatform (class in autosubmit.platforms.ecplatform), 80

experiment\_file (autosubmit.config.config\_common.AutosubmitConfig attribute), 55

expid (autosubmit.job.job\_list.JobList attribute), 72

expid() (autosubmit.autosubmit.Autosubmit static method), 49

## G

generate() (autosubmit.job.job\_list.JobList method), 72

generate\_output() (autosubmit.monitor.monitor.Monitor method), 79

generate\_output\_stats() (autosubmit.monitor.monitor.Monitor method), 79

generate\_output\_txt() (autosubmit.monitor.monitor.Monitor method), 80

`generate_scripts_andor_wrappers()` (*autosubmit.autosubmit.Autosubmit static method*), 49  
`get_active()` (*autosubmit.job.job\_list.JobList method*), 72  
`get_all()` (*autosubmit.job.job\_list.JobList method*), 73  
`get_autosubmit_version()` (*in module autosubmit.database.db\_common*), 65  
`get_checkAlljobs_cmd()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 85  
`get_checkjob_cmd()` (*autosubmit.platforms.ecplatform.EcPlatform method*), 80  
`get_checkjob_cmd()` (*autosubmit.platforms.locplatform.LocalPlatform method*), 87  
`get_checkjob_cmd()` (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 82  
`get_checkjob_cmd()` (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 83  
`get_checkjob_cmd()` (*autosubmit.platforms.sgeplatform.SgePlatform method*), 84  
`get_checkjob_cmd()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 86  
`get_chunk_ini()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 55  
`get_chunk_list()` (*autosubmit.job.job\_list.JobList method*), 73  
`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*), 73  
`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*), 56  
`get_date_list()` (*autosubmit.job.job\_list.JobList method*), 73  
`get_default_job_type()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_delay_retry_time()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_delayed()` (*autosubmit.job.job\_list.JobList method*), 73  
`get_dependencies()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_disable_recovery_threads()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_export()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_extensible_wallclock()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_failed()` (*autosubmit.job.job\_list.JobList method*), 73  
`get_fetch_single_branch()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56  
`get_file()` (*autosubmit.platforms.ecplatform.EcPlatform method*), 80  
`get_file()` (*autosubmit.platforms.locplatform.LocalPlatform method*), 87  
`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*), 73  
`get_full_config_as_dict()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 56

<code>get_full_config_as_json()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 56	<code>get_max_waiting_jobs()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57
<code>get_general_stats()</code>	( <i>autosubmit.monitor.monitor.Monitor</i> static method), 80	<code>get_max_wallclock()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57
<code>get_git_project_branch()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 56	<code>get_max_wrapped_jobs()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57
<code>get_git_project_commit()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_max_wrapped_jobs_horizontal()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_git_project_origin()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_max_wrapped_jobs_vertical()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_git_remote_project_root()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_member_list()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_held_jobs()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 73	<code>get_member_list()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74
<code>get_hyperthreading()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_memory()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_in_queue()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_memory_per_task()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_job_by_name()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_migrate_duplicate()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_job_list()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_migrate_host_to()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_job_names()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_migrate_project_to()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_job_related()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_migrate_user_to()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_jobs_by_section()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_min_wrapped_jobs()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 58
<code>get_jobs_sections()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_min_wrapped_jobs_horizontal()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 59
<code>get_last_retrials()</code>	( <i>autosubmit.job.job.Job</i> method), 69	<code>get_min_wrapped_jobs_vertical()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 59
<code>get_local_project_path()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57	<code>get_mkdir_cmd()</code>	( <i>autosubmit.platforms.ecplatform.EcPlatform</i> method), 81
<code>get_logs()</code>	( <i>autosubmit.job.job_list.JobList</i> method), 74	<code>get_mkdir_cmd()</code>	( <i>autosubmit.platforms.locplatform.LocalPlatform</i> method), 88
<code>get_logs_files()</code>	( <i>autosubmit.platforms.locplatform.LocalPlatform</i> method), 88	<code>get_mkdir_cmd()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57
<code>get_mails_to()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57		
<code>get_max_processors()</code>	( <i>autosubmit.config.config_common.AutosubmitConfig</i> method), 57		

*mit.platforms.lsfplatform.LsfPlatform* method), 82

*get\_mkdir\_cmd()* (*autosubmit.platforms.pbsplatform.PBSPlatform* method), 83

*get\_mkdir\_cmd()* (*autosubmit.platforms.sgeplatform.SgePlatform* method), 84

*get\_mkdir\_cmd()* (*autosubmit.platforms.slurmplatform.SlurmPlatform* method), 86

*get\_not\_in\_queue()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_notifications()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 59

*get\_notifications\_crash()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 59

*get\_num\_chunks()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 59

*get\_ordered\_jobs\_by\_date\_member()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_output\_type()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 59

*get\_parse\_two\_step\_start()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 59

*get\_parser()* (*autosubmit.config.config\_common.AutosubmitConfig* static method), 59

*get\_platform()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_prepared()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_processors()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_project\_destination()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_project\_dir()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_project\_type()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_queuing()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_ready()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_remote\_dependencies()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_rerun()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_rerun\_jobs()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_retrials()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_running()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_safetysleeptime()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 60

*get\_scratch\_free\_space()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 61

*get\_skipped()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_ssh\_output()* (*autosubmit.platforms.ecplatform.EcPlatform* method), 81

*get\_ssh\_output()* (*autosubmit.platforms.locplatform.LocalPlatform* method), 88

*get\_storage\_type()* (*autosubmit.config.config\_common.AutosubmitConfig* method), 61

*get\_submit\_cmd()* (*autosubmit.platforms.ecplatform.EcPlatform* method), 81

*get\_submit\_cmd()* (*autosubmit.platforms.locplatform.LocalPlatform* method), 88

*get\_submit\_cmd()* (*autosubmit.platforms.lsfplatform.LsfPlatform* method), 82

*get\_submit\_cmd()* (*autosubmit.platforms.pbsplatform.PBSPlatform* method), 83

*get\_submit\_cmd()* (*autosubmit.platforms.sgeplatform.SgePlatform* method), 84

*get\_submit\_cmd()* (*autosubmit.platforms.slurmplatform.SlurmPlatform* method), 86

*get\_submitted()* (*autosubmit.job.job\_list.JobList* method), 75

*get\_submitted\_job\_id()* (*autosubmit.platforms.ecplatform.EcPlatform* method), 81



`get_submitted_job_id()` (*autosubmit.platforms.locplatform.LocalPlatform method*), 88  
`get_submitted_job_id()` (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 83  
`get_submitted_job_id()` (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 84  
`get_submitted_job_id()` (*autosubmit.platforms.sgeplatform.SgePlatform method*), 85  
`get_submitted_job_id()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 86  
`get_submodules_list()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_suspended()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_svn_project_revision()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_svn_project_url()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_synchronize()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_tasks()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_threads()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_total_jobs()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_uncompleted()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_uncompleted_and_not_waiting()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_unknown()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_unsubmitted()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_version()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_waiting()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_waiting_remote_dependencies()` (*autosubmit.job.job\_list.JobList method*), 76  
`get_wallclock()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_wchunkinc()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 61  
`get_wrapper_check_time()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_export()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_jobs()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_machinefiles()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_method()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_multi()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_policy()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_queue()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_retrials()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_wrapper_type()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 62  
`get_x11()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`get_x11_jobs()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`graph` (*autosubmit.job.job\_list.JobList attribute*), 77

## H

`has_children()` (*autosubmit.job.job.Job method*), 69  
`has_parents()` (*autosubmit.job.job.Job method*), 69

## I

`inc_fail_count()` (*autosubmit.job.job.Job method*), 69  
`increase_wallclock_by_chunk()` (*in module autosubmit.job.job\_common*), 71  
`inspect()` (*autosubmit.autosubmit.Autosubmit static method*), 49

`install()` (*autosubmit.autosubmit.Autosubmit static method*), 50  
`is_a_completed_retrial()` (*autosubmit.job.job.Job static method*), 69  
`is_ancestor()` (*autosubmit.job.job.Job method*), 69  
`is_parent()` (*autosubmit.job.job.Job method*), 69

## J

`Job` (*class in autosubmit.job.job*), 66  
`JobList` (*class in autosubmit.job.job\_list*), 71  
`jobs_file` (*autosubmit.config.config\_common.AutosubmitConfig attribute*), 63  
`jobs_in_queue()` (*autosubmit.platforms.ecplatform.EcPlatform method*), 81

## L

`last_name_used()` (*in module autosubmit.database.db\_common*), 65  
`load()` (*autosubmit.job.job\_list.JobList method*), 77  
`load_file()` (*autosubmit.job.job\_list.JobList static method*), 77  
`load_parameters()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`load_platform_parameters()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`load_project_parameters()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`load_section_parameters()` (*autosubmit.config.config\_common.AutosubmitConfig method*), 63  
`LocalPlatform` (*class in autosubmit.platforms.locplatform*), 87  
`log_job()` (*autosubmit.job.job.Job method*), 69  
`long_name` (*autosubmit.job.job.Job attribute*), 69  
`LsfPlatform` (*class in autosubmit.platforms.lsfplatform*), 82

## M

`migrate()` (*autosubmit.autosubmit.Autosubmit static method*), 50  
`Monitor` (*class in autosubmit.monitor.monitor*), 79  
`monitor()` (*autosubmit.autosubmit.Autosubmit static method*), 50  
`move_file()` (*autosubmit.platforms.ecplatform.EcPlatform method*), 81  
`move_file()` (*autosubmit.platforms.locplatform.LocalPlatform method*), 88

## O

`open_conn()` (*in module autosubmit.database.db\_common*), 65  
`open_submit_script()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 86

## P

`parameters` (*autosubmit.job.job\_list.JobList attribute*), 77  
`parents` (*autosubmit.job.job.Job attribute*), 69  
`parse_all_jobs_output()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 86  
`parse_args()` (*autosubmit.autosubmit.Autosubmit static method*), 50  
`parse_job_finish_data()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 86  
`parse_job_output()` (*autosubmit.platforms.ecplatform.EcPlatform method*), 81  
`parse_job_output()` (*autosubmit.platforms.locplatform.LocalPlatform method*), 89  
`parse_job_output()` (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 83  
`parse_job_output()` (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 84  
`parse_job_output()` (*autosubmit.platforms.sgeplatform.SgePlatform method*), 85  
`parse_job_output()` (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 87  
`parse_output_number()` (*in module autosubmit.job.job\_common*), 71  
`PBSPlatform` (*class in autosubmit.platforms.pbsplatform*), 83  
`pkl_fix()` (*autosubmit.autosubmit.Autosubmit static method*), 50  
`platform` (*autosubmit.job.job.Job attribute*), 69  
`platforms_file` (*autosubmit.config.config\_common.AutosubmitConfig attribute*), 63  
`platforms_parser` (*autosubmit.config.config\_common.AutosubmitConfig attribute*), 63  
`print_job()` (*autosubmit.job.job.Job method*), 69  
`print_parameters()` (*autosubmit.job.job.Job method*), 69

`print_with_status()` (autosubmit.job.job\_list.JobList method), 77  
`project_file` (autosubmit.config.config\_common.AutosubmitConfig attribute), 63

## Q

`queue` (autosubmit.job.job.Job attribute), 69

## R

`read()` (autosubmit.config.basicConfig.BasicConfig static method), 54  
`recovery()` (autosubmit.autosubmit.Autosubmit static method), 51  
`refresh()` (autosubmit.autosubmit.Autosubmit static method), 51  
`reload()` (autosubmit.config.config\_common.AutosubmitConfig method), 63  
`remove_redundant_parents()` (autosubmit.job.job.Job method), 70  
`remove_rerun_only_jobs()` (autosubmit.job.job\_list.JobList method), 77  
`report()` (autosubmit.autosubmit.Autosubmit static method), 51  
`rerun()` (autosubmit.job.job\_list.JobList method), 77  
`rerun_recovery()` (autosubmit.autosubmit.Autosubmit static method), 51  
`restore_connection()` (autosubmit.platforms.ecplatform.EcPlatform method), 82  
`restore_connection()` (autosubmit.platforms.sgeplatform.SgePlatform method), 85  
`retrieve_packages()` (autosubmit.job.job\_list.JobList static method), 77  
`retrieve_times()` (autosubmit.job.job\_list.JobList static method), 77  
`run_experiment()` (autosubmit.autosubmit.Autosubmit static method), 51

## S

`save()` (autosubmit.job.job\_list.JobList method), 78  
`save_experiment()` (in module autosubmit.database.db\_common), 65  
`send_command()` (autosubmit.platforms.ecplatform.EcPlatform method), 82  
`send_command()` (autosubmit.platforms.locplatform.LocalPlatform method), 89

`send_file()` (autosubmit.platforms.ecplatform.EcPlatform method), 82  
`send_file()` (autosubmit.platforms.locplatform.LocalPlatform method), 89  
`set_expid()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_git_project_commit()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_new_host()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_new_project()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_new_user()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_platform()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_safetysleeptime()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`set_status()` (autosubmit.autosubmit.Autosubmit static method), 51  
`set_version()` (autosubmit.config.config\_common.AutosubmitConfig method), 64  
`SgePlatform` (class in autosubmit.platforms.sgeplatform), 84  
`signal_handler()` (in module autosubmit.autosubmit), 53  
`signal_handler_create()` (in module autosubmit.autosubmit), 53  
`SlurmPlatform` (class in autosubmit.platforms.slurmplatform), 85  
`sort_by_id()` (autosubmit.job.job\_list.JobList method), 78  
`sort_by_name()` (autosubmit.job.job\_list.JobList method), 78  
`sort_by_status()` (autosubmit.job.job\_list.JobList method), 78  
`sort_by_type()` (autosubmit.job.job\_list.JobList method), 78  
`statistics()` (autosubmit.autosubmit.Autosubmit static method), 52  
`StatisticsSnippetBash` (class in autosubmit.job.job\_common), 71  
`StatisticsSnippetEmpty` (class in autosubmit.job.job\_common), 71



StatisticsSnippetPython (class in *autosubmit.job.job\_common*), 71  
 StatisticsSnippetR (class in *autosubmit.job.job\_common*), 71  
 Status (class in *autosubmit.job.job\_common*), 71  
 status\_str (*autosubmit.job.job.Job* attribute), 70  
 submit\_ready\_jobs() (*autosubmit.autosubmit.Autosubmit* static method), 52  
 submit\_Script() (*autosubmit.platforms.slurmplatform.SlurmPlatform* method), 87

## T

test() (*autosubmit.autosubmit.Autosubmit* static method), 52  
 test\_connection() (*autosubmit.platforms.ecplatform.EcPlatform* method), 82  
 test\_connection() (*autosubmit.platforms.locplatform.LocalPlatform* method), 89  
 test\_connection() (*autosubmit.platforms.sgeplatform.SgePlatform* method), 85  
 testcase() (*autosubmit.autosubmit.Autosubmit* static method), 53  
 total\_processors (*autosubmit.job.job.Job* attribute), 70  
 Type (class in *autosubmit.job.job\_common*), 71

## U

unarchive() (*autosubmit.autosubmit.Autosubmit* static method), 53  
 update\_cmds() (*autosubmit.platforms.ecplatform.EcPlatform* method), 82  
 update\_cmds() (*autosubmit.platforms.locplatform.LocalPlatform* method), 89  
 update\_cmds() (*autosubmit.platforms.lsfplatform.LsfPlatform* method), 83  
 update\_cmds() (*autosubmit.platforms.pbsplatform.PBSPlatform* method), 84  
 update\_cmds() (*autosubmit.platforms.sgeplatform.SgePlatform* method), 85  
 update\_cmds() (*autosubmit.platforms.slurmplatform.SlurmPlatform* method), 87  
 update\_content() (*autosubmit.job.job.Job* method), 70

update\_experiment\_descrip\_version() (in module *autosubmit.database.db\_common*), 65  
 update\_from\_file() (*autosubmit.job.job\_list.JobList* method), 78  
 update\_genealogy() (*autosubmit.job.job\_list.JobList* method), 78  
 update\_list() (*autosubmit.job.job\_list.JobList* method), 78  
 update\_parameters() (*autosubmit.job.job.Job* method), 70  
 update\_status() (*autosubmit.job.job.Job* method), 70  
 update\_version() (*autosubmit.autosubmit.Autosubmit* static method), 53

## W

WrapperJob (class in *autosubmit.job.job*), 70  
 write\_end\_time() (*autosubmit.job.job.Job* method), 70  
 write\_start\_time() (*autosubmit.job.job.Job* method), 70  
 write\_submit\_time() (*autosubmit.job.job.Job* method), 70  
 write\_total\_stat\_by\_retries() (*autosubmit.job.job.Job* method), 70