
autosubmit Documentation

Release 3.2.0

Domingo Manubens - Javier Vegas

January 26, 2016

CONTENTS

1	Introduction	1
1.1	What is Autosubmit ?	1
1.2	Why is Autosubmit needed ?	1
1.3	How does Autosubmit work ?	1
2	Tutorial	5
2.1	Quick start guide	5
3	Installation	15
3.1	How to install	15
3.2	How to configure	15
4	Usage	17
4.1	Command list	17
4.2	How to create an experiment	17
4.3	How to create a copy of an experiment	18
4.4	How to create a dummy experiment	18
4.5	How to configure the experiment	18
4.6	How to check the experiment configuration	19
4.7	How to run the experiment	20
4.8	How to test the experiment	20
4.9	How to monitor the experiment	21
4.10	How to monitor job statistics	22
4.11	How to stop the experiment	22
4.12	How to restart the experiment	22
4.13	How to rerun a part of the experiment	23
4.14	How to clean the experiment	24
4.15	How to refresh the experiment project	24
4.16	How to delete the experiment	25
4.17	How to add a new job	25
4.18	How to add a new platform	26
4.19	How to archive an experiment	27
4.20	How to unarchive an experiment	28
5	Defining the workflow	29
5.1	Simple workflow	29
5.2	Running jobs once per startdate, member or chunk	30
5.3	Dependencies	30
5.4	Job frequency	32
5.5	Rerun dependencies	33

6	Troubleshooting	35
6.1	How to change the job status stopping autosubmit	35
6.2	How to change the job status without stopping autosubmit	36
7	Developing a project	39
8	Variables reference	41
8.1	Job variables	41
8.2	Platform variables	42
8.3	Project variables	42
9	Module documentation	43
9.1	autosubmit	43
9.2	autosubmit.config	43
9.3	autosubmit.database	51
9.4	autosubmit.date	53
9.5	autosubmit.git	56
9.6	autosubmit.job	56
9.7	autosubmit.monitor	64
9.8	autosubmit.platform	66
	Python Module Index	67
	Index	69

INTRODUCTION

1.1 What is Autosubmit ?

Autosubmit is a python-based tool 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 EC-Earth, NEMO and NMMB air quality model.

Autosubmit has been used to manage models running at supercomputers in IC3, BSC, ECMWF, EPCC, PDC and OLCF.

Autosubmit 3.0 version is now available via *PyPi* package under the terms of *GNU General Public License*.

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.

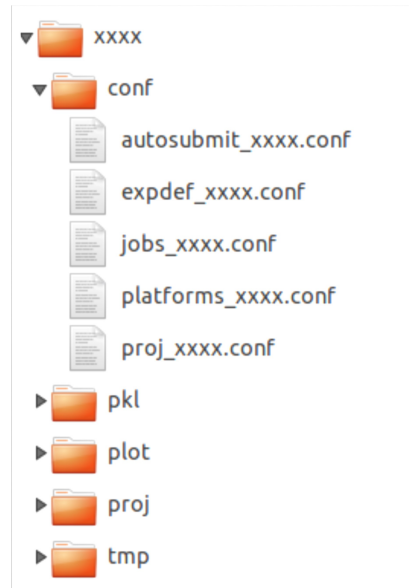


Figure 1.1: Example of an experiment directory tree.

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).

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 ap-

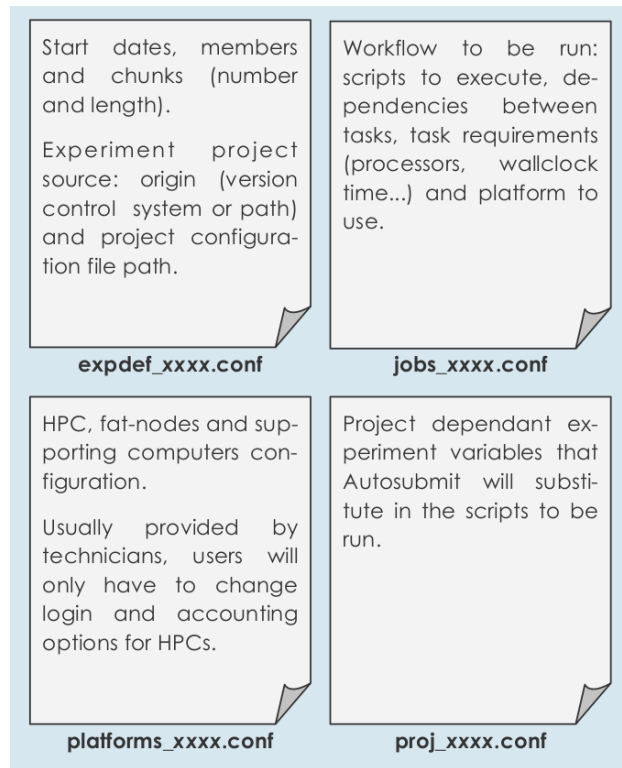


Figure 1.2: Configuration files content

pear in *%variable_name%* format. Autosubmit provides variables for *current chunk*, *start date*, *member*, *computer configuration* and more, and also will replace variables form `proj_xxxx.conf`.

To monitor the status of the experiment, the command:

```
autosubmit monitor xxxx
```

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

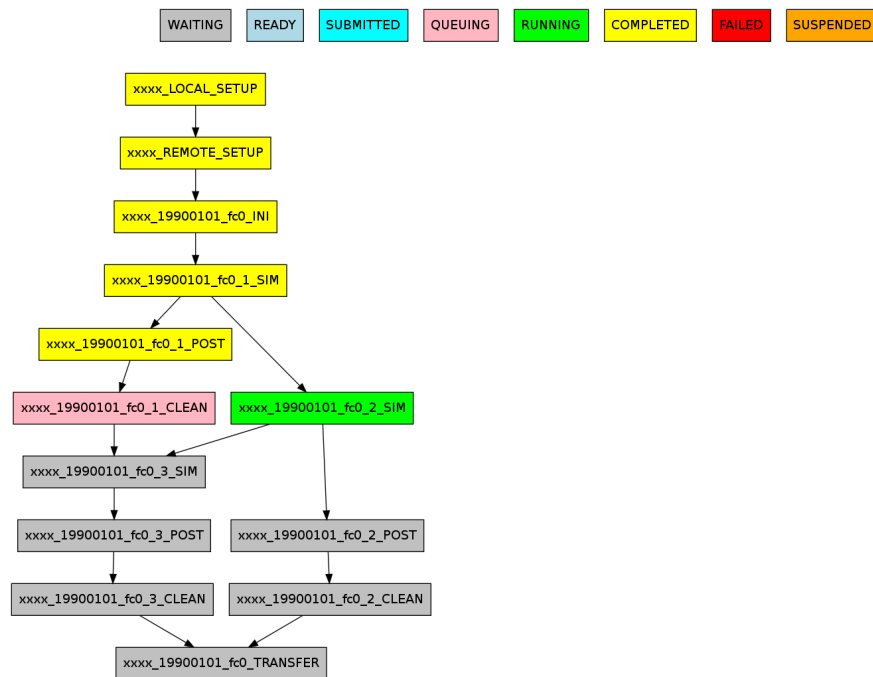


Figure 1.3: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

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.

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.

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]

```
# 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
```

[rerun]

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

[project]

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

```
# If PROJECT_TYPE is not git, no need to change
```

[git]

```
# Repository URL STRING = 'https://github.com/torvalds/linux.git'
PROJECT_ORIGIN = https://gitlab.cfu.local/cfu/auto-ecearth3.git
# Select branch or tag, STRING, default = 'master',
# help = {'master' (default), 'develop', 'v3.1b', ...}
PROJECT_BRANCH = develop
# type = STRING, default = leave empty, help = if model branch is a TAG leave empty
PROJECT_COMMIT =
```

```
# If PROJECT_TYPE is not svn, no need to change
```

[svn]

```
# type = STRING, help = 'https://svn.ec-earth.org/ecearth3'
PROJECT_URL =
# Select revision number. NUMERIC = 1778
PROJECT_REVISION =
```

```
# If PROJECT_TYPE is not local, no need to change
```

[local]

```
# type = STRING, help = /foo/bar/ecearth
PROJECT_PATH =
```

```
# 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 expedf 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 stardate, once per member or once per chunk. Options: once, da
```

```
## 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 creat
```

```
## If not specified, defaults to 1
```

```
# FREQUENCY = 3
```

```
## Specifies if a job with FREQUENCY > 1 has only to wait for all the jobs in the previous chunks on
## for the ones in the chunk it is going to execute
## If not specified, defaults to True
# WAIT = False
## Defines if job is only to be executed in reruns. If not specified, defaults to false.
# RERUN_ONLY = False
## Defines jobs needed to be rerun if this job is going to be rerun
# RERUN_DEPENDENCIES = RERUN INI LOCAL_SETUP REMOTE_SETUP TRANSFER
## Wallclock to be submitted to the HPC queue in format HH:MM
# WALLCLOCK = 00:05
## Processors number to be submitted to the HPC. If not specified, defaults to 1.
# PROCESSORS = 1
## Threads number to be submitted to the HPC. If not specified, defaults to 1.
# THREADS = 1
## 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
# RETRIALS = 4
## Some jobs can not be checked before running previous jobs. Set this option to false if that is the
# CHECK = False
```

[LOCAL_SETUP]

```
FILE = LOCAL_SETUP.sh
PLATFORM = LOCAL
```

[REMOTE_SETUP]

```
FILE = REMOTE_SETUP.sh
DEPENDENCIES = LOCAL_SETUP
WALLCLOCK = 00:05
```

[INI]

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

[SIM]

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

[POST]

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

[CLEAN]

```
FILE = CLEAN.sh
DEPENDENCIES = POST
RUNNING = chunk
```

```
WALLCLOCK = 00:05
```

[TRANSFER]

```
FILE = TRANSFER.sh
PLATFORM = LOCAL
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: 10, 11, 12)
```

```
# 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 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 any platform
```

```
## Default = 6
```

```
# TOTAL_JOBS = 6
```

[ithaca]

```
# 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
# 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 be override at job level
# Default = 0
RETRIALS = 0
```

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.

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)
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-procassing 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
NRCF = 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 =
# 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 =
# 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 =
# 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
```

INSTALLATION

3.1 How to install

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

- Pre-requisties: These packages (bash, python2, sqlite3, git-scm > 1.8.2, subversion) must be available at local host machine. These packages (argparse, dateutil, pyparsing, numpy, pydotplus, matplotlib, paramiko) must be available for python runtime.

Important: The host machine has to be able to access HPC's/Clusters via password-less ssh.

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. It can be done at host, user or local level (by default at host level). If it does not exist, create a repository for experiments: Say for example `/cfu/autosubmit`

Then follow the configre instructions after executing:

```
autosubmit configure
```

and introduce path to experiment storage and database. Folders must exit.

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.

Now you are ready to use Autosubmit !

4.1 Command list

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

4.2 How to create an experiment

To create a new experiment, just 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.

Options:

```
usage: autosubmit expid [-h] [-y COPY | -dm] -H HPC -d DESCRIPTION
```

```
-h, --help                show this help message and exit
-y COPY, --copy COPY      makes a copy of the specified experiment
-dm, --dummy              creates a new experiment with default values, usually for testing
-H HPC, --HPC HPC         specifies the HPC to use for the experiment
-d DESCRIPTION, --description DESCRIPTION
                           sets a description for the experiment to store in the database.
```

Example:

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

4.3 How to create a copy of an experiment

This option makes a copy of an existing experiment. It registers a new unique identifier and copies all configuration files in the new experiment folder:

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

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

Example:

```
autosubmit expid -H ithaca -y cxxx -d "experiment is about..."
```

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

4.4 How to create a dummy experiment

This command creates a new experiment with default values, useful for testing:

```
autosubmit expid -H HPCname -dm -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.

Example:

```
autosubmit expid -H ithaca -dm "experiment is about..."
```

4.5 How 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.

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 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.

Then, Autosubmit *create* command uses the `expdef_cxxx.conf` and generates the experiment: After editing the files you can proceed to the experiment workflow creation. Experiment workflow, which contains all the jobs and its dependencies, will be saved as a *pkl* file:

```
autosubmit create EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit create [-h] [-np] expid
```

```
    expid                experiment identifier
```

```
    -h, --help           show this help message and exit
```

```
    -np, --noplot        omit plot
```

Example:

```
autosubmit create cxxx
```

More info on pickle can be found at <http://docs.python.org/library/pickle.html>

4.6 How to check the experiment configuration

To check the configuration of the experiment, use the command:

```
autosubmit check EXPID
```

EXPID is the experiment identifier.

It checks experiment configuration and warns about any detected error or inconsistency.

Options:

```
usage: autosubmit check [-h] expid
```

```
    expid                experiment identifier
```

```
    -h, --help            show this help message and exit
```

Example:

```
autosubmit check cxxx
```

4.7 How to run the experiment

Launch Autosubmit with the command:

```
autosubmit run EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit run [-h] expid
```

```
    expid                experiment identifier
```

```
    -h, --help            show this help message and exit
```

Example:

```
autosubmit run cxxx
```

Hint: It is recommended to launch it in background and with `nohup` (continue running although the user who launched the process logs out).

Example:

```
nohup autosubmit run cxxx &
```

Important: Before launching Autosubmit check password-less ssh is feasible (*HPCName* is the hostname):

```
ssh HPCName
```

More info on password-less ssh can be found at: http://www.linuxproblem.org/art_9.html

Caution: After launching Autosubmit, one must be aware of login expiry limit and policy (if applicable for any HPC) and renew the login access accordingly (by using token/key etc) before expiry.

4.8 How to test the experiment

This method is 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.

To test the experiment, use the command:


```
autosubmit test CHUNKS EXPID
```

EXPID is the experiment identifier. *CHUNKS* is the number of chunks to run in the test.

Options:

```
usage: autosubmit test [-h] -c CHUNKS [-m MEMBER] [-s STARDATE] [-H HPC] [-b BRANCH] expid
```

```
expid                experiment identifier

-h, --help            show this help message and exit
-c CHUNKS, --chunks CHUNKS
                        chunks to run
-m MEMBER, --member MEMBER
                        member to run
-s STARDATE, --stardate STARDATE
                        stardate to run
-H HPC, --HPC HPC     HPC to run experiment on it
-b BRANCH, --branch BRANCH
                        branch from git to run (or revision from subversion)
```

Example:

```
autosubmit test -c 1 -s 19801101 -m fc0 -H ithaca -b develop cxxx
```

4.9 How to monitor the experiment

To monitor the status of the experiment, use the command:

```
autosubmit monitor EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit monitor [-h] [-o {pdf,png,ps,svg}] expid
```

```
expid                experiment identifier

-h, --help            show this help message and exit
-o {pdf,png,ps,svg}, --output {pdf,png,ps,svg}
                        type of output for generated plot
```

Example:

```
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
```

Hint: Very large plots may be a problem for some pdf and image viewers. If you are having trouble with your usual monitoring tool, try using svg output and opening it with Google Chrome with the SVG Navigator extension installed.

4.10 How to monitor job statistics

The following command could be adopted to generate the plots for visualizing the jobs statistics of the experiment at any instance:

```
autosubmit stats EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit stats [-h] [-ft] [-fp] [-o {pdf,png,ps,svg}] expid

expid                        experiment identifier

-h, --help                  show this help message and exit
-ft FILTER_TYPE, --filter_type FILTER_TYPE
                            Select the job type to filter the list of jobs
-fp FILTER_PERIOD, --filter_period FILTER_PERIOD
                            Select the period of time to filter the jobs
                            from current time to the past in number of hours back
-o {pdf,png,ps,svg}, --output {pdf,png,ps,svg}
                            type of output for generated plot
```

Example:

```
autosubmit stats cxxx
```

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

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

4.11 How to stop the experiment

You can stop Autosubmit by sending a signal to the process. To get the process identifier (PID) you can use the `ps` command on a shell interpreter/terminal.

```
ps -ef | grep autosubmit
dmanubens  22835      1   1 May04 ?           00:45:35 autosubmit run cxxx
dmanubens  25783      1   1 May04 ?           00:42:25 autosubmit run cxxx
```

To send a signal to a process you can use `kill` also on a terminal.

To stop immediately experiment `cxxx`:

```
kill -9 22835
```

Important: In case you want to restart the experiment, you must follow the [How to restart the experiment](#) procedure, explained below, in order to properly resynchronize all completed jobs.

4.12 How to restart the experiment

This procedure allows you to restart an experiment.

You must execute:

```
autosubmit recovery EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit recovery [-h] [-all] [-s] expid
```

```
    expid            experiment identifier

    -h, --help       show this help message and exit
    -all             Get all completed files to synchronize pkl
    -s, --save       Save changes to disk
```

Example:

```
autosubmit recovery cxxx -s
```

Hint: When we are satisfied with the results we can use the parameter `-s`, which will save the change to the pkl file and rename the update file.

The `-all` flag is used to synchronize all jobs of our experiment locally with the information available on the remote platform (i.e.: download the COMPLETED files we may not have). In case new files are found, the pkl will be updated.

Example:

```
autosubmit recovery cxxx -all -s
```

4.13 How to rerun a part of the experiment

This procedure allows you to create automatically a new pickle with a list of jobs of the experiment to rerun.

Using the `expdef_<expid>.conf` the `create` command will generate the rerun if the variable `RERUN` is set to `TRUE` and a `CHUNKLIST` is provided.

```
autosubmit create cxxx
```

It will read the list of chunks specified in the `CHUNKLIST` and will generate a new plot.

Note: The results are saved in the new pkl `rerun_job_list.pkl`.

Example:

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

```
[...]
```

```
[rerun]
```

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

```
[...]
```

Then you are able to start again Autosubmit for the rerun of cxxx 19601101, chunk 1, member 1:

```
nohup autosubmit run cxxx &
```

4.14 How to clean the experiment

This procedure allows you to save space after finalising an experiment. You must execute:

```
autosubmit clean EXPID
```

Options:

```
usage: autosubmit clean [-h] [-pr] [-p] [-s] expid
```

expid	experiment identifier
-h, --help	show this help message and exit
-pr, --project	clean project
-p, --plot	clean plot, only 2 last will remain
-s, --stats	clean stats, only last will remain

- The -p and -s flag are used to clean our experiment `plot` folder to save disk space. Only the two latest plots will be kept. Older plots will be removed.

Example:

```
autosubmit clean cxxx -p
```

- The -pr flag is used to clean our experiment `proj` locally in order to save space (it could be particullary big).

Caution: Bear in mind that if you have not synchronized your experiment project folder with the information available on the remote repository (i.e.: commit and push any changes we may have), or in case new files are found, the clean procedure will be failing although you specify the -pr option.

Example:

```
autosubmit clean cxxx -pr
```

A bare copy (which occupies less space on disk) will be automatically made.

Hint: That bare clone can be always reconverted in a working clone if we want to run again the experiment by using `git clone bare_clone original_clone`.

Note: In addition, every time you run this command with -pr option, it will check the commit unique identifier for local working tree existing on the `proj` directory. In case that commit identifier exists, clean will register it to the `expdef_cxxx.conf` file.

4.15 How to refresh the experiment project

To refresh the project directory of the experiment, use the command:

```
autosubmit refresh EXPID
```

EXPID is the experiment identifier.

It checks experiment configuration and copy code from original repository to project directory.

Warning: DO NOT USE THIS COMMAND IF YOU ARE NOT SURE ! Project directory will be overwritten and you may loose local changes.

Options:

```
usage: autosubmit refresh [-h] expid
```

```

    expid                experiment identifier

    -h, --help            show this help message and exit
    -mc, --model_conf     overwrite model conf file
    -jc, --jobs_conf      overwrite jobs conf file

```

Example:

```
autosubmit refresh cxxx
```

4.16 How to delete the experiment

To delete the experiment, use the command:

```
autosubmit delete EXPID
```

EXPID is the experiment identifier.

Warning: DO NOT USE THIS COMMAND IF YOU ARE NOT SURE ! It deletes the experiment from database and experiment's folder.

Options:

```
usage: autosubmit delete [-h] [-f] expid
```

```

    expid                experiment identifier

    -h, --help            show this help message and exit
    -f, --force           deletes experiment without confirmation

```

Example:

```
autosubmit delete cxxx
```

Warning: Be careful ! force option does not ask for your confirmation.

4.17 How to add a new job

To add a new job, open the <experiments_directory>/cxxx/conf/jobs_cxxx.conf file where cxxx is the experiment identifier and add this text:

```
[new_job]
```

```
FILE = <new_job_template>
```

This will create a new job named “new_job” that will be executed once at the default platform. This job will use the template located at <new_job_template> (path is relative to project folder).

This is the minimum job definition and usually is not enough. You usually will need to add some others parameters:

- **PLATFORM:** allows you to execute the job in a platform of your choice. It must be defined in the experiment’s platforms.conf file or to have the value ‘LOCAL’ that always refer to the machine running Autosubmit
- **RUNNING:** defines if jobs runs only once or once per stardate, member or chunk. Options are: once, date, member, chunk
- **DEPENDENCIES:** defines dependencies from job as a list of parents jobs separated by spaces. For example, if ‘new_job’ has to wait for “old_job” to finish, you must add the line “DEPENDENCIES = old_job”. For dependencies to jobs running in previous chunks, members or startdates, use -(DISTANCE). For example, for a job “SIM” waiting for the previous “SIM” job to finish, you have to add “DEPENDENCIES = SIM-1”

For jobs running in HPC platforms, usually you have to provide information about processors, wallclock times and more . To do this use:

- **WALLCLOCK:** wallclock time to be submitted to the HPC queue in format HH:MM
- **PROCESSORS:** processors number to be submitted to the HPC. If not specified, defaults to 1.
- **THREADS:** threads number to be submitted to the HPC. If not specified, defaults to 1.
- **TASKS:** tasks number to be submitted to the HPC. If not specified, defaults to 1.
- **QUEUE:** queue to add the job to. If not specified, uses PLATFORM default.

There are also another, less used features that you can use:

- **FREQUENCY:** specifies that a job has only to be run after X dates, members or chunk. A job will always be created for the last one. If not specified, defaults to 1
- **RERUN_ONLY:** determines if a job is only to be executed in reruns. If not specified, defaults to false.
- **RERUN_DEPENDENCIES:** defines the jobs to be rerun if this job is going to be rerun. Syntax is identical to the used in DEPENDENCIES

Example:

```
[SIM]
```

```
FILE = templates/ecearth3/ecearth3.sim
DEPENDENCIES = INI SIM-1 CLEAN-2
RUNNING = chunk
WALLCLOCK = 04:00
PROCESSORS = 1616
THREADS = 1
TASKS = 1
```

4.18 How to add a new platform

To add a new platform, open the <experiments_directory>/cxxx/conf/platforms_cxxx.conf file where cxxx is the experiment identifier and add this text:

```
[new_platform]
```

```
TYPE = <platform_type>
HOST = <host_name>
```

```
PROJECT = <project>
USER = <user>
SCRATCH = <scratch_dir>
```

This will create a platform named “new_platform”. The options specified are all mandatory:

- TYPE: queue type for the platform. Options supported are PBS, SGE, PS, LSF, ecaccess and SLURM
- HOST: hostname of the platform
- PROJECT: project for the machine scheduler.
- USER: user for the machine scheduler
- SCRATCH_DIR: path to the scratch directory of the machine

Warning: With some platform types, Autosubmit may also need the version, forcing you to add the parameter VERSION. These platforms are PBS (options: 10, 11, 12) and ecaccess (options: pbs, loadleveler)

Some platforms may require to run serial jobs in a different queue or platform. To avoid changing the job configuration, you can specify what platform or queue to use to run serial jobs assigned to this platform:

- SERIAL_PLATFORM: if specified, Autosubmit will run jobs with only one processor in the specified platform.
- SERIAL_QUEUE: if specified, Autosubmit will run jobs with only one processor in the specified queue. Autosubmit will ignore this configuration if SERIAL_PLATFORM is provided

There are some other parameters that you must need to specify:

- BUDGET: budget account for the machine scheduler. If omitted, takes the value defined in PROJECT
- ADD_PROJECT_TO_HOST = option to add project name to host. This is required for some HPCs
- QUEUE: if given, Autosubmit will add jobs to the given queue instead of platform’s default queue
- TEST_SUITE: if true, autosubmit test command can use this queue as a main queue. Defaults to false
- MAX_WAITING_JOBS: maximum number of jobs to be waiting in this platform.
- TOTAL_JOBS: maximum number of jobs to be running at the same time in this platform.

Example:

```
[platform]
TYPE = SGE
HOST = hostname
PROJECT = my_project
ADD_PROJECT_TO_HOST = true
USER = my_user
SCRATCH_DIR = /scratch
TEST_SUITE = True
```

4.19 How to archive an experiment

To archive the experiment, use the command:

```
autosubmit archive EXPID
```

EXPID is the experiment identifier.

Warning: this command calls implicitly the clean command. Check clean command documentation.

Warning: experiment will be unusable after archiving. If you want to use it, you will need to call first the unarchive command

Options:

```
usage: autosubmit archive [-h] expid
```

expid	experiment identifier
-h, --help	show this help message and exit

Example:

```
autosubmit archive cxxx
```

Hint: Archived experiment will be stored as a tar.gz file on a folder named after the year of the last COMPLETED file date. If not COMPLETED file is present, it will be stored in the folder matching the date at the time the archive command was run.

4.20 How to unarchive an experiment

To unarchive an experiment, use the command:

```
autosubmit unarchive EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit unarchive [-h] expid
```

expid	experiment identifier
-h, --help	show this help message and exit

Example:

```
autosubmit unarchive cxxx
```


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. On 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 on figure 5.1

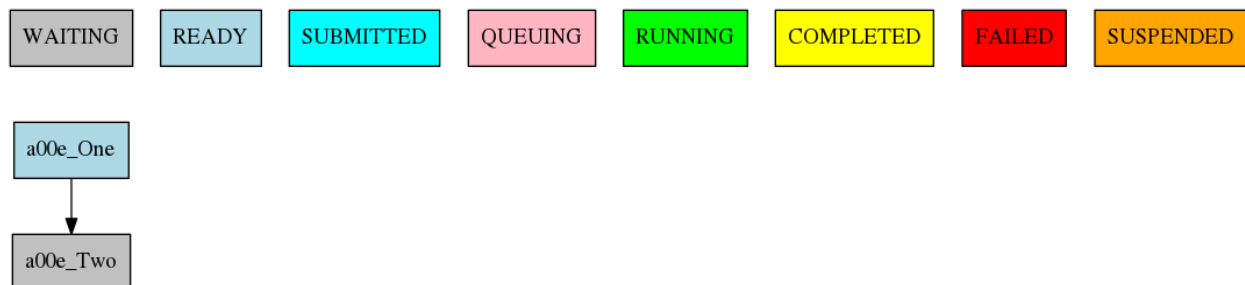


Figure 5.1: Example showing a simple workflow with two sequential jobs

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.

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 on figure 5.2 for an experiment with 2 startdates, 2 members and 2 chunks.

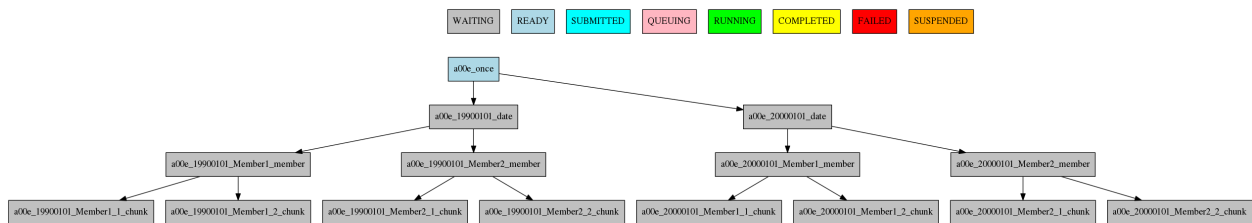


Figure 5.2: Example showing how to run jobs once per startdate, member or chunk.

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 wait a simulation job 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 on 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

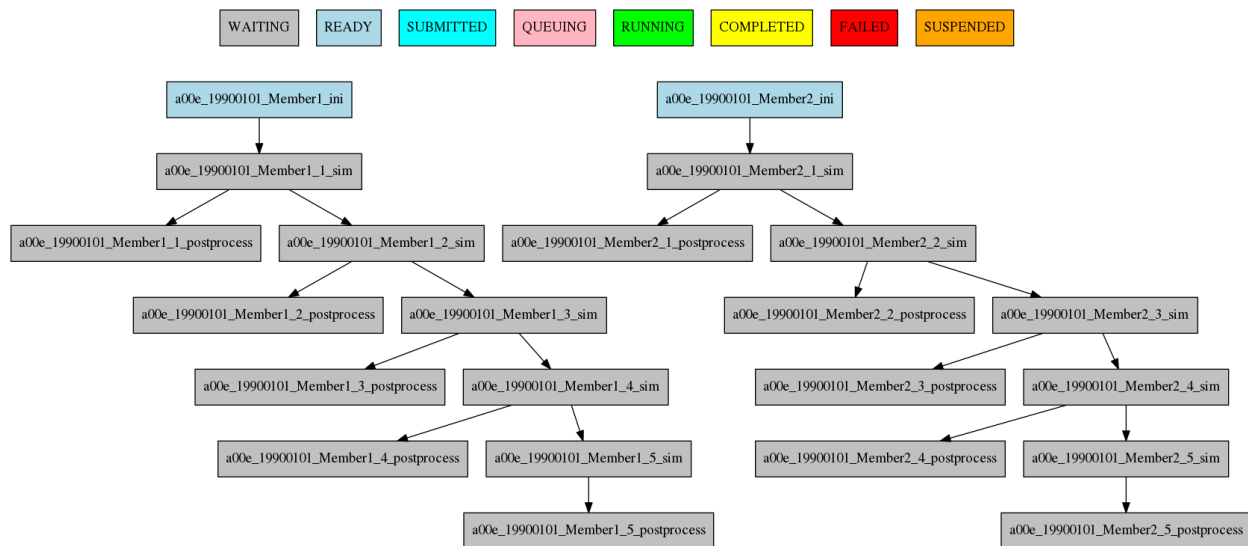


Figure 5.3: Example showing dependencies between sim jobs on different chunks.

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 on figure 5.4

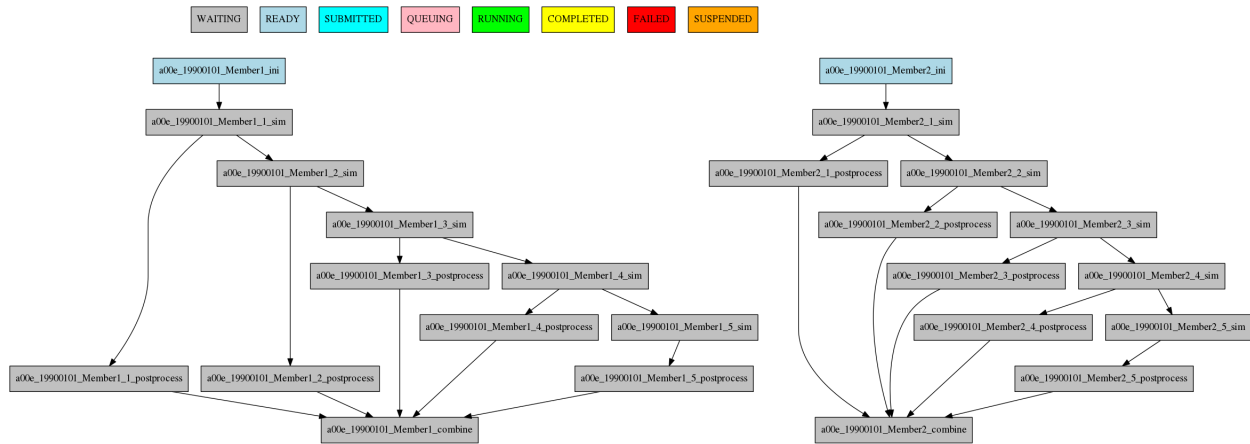


Figure 5.4: Example showing dependencies between jobs running at different levels.

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 postprocess job after various chunks have completed. This behaviour can be achieved by using the **FREQUENCY** attribute. You can specify an integer *I* on 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 on figure 5.5

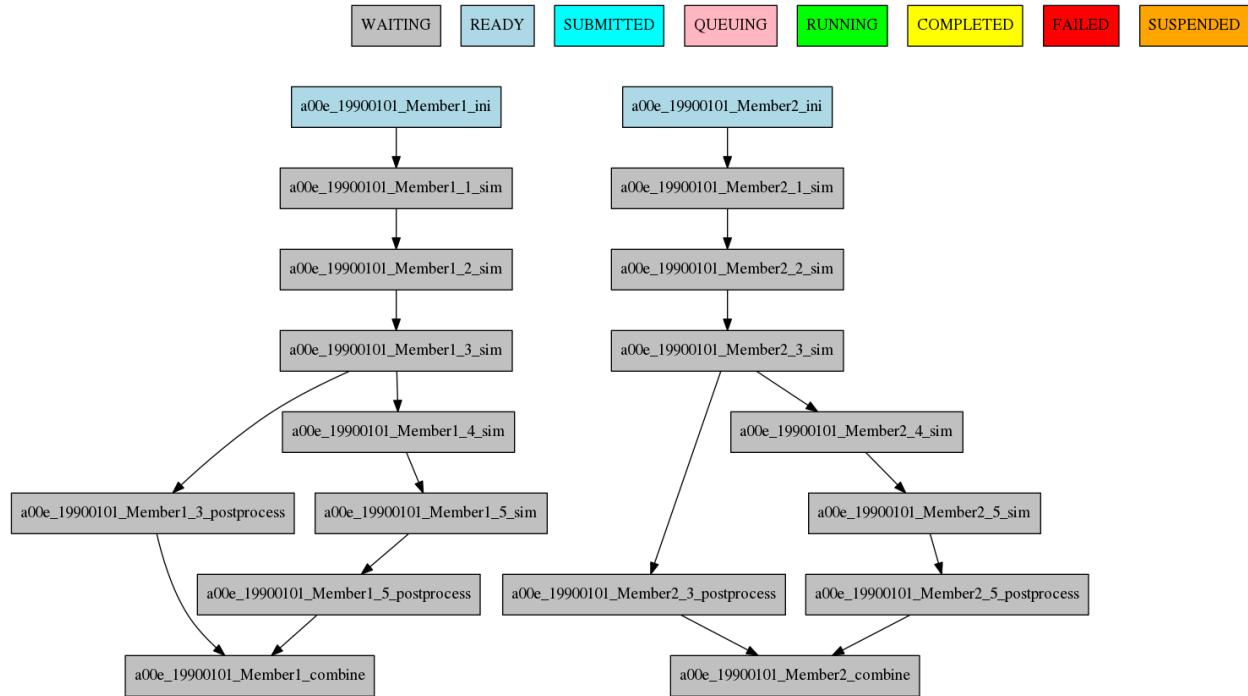


Figure 5.5: Example showing dependencies between jobs running at different frequencies.

5.5 Rerun dependencies

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

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

```
[prepare_rerun]
```

```
FILE = prepare_rerun.sh
```

```
RERUN_ONLY = true
```

```
RUNNING = member
```

```
[ini]
```

```
FILE = ini.sh
```

```
RUNNING = member
```

[sim]

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

[postprocess]

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

[combine]

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

The resulting workflow can be seen on figure 5.6 for a rerun of chunks 2 and 3 of member 2.

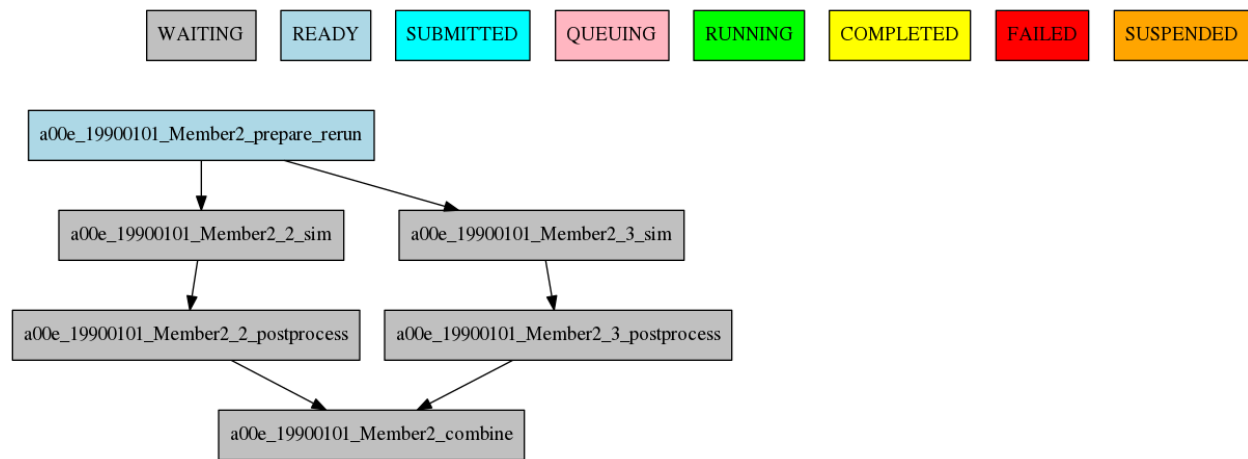


Figure 5.6: Example showing a rerun workflow for chunks 2 and 3.

TROUBLESHOOTING

6.1 How to change the job status stopping autosubmit

This procedure allows you to modify the status of your jobs.

Warning: Beware that Autosubmit must be stopped to use `setstatus`. Otherwise a running instance of Autosubmit, at some point, will overwrite any change you may have done.

You must execute:

```
autosubmit setstatus EXPID -fs STATUS_ORIGINAL -t STATUS_FINAL -s
```

EXPID is the experiment identifier. *STATUS_ORIGINAL* is the original status to filter by the list of jobs. *STATUS_FINAL* the desired target status.

Options:

```
usage: autosubmit setstatus [-h] [-s] -t
      {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN,QUEUING,RUNNING}
      (-fl LIST
      | -fc FILTER_CHUNKS
      | -fs {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
      | -ft FILTER_TYPE)
      expid

expid          experiment identifier
-h, --help      show this help message and exit
-s, --save      Save changes to disk
-t {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN},
      --status_final {READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
                  Supply the target status
-fc FILTER_CHUNKS, --filter_chunks FILTER_CHUNKS
                  Supply the list of chunks to change the status.
                  Default = "Any". LIST = "[ 19601101 [ fc0 [1 2 3 4]
                  fc1 [1] ] 19651101 [ fc0 [16-30] ] ]"
-fs {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN},
      --filter_status {Any,READY,COMPLETED,WAITING,SUSPENDED,FAILED,UNKNOWN}
                  Select the original status to filter the list of jobs
-ft FILTER_TYPE, --filter_type FILTER_TYPE
                  Select the job type to filter the list of jobs
```

Examples:

```
autosubmit setstatus cxxx -fl "cxxx_20101101_fc3_21_sim cxxx_20111101_fc4_26_sim" -t READY -s
autosubmit setstatus cxxx -fc [ 19601101 [ fc1 [1] ] ] -t READY -s
autosubmit setstatus cxxx -fs FAILED -t READY -s
autosubmit setstatus cxxx -ft TRANSFER -t SUSPENDED -s
```

This script has two mandatory arguments.

The `-t` where you must specify the target status of the jobs you want to change to:

{READY, COMPLETED, WAITING, SUSPENDED, FAILED, UNKNOWN}

The second argument has four alternatives, the `-fl`, `-fc`, `-fs` and `-ft`; with those we can apply a filter for the jobs we want to change:

- The `-fl` variable receives a list of jobnames separated by blank spaces: e.g.:

```
"cxxx_20101101_fc3_21_sim cxxx_20111101_fc4_26_sim"
```

If we supply the key word “Any”, all jobs will be changed to the target status.

- The variable `-fc` should be a list of individual chunks or ranges of chunks in the following format:

```
[ 19601101 [ fc0 [1 2 3 4] fc1 [1] ] 19651101 [ fc0 [16-30] ] ]
```

- The variable `-fs` can be one of the following status for job:

{Any, READY, COMPLETED, WAITING, SUSPENDED, FAILED, UNKNOWN}

- The variable `-ft` can be one of the defined types of job.

Hint: When we are satisfied with the results we can use the parameter `-s`, which will save the change to the `pkl` file.

6.2 How to change the job status without stopping autosubmit

This procedure allows you to modify the status of your jobs without having to stop Autosubmit.

You must create a file in `<experiments_directory>/<expid>/pkl/` named:

`updated_list_<expid>.txt`

Format:

This file should have two columns: the first one has to be the `job_name` and the second one the status.

Options:

READY, COMPLETED, WAITING, SUSPENDED, FAILED, UNKNOWN

Example:

```
vi updated_list_cxxx.txt
```

```
cxxx_20101101_fc3_21_sim    READY
cxxx_20111101_fc4_26_sim    READY
```

If Autosubmit finds the above file, it will process it. You can check that the processing was OK at a given date and time, if you see that the file name has changed to:

update_list_<expid>_<date>_<time>.txt

Note: A running instance of Autosubmit will check the existence of above file after checking already submitted jobs. It may take some time, depending on the setting `SAFETYSLEEPTIME`.

Warning: Keep in mind that autosubmit reads the file automatically so it is suggested to create the file in another location like `/tmp` or `/var/tmp` and then copy/move it to the `pkl` folder. Alternatively you can create the file with a different name and rename it when you have finished.

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.

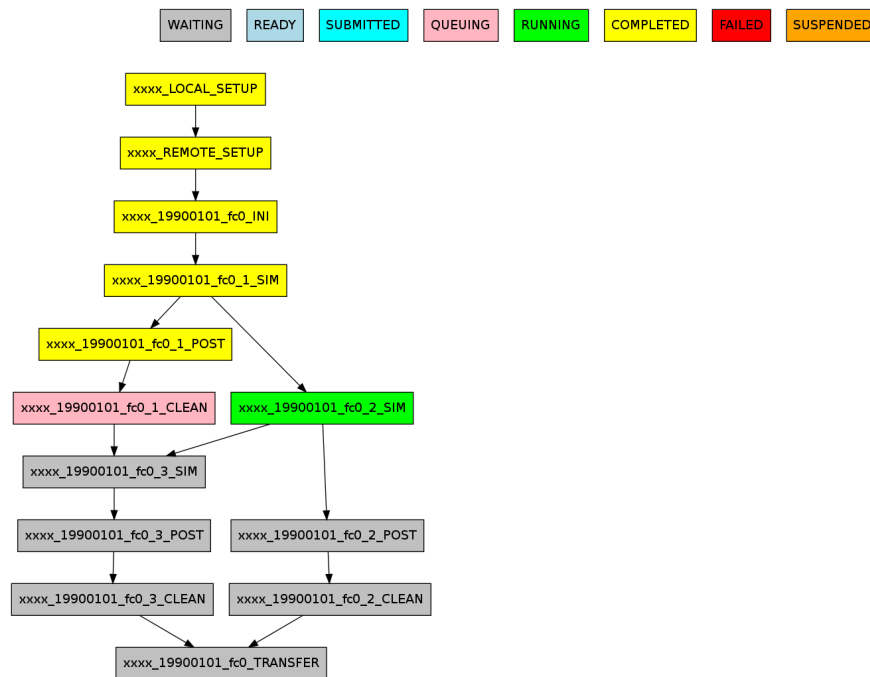


Figure 7.1: Example of monitoring plot for EC-Earth run with Autosubmit for 1 start date, 1 member and 3 chunks.

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.

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

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

VARIABLES REFERENCE

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

8.1 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.
- **DAY_BEFORE**: day before the startdate
- **Chunk_End_IN_DAYS**: chunk's length in days
- **Chunk_START_DATE**: chunk's start date
- **Chunk_START_YEAR**: chunk's start year
- **Chunk_START_MONTH**: chunk's start month
- **Chunk_START_DAY**: chunk's start day
- **Chunk_START_HOUR**: chunk's start hout
- **Chunk_END_DATE**: chunk's end date
- **Chunk_END_YEAR**: chunk's end year
- **Chunk_END_MONTH**: chunk's end month
- **Chunk_END_DAY**: chunk's end day
- **Chunk_END_HOUR**: chunk's end hour
- **PREV**: days since startdate at the chunk's start
- **Chunk_FIRST**: True if the current chunk is the first, false otherwise.
- **Chunk_LAST**: True if the current chunk is the last, false otherwise.
- **NUMPROC**: Number of processors that the job will use.

- **NUMTHREADS**: Number of threads that the job will use.
- **NUMTASK**: Number of tasks that the job will use.
- **WALLCLOCK**: Number of processors that the job will use.

8.2 Platform variables

This variables are relative to the platforms 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}_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

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

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

8.3 Project variables

- **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

MODULE DOCUMENTATION

9.1 autosubmit

9.2 autosubmit.config

9.2.1 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.

9.2.2 autosubmit.config.config_common

class `autosubmit.config.config_common.AutosubmitConfig` (*expid*)

Class to handle experiment configuration coming from file or database

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

check_conf_files()

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

static check_exists (*parser*, *section*, *option*)

Checks if an option exists in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check

Returns True if option exists, False otherwise

Return type bool

static check_is_boolean (*parser*, *section*, *option*, *must_exist*)

Checks if an option is a boolean value in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is boolean, False otherwise

Return type bool

static check_is_choice (*parser, section, option, must_exist, choices*)

Checks if an option is a valid choice in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **choices** (*list*) – valid choices

Returns True if option value is a valid choice, False otherwise

Return type bool

static check_is_int (*parser, section, option, must_exist*)

Checks if an option is an integer value in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist

Returns True if option value is integer, False otherwise

Return type bool

static check_json (*key, value*)

Checks if value is a valid json

Parameters

- **key** (*str*) – key to check
- **value** (*str*) – value

Returns True if value is a valid json, False otherwise

Return type bool

check_proj ()

Checks project config file

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

Return type bool

static check_regex (*parser, section, option, must_exist, regex*)

Checks if an option complies with a regular expression in given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to check
- **must_exist** (*bool*) – if True, option must exist
- **regex** (*str*) – regular expression to check

Returns True if option complies with regex, False otherwise

Return type bool

experiment_file

Returns experiment's config file name

static get_bool_option (*parser, section, option, default*)

Gets a boolean option from given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*bool*) – value to be returned if option is not present

Returns option value

Return type bool

get_chunk_list ()

Returns chunk list from experiment's config file

Returns experiment's chunks

Return type list

get_chunk_size_unit ()

Unit for the chunk length

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

Return type str

get_date_list ()

Returns startdates list from experiment's config file

Returns experiment's startdates

Return type list

get_exp_id ()

Returns experiment identifier read from experiment's config file

Returns experiment identifier

Return type str

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_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_local_project_path()

Gets path to origin for local project

Returns path to local project

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_member_list()

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_num_chunks()

Returns number of chunks to run for each member

Returns number of chunks

Return type int

static get_option(parser, section, option, default)

Gets an option from given parser

Parameters

- **parser** (*SafeConfigParser*) – parser to use
- **section** (*str*) – section that contains the option
- **option** (*str*) – option to get
- **default** (*object*) – value to be returned if option is not present

Returns option value

Return type str

static get_parser (*file_path*)

Gets parser for given file

Parameters **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_rerun ()

Returns startdates list from experiment's config file

Returns rerun value

Return type list

get_retrials ()

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

Returns safety sleep time

Return type int

get_safetysleeptime()

Returns safety sleep time from autosubmit's config file

Returns safety sleep time

Return type int

get_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_tasks(*section*)

Gets tasks needed for the given job type :param section: job type :type section: str :return: tasks needed :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_wallclock(*section*)

Gets wallclock for the given job type :param section: job type :type section: str :return: wallclock time :rtype: str

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_project_parameters()

Loads parameters from model config file

Returns 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

static print_parameters (*title, parameters*)

Prints the parameters table in a tabular mode

Parameters

- **title** (*str*) – table's title
- **parameters** – parameters to print

Type list

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_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

9.2.3 autosubmit.config.log

class autosubmit.config.log.Log

Static class to manage the log for the application. Messages will be sent to console and to file if it is configured. Levels can be set for each output independently. These levels are (from lower to higher priority):

- **EVERYTHING** : this level is just defined to show every output
- **DEBUG**
- **INFO**
- **RESULT**
- **USER_WARNING**

- WARNING
- ERROR
- CRITICAL
- NO_LOG : this level is just defined to remove every output

static critical (*msg, *args*)

Sends critical errors to the log. It will be shown in red in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using format() method on str)

static debug (*msg, *args*)

Sends debug information to the log

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using format() method on str)

static error (*msg, *args*)

Sends errors to the log. It will be shown in red in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using format() method on str)

static info (*msg, *args*)

Sends information to the log

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using format() method on str)

static result (*msg, *args*)

Sends results information to the log. It will be shown in green in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using format() method on str)

static set_console_level (*level*)

Sets log level for logging to console. Every output of level equal or higher to parameter level will be printed on console

Parameters **level** – new level for console

Returns None

static set_file (*file_path*)

Configure the file to store the log. If another file was specified earlier, new messages will only go to the new file.

Parameters **file_path** (*str*) – file to store the log

static `set_file_level` (*level*)

Sets log level for logging to file. Every output of level equal or higher to parameter level will be added to log file

Parameters `level` – new level for log file

static `user_warning` (*msg*, **args*)

Sends warnings for the user to the log. It will be shown in yellow in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using `format()` method on `str`)

static `warning` (*msg*, **args*)

Sends program warnings to the log. It will be shown in yellow in the console.

Parameters

- **msg** – message to show
- **args** – arguments for message formatting (it will be done using `format()` method on `str`)

class `autosubmit.config.log.LogFormatter` (*to_file=False*)

Class to format log output.

Parameters `to_file` (*bool*) – If True, creates a LogFormatter for files; if False, for console

format (*record*)

Format log output, adding labels if needed for log level. If logging to console, also manages font color. If logging to file adds timestamp

Parameters `record` (*LogRecord*) – log record to format

Returns formatted record

Return type `str`

9.3 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.base36decode` (*number*)

Converts a base36 string to a positive integer

Parameters `number` (*str*) – base36 string to convert

Returns number's integer value

Return type `int`

`autosubmit.database.db_common.base36encode` (*number*, *alphabet='0123456789abcdefghijklmnopqrstuvwxyz'*)

Convert positive integer to a base36 string.

Parameters

- **number** (*int*) – number to convert
- **alphabet** (*str*) – set of characters to use

Returns number's base36 string value

Return type str

`autosubmit.database.db_common.check_db()`

Checks if database file exist

Returns None if exists, terminates program if not

`autosubmit.database.db_common.check_experiment_exists(name, error_on_inexistence=True)`

Checks if exist an experiment with the given name.

Parameters

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

Returns If experiment exists returns true, if not returns false

Return type bool

`autosubmit.database.db_common.check_name(name)`

Checks if it is a valid experiment identifier

Parameters **name** (*str*) – experiment identifier to check

Returns name if is valid, terminates program otherwise

Return type str

`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.copy_experiment(name, description, version, test=False)`

Creates a new experiment by copying an existing experiment

Parameters

- **test** (*bool*) – specifies if it is a test experiment
- **version** (*str*) – experiment's associated autosubmit version
- **name** (*str*) – identifier of experiment to copy
- **description** (*str*) – experiment's description

Returns experiment id for the new experiment

Return type str

`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(name)`

Removes experiment from database

Parameters **name** (*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

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)`

Gets last experiment identifier used

Parameters `test` (*bool*) – flag for test experiments

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

Return type str

`autosubmit.database.db_common.new_experiment(description, version, test=False)`

Stores a new experiment on the database and generates its identifier

Parameters

- **version** (*str*) – autosubmit version associated to the experiment
- **test** (*bool*) – flag for test experiments
- **description** (*str*) – experiment's description

Returns experiment id for the new experiment

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

9.4 autosubmit.date

In this python script there are tools to manipulate the dates and make mathematical operations between them.

`autosubmit.date.chunk_date_lib.add_days(date, number_of_days, cal)`

Adds days to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_days** (*int*) – number of days to add
- **cal** (*str*) – calendar to use

Returns base date plus added days

Return type date

`autosubmit.date.chunk_date_lib.add_hours(date, number_of_hours, cal)`

Adds hours to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_hours** (*int*) – number of hours to add
- **cal** (*str*) – calendar to use

Returns base date plus added hours

Return type *datetime*

`autosubmit.date.chunk_date_lib.add_months (date, number_of_months, cal)`

Adds months to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_months** (*int*) – number of months to add
- **cal** (*str*) – calendar to use

Returns base date plus added months

Return type *date*

`autosubmit.date.chunk_date_lib.add_time (date, total_size, chunk_unit, cal)`

Adds given time to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **total_size** (*int*) – time to add
- **chunk_unit** (*str*) – unit of time to add
- **cal** (*str*) – calendar to use

Returns result of adding time to base date

Return type *datetime.datetime*

`autosubmit.date.chunk_date_lib.add_years (date, number_of_years)`

Adds years to a date

Parameters

- **date** (*datetime.datetime*) – base date
- **number_of_years** (*int*) – number of years to add

Returns base date plus added years

Return type *date*

`autosubmit.date.chunk_date_lib.chunk_end_date (start_date, chunk_length, chunk_unit, cal)`

Gets chunk interval end date

Parameters

- **start_date** (*datetime.datetime*) – chunk's start date
- **chunk_length** (*int*) – length of the chunks
- **chunk_unit** (*str*) – chunk length unit
- **cal** (*str*) – calendar to use

Returns chunk's end date

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**chunk_start_date**(date, chunk, chunk_length, chunk_unit, cal)

Gets chunk's interval start date

Parameters

- **date** (datetime.datetime) – start date for member
- **chunk** (int) – number of chunk
- **chunk_length** (int) – length of chunks
- **chunk_unit** (str) – chunk length unit
- **cal** (str) – calendar to use

Returns chunk's start date

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**date2str**(date, date_format='')

Converts a datetime object to a str

Parameters **date_format** – specifies format for date time convcersion. It can be H to show hours,

M to show hour and minute. Other values will return only the date. :type date_format: str :param date: date to convert :type date: datetime.datetime :rtype: str

autosubmit.date.chunk_date_lib.**parse_date**(string_date)

Parses a string into a datetime object

Parameters **string_date** (str) – string to parse

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**previous_day**(date, cal)

Gets previous day

Parameters

- **date** (datetime.datetime) – base date
- **cal** (str) – calendar to use

Returns base date minus one day

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**sub_days**(date, number_of_days, cal)

Substract days to a date

Parameters

- **date** (datetime.datetime) – base date
- **number_of_days** (int) – number of days to subtract
- **cal** (str) – calendar to use

Returns base date minus subtracted days

Return type datetime.datetime

autosubmit.date.chunk_date_lib.**subs_dates**(start_date, end_date, cal)

Gets days between start_date and end_date

Parameters

- **start_date** (*datetime.datetime*) – interval’s start date
- **end_date** (*datetime.datetime*) – interval’s end date
- **cal** (*str*) – calendar to use

Returns interval length in days

Return type int

9.5 autosubmit.git

class `autosubmit.git.git_common.AutosubmitGit` (*expid*)

Class to handle experiment git repository

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

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*)

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

Returns True if clone was succesfull, False otherwise

9.6 autosubmit.job

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

class `autosubmit.job.job.Job` (*name, jobid, 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 (**new_parent*)

Add parents for the job. It also adds current job as a child for all the new parents

Parameters **new_parent** (**Job*) – job’s parents to add

ancestors

Returns all job’s ancestors

Returns job ancestors

Return type set

check_completion (*default_status=-1*)

Check the presence of *COMPLETED* file and touch a Checked or failed 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*)

Checks if script is well formed

Parameters

- **parameters** (*dict*) – script parameters
- **as_conf** (*AutosubmitConfig*) – configuration file

Returns true if not problem has been detected, false otherwise

Return type bool

check_start_time ()

Returns job's start time

Returns start time

Return type str

check_started_after (*date_limit*)

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

children

Returns a list containing all children of the job

Returns child jobs

Return type set

compare_by_id (*other*)

Compare jobs by ID

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

Returns comparison result

Return type bool

compare_by_name (*other*)

Compare jobs by name

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

Returns comparison result

Return type bool

compare_by_status (*other*)

Compare jobs by status value

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

Returns comparison result

Return type bool

create_script (*as_conf*)

Creates script file to be run for the job

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

Returns script's filename

Return type str

delete_child (*child*)

Removes a child from the job

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

delete_parent (*parent*)

Remove a parent from the job

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

get_platform ()

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

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

get_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

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

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
Return parent jobs list
Returns parent jobs
Return type set

print_job()
Prints debug information about the job

print_parameters()
Print sjob parameters in log

remove_dependencies()
Checks if job is completed and then remove dependencies for childs

set_platform(value)
Sets the HPC platforms to be used by the job.
Parameters *value* (*HPCPlatform*) – platforms to set

set_queue(value)
Sets the queue to be used by the job.
Parameters *value* (*HPCPlatform*) – queue to set

short_name
Job short name
Returns short name
Return type str

update_content(project_dir)
Create the script content to be run for the job
Parameters *project_dir* (*str*) – project directory
Returns script code
Return type str

update_parameters(as_conf, parameters)
Refresh parameters value
Parameters

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

update_status (*new_status*)

Updates job status, checking COMPLETED file if needed

Parameters *new_status* – job status retrieved from the platform

Type Status

write_end_time (*completed*)

Writes ends date and time to TOTAL_STATS file :param completed: True if job was completed succesfully, False otherwise :type completed: bool

write_start_time ()

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

write_submit_time ()

Writes submit date and time to TOTAL_STATS file

class autosubmit.job.job_common.**StatisticsSnippet**

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_list.**DicJobs** (*joblist*, *parser*, *date_list*, *member_list*, *chunk_list*, *date_format*, *default_retrials*)

Class to create jobs from conf file and to find jobs by stardate, member and chunk

Parameters

- **joblist** (*JobList*) – joblist to use
- **parser** (*SafeConfigParser*) – jobs conf file parser
- **date_list** (*list*) – startdates
- **member_list** (*list*) – member
- **chunk_list** (*list*) – chunks
- **date_format** (*str*) – option to formate dates
- **default_retrials** (*int*) – default retrials for ech job

get_jobs (*section*, *date=None*, *member=None*, *chunk=None*)

Return all the jobs matching section, date, member and chunk provided. If any parameter is none, returns all the jobs without checking that parameter value. If a job has one parameter to None, is returned if all the others match parameters passed

Parameters

- **section** (*str*) – section to return
- **date** (*str*) – stardate to return
- **member** (*str*) – member to return
- **chunk** (*int*) – chunk to return

Returns jobs matching parameters passed

Return type list

get_option (*section*, *option*, *default*)

Returns value for a given option

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option to return
- **default** (*object*) – value to return if not defined in configuration file

read_section (*section, priority*)

Read a section from jobs conf and creates all jobs for it

Parameters

- **section** (*str*) – section to read
- **priority** (*int*) – priority for the jobs

class `autosubmit.job.job_list.JobList` (*expid*)

Class to manage the list of jobs to be run by autosubmit

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

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

create (*date_list, member_list, num_chunks, parameters, date_format, default_retrials*)

Creates all jobs needed for the current workflow

Parameters

- **date_list** (*list*) – start dates
- **member_list** (*list*) – members
- **num_chunks** (*int*) – number of chunks to run
- **parameters** (*dict*) – parameters for the jobs
- **date_format** (*str*) – option to format dates
- **default_retrials** (*int*) – default retrials for each job

expid

Returns experiment identifier

Returns experiment's identifier

Return type str

get_active (*platform=None*)

Returns a list of active jobs (In platforms, Ready)

Parameters **platform** (*HPCCPlatform*) – job platform

Returns active jobs

Return type list

get_completed (*platform=None*)

Returns a list of completed jobs

Parameters **platform** (*HPCCPlatform*) – job platform

Returns completed jobs

Return type list

get_failed (*platform=None*)

Returns a list of failed jobs

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

Returns failed jobs

Return type list

get_finished (*platform=None*)

Returns a list of jobs finished (Completed, Failed)

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

Returns finished jobs

Return type list

get_in_queue (*platform=None*)

Returns a list of jobs in the platforms (Submitted, Running, Queuing)

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_not_in_queue (*platform=None*)

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_queuing (*platform=None*)

Returns a list of jobs queuing

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

Returns queued jobs

Return type list

get_ready (*platform=None*)

Returns a list of ready jobs

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

Returns ready jobs

Return type list

get_running (*platform=None*)

Returns a list of jobs running

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

Returns running jobs

Return type list

get_submitted (*platform=None*)

Returns a list of submitted jobs

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

Returns submitted jobs

Return type list

get_unknown (*platform=None*)

Returns a list of jobs on unknown state

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

Returns unknown state jobs

Return type list

get_waiting (*platform=None*)

Returns a list of jobs waiting

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

Returns waiting jobs

Return type list

load ()

Recreates an stored joblist from the pickle file

Returns loaded joblist 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

remove_rerun_only_jobs ()

Removes all jobs to be runned only in reruns

rerun (*chunk_list*)

Updates joblist to rerun the jobs specified by chunk_list

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

save ()

Stores joblist as a pickle file

Returns loaded joblist object

Return type JobList

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 joblist 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()

When we have created the joblist, every type of job is created. Update genealogy remove jobs that have no templates

update_list (*as_conf*)

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

update_shortened_names()

In some cases the scheduler only can operate with names shorter than 15 characters. Update the job list replacing job names by the corresponding shortened job name

9.7 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

static create_bar_diagram (*expid, joblist, output_file, period_ini=None, period_fi=None*)

Function to plot statistics

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot
- **output_file** (*str*) – path to create file
- **period_ini** (*datetime*) – initial datetime of filtered period
- **period_fi** (*datetime*) – final datetime of filtered period

create_tree_list (*expid, joblist*)

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, output_format='pdf', show=False*)

Plots graph 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
- **show** (*bool*) – if true, will open the new plot with the default viewer

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

Plots stats for joblist and stores it in a file

Parameters

- **expid** (*str*) – experiment's identifier
- **joblist** (*JobList*) – joblist to plot
- **output_format** (*str (png, pdf, ps)*) – file format for plot
- **period_ini** (*datetime*) – initial datetime of filtered period

- **period_fi** (*datetime*) – final datetime of filtered period
- **show** (*bool*) – if true, will open the new plot with the default viewer

9.8 autosubmit.platform

a

`autosubmit.config.basicConfig`, 43
`autosubmit.config.config_common`, 43
`autosubmit.config.log`, 49
`autosubmit.database.db_common`, 51
`autosubmit.date.chunk_date_lib`, 53
`autosubmit.git.git_common`, 56
`autosubmit.job.job`, 56
`autosubmit.job.job_common`, 60
`autosubmit.job.job_list`, 60
`autosubmit.monitor.monitor`, 64

A

[add_days\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [53](#)
[add_hours\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [53](#)
[add_months\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [54](#)
[add_parent\(\)](#) (`autosubmit.job.job.Job` method), [56](#)
[add_time\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [54](#)
[add_years\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [54](#)
[ancestors](#) (`autosubmit.job.job.Job` attribute), [56](#)
[autosubmit.config.basicConfig](#) (module), [43](#)
[autosubmit.config.config_common](#) (module), [43](#)
[autosubmit.config.log](#) (module), [49](#)
[autosubmit.database.db_common](#) (module), [51](#)
[autosubmit.date.chunk_date_lib](#) (module), [53](#)
[autosubmit.git.git_common](#) (module), [56](#)
[autosubmit.job.job](#) (module), [56](#)
[autosubmit.job.job_common](#) (module), [60](#)
[autosubmit.job.job_list](#) (module), [60](#)
[autosubmit.monitor.monitor](#) (module), [64](#)
[AutosubmitConfig](#) (class in `autosubmit.config.config_common`), [43](#)
[AutosubmitGit](#) (class in `autosubmit.git.git_common`), [56](#)

B

[base36decode\(\)](#) (in module `autosubmit.database.db_common`), [51](#)
[base36encode\(\)](#) (in module `autosubmit.database.db_common`), [51](#)
[BasicConfig](#) (class in `autosubmit.config.basicConfig`), [43](#)

C

[check_completion\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_conf_files\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` method), [43](#)
[check_db\(\)](#) (in module `autosubmit.database.db_common`), [52](#)
[check_end_time\(\)](#) (`autosubmit.job.job.Job` method), [57](#)

[check_exists\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [43](#)
[check_experiment_exists\(\)](#) (in module `autosubmit.database.db_common`), [52](#)
[check_is_boolean\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [43](#)
[check_is_choice\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [44](#)
[check_is_int\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [44](#)
[check_json\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [44](#)
[check_name\(\)](#) (in module `autosubmit.database.db_common`), [52](#)
[check_proj\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` method), [44](#)
[check_regex\(\)](#) (`autosubmit.config.config_common.AutosubmitConfig` static method), [44](#)
[check_retrials_end_time\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_retrials_start_time\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_retrials_submit_time\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_running_after\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_script\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_scripts\(\)](#) (`autosubmit.job.job_list.JobList` method), [61](#)
[check_start_time\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[check_started_after\(\)](#) (`autosubmit.job.job.Job` method), [57](#)
[children](#) (`autosubmit.job.job.Job` attribute), [57](#)
[chunk_end_date\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [54](#)
[chunk_start_date\(\)](#) (in module `autosubmit.date.chunk_date_lib`), [55](#)

[clean_git\(\)](#) (autosubmit.git.git_common.AutosubmitGit static method), [56](#)
[clean_plot\(\)](#) (autosubmit.monitor.monitor.Monitor static method), [64](#)
[clean_stats\(\)](#) (autosubmit.monitor.monitor.Monitor static method), [64](#)
[clone_repository\(\)](#) (autosubmit.git.git_common.AutosubmitGit static method), [56](#)
[close_conn\(\)](#) (in module autosubmit.database.db_common), [52](#)
[color_status\(\)](#) (autosubmit.monitor.monitor.Monitor static method), [65](#)
[compare_by_id\(\)](#) (autosubmit.job.job.Job method), [58](#)
[compare_by_name\(\)](#) (autosubmit.job.job.Job method), [58](#)
[compare_by_status\(\)](#) (autosubmit.job.job.Job method), [58](#)
[copy_experiment\(\)](#) (in module autosubmit.database.db_common), [52](#)
[create\(\)](#) (autosubmit.job.job_list.JobList method), [61](#)
[create_bar_diagram\(\)](#) (autosubmit.monitor.monitor.Monitor static method), [65](#)
[create_db\(\)](#) (in module autosubmit.database.db_common), [52](#)
[create_script\(\)](#) (autosubmit.job.job.Job method), [58](#)
[create_tree_list\(\)](#) (autosubmit.monitor.monitor.Monitor method), [65](#)
[critical\(\)](#) (autosubmit.config.log.Log static method), [50](#)

D

[date2str\(\)](#) (in module autosubmit.date.chunk_date_lib), [55](#)
[DbException](#), [51](#)
[debug\(\)](#) (autosubmit.config.log.Log static method), [50](#)
[delete_child\(\)](#) (autosubmit.job.job.Job method), [58](#)
[delete_experiment\(\)](#) (in module autosubmit.database.db_common), [52](#)
[delete_parent\(\)](#) (autosubmit.job.job.Job method), [58](#)
[DicJobs](#) (class in autosubmit.job.job_list), [60](#)

E

[error\(\)](#) (autosubmit.config.log.Log static method), [50](#)
[experiment_file](#) (autosubmit.config.config_common.AutosubmitConfig attribute), [45](#)
[expid](#) (autosubmit.job.job_list.JobList attribute), [61](#)

F

[format\(\)](#) (autosubmit.config.log.LogFormatter method), [51](#)

G

[generate_output\(\)](#) (autosubmit.monitor.monitor.Monitor method), [65](#)
[generate_output_stats\(\)](#) (autosubmit.monitor.monitor.Monitor method), [65](#)
[get_active\(\)](#) (autosubmit.job.job_list.JobList method), [61](#)
[get_autosubmit_version\(\)](#) (in module autosubmit.database.db_common), [53](#)
[get_bool_option\(\)](#) (autosubmit.config.config_common.AutosubmitConfig static method), [45](#)
[get_chunk_list\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [45](#)
[get_chunk_size_unit\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [45](#)
[get_completed\(\)](#) (autosubmit.job.job_list.JobList method), [61](#)
[get_date_list\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [45](#)
[get_expid\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [45](#)
[get_failed\(\)](#) (autosubmit.job.job_list.JobList method), [61](#)
[get_file_jobs_conf\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [45](#)
[get_file_project_conf\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_finished\(\)](#) (autosubmit.job.job_list.JobList method), [62](#)
[get_git_project_branch\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_git_project_commit\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_git_project_origin\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_in_queue\(\)](#) (autosubmit.job.job_list.JobList method), [62](#)
[get_job_by_name\(\)](#) (autosubmit.job.job_list.JobList method), [62](#)
[get_job_list\(\)](#) (autosubmit.job.job_list.JobList method), [62](#)
[get_jobs\(\)](#) (autosubmit.job.job_list.DicJobs method), [60](#)
[get_local_project_path\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_max_waiting_jobs\(\)](#) (autosubmit.config.config_common.AutosubmitConfig method), [46](#)
[get_member_list\(\)](#) (autosubmit.config.config_common.AutosubmitConfig

- method), 46
- get_memory() (autosubmit.config.config_common.AutosubmitConfig method), 46
- get_not_in_queue() (autosubmit.job.job_list.JobList method), 62
- get_num_chunks() (autosubmit.config.config_common.AutosubmitConfig method), 46
- get_option() (autosubmit.config.config_common.AutosubmitConfig static method), 46
- get_option() (autosubmit.job.job_list.DicJobs method), 60
- get_parser() (autosubmit.config.config_common.AutosubmitConfig static method), 47
- get_platform() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_platform() (autosubmit.job.job.Job method), 58
- get_processors() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_project_destination() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_project_dir() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_project_type() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_queue() (autosubmit.job.job.Job method), 58
- get_queuing() (autosubmit.job.job_list.JobList method), 62
- get_ready() (autosubmit.job.job_list.JobList method), 62
- get_rerun() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_retrials() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_running() (autosubmit.job.job_list.JobList method), 62
- get_safetysleeptime() (autosubmit.config.config_common.AutosubmitConfig method), 47
- get_submitted() (autosubmit.job.job_list.JobList method), 63
- get_svn_project_revision() (autosubmit.config.config_common.AutosubmitConfig method), 48
- get_svn_project_url() (autosubmit.config.config_common.AutosubmitConfig method), 48
- get_tasks() (autosubmit.config.config_common.AutosubmitConfig method), 48
- get_threads() (autosubmit.config.config_common.AutosubmitConfig method), 48
- get_total_jobs() (autosubmit.config.config_common.AutosubmitConfig method), 48
- get_unknown() (autosubmit.job.job_list.JobList method), 63
- get_waiting() (autosubmit.job.job_list.JobList method), 63
- get_wallclock() (autosubmit.config.config_common.AutosubmitConfig method), 48
- ## H
- has_children() (autosubmit.job.job.Job method), 58
- has_parents() (autosubmit.job.job.Job method), 58
- ## I
- inc_fail_count() (autosubmit.job.job.Job method), 59
- info() (autosubmit.config.log.Log static method), 50
- ## J
- Job (class in autosubmit.job.job), 56
- JobList (class in autosubmit.job.job_list), 61
- jobs_file (autosubmit.config.config_common.AutosubmitConfig attribute), 48
- ## L
- last_name_used() (in module autosubmit.database.db_common), 53
- load() (autosubmit.job.job_list.JobList method), 63
- load_file() (autosubmit.job.job_list.JobList static method), 63
- load_parameters() (autosubmit.config.config_common.AutosubmitConfig method), 48
- load_project_parameters() (autosubmit.config.config_common.AutosubmitConfig method), 48
- Log (class in autosubmit.config.log), 49
- log_job() (autosubmit.job.job.Job method), 59
- LogFormatter (class in autosubmit.config.log), 51
- long_name (autosubmit.job.job.Job attribute), 59
- ## M
- Monitor (class in autosubmit.monitor.monitor), 64
- ## N
- new_experiment() (in module autosubmit.database.db_common), 53

O

`open_conn()` (in module `autosubmit.database.db_common`), 53

P

`parameters` (`autosubmit.job.job_list.JobList` attribute), 63

`parents` (`autosubmit.job.job.Job` attribute), 59

`parse_date()` (in module `autosubmit.date.chunk_date_lib`), 55

`platforms_file` (`autosubmit.config.config_common.AutosubmitConfig` attribute), 48

`platforms_parser` (`autosubmit.config.config_common.AutosubmitConfig` attribute), 48

`previous_day()` (in module `autosubmit.date.chunk_date_lib`), 55

`print_job()` (`autosubmit.job.job.Job` method), 59

`print_parameters()` (`autosubmit.config.config_common.AutosubmitConfig` static method), 49

`print_parameters()` (`autosubmit.job.job.Job` method), 59

`project_file` (`autosubmit.config.config_common.AutosubmitConfig` attribute), 49

R

`read()` (`autosubmit.config.basicConfig.BasicConfig` static method), 43

`read_section()` (`autosubmit.job.job_list.DicJobs` method), 61

`reload()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`remove_dependencies()` (`autosubmit.job.job.Job` method), 59

`remove_rerun_only_jobs()` (`autosubmit.job.job_list.JobList` method), 63

`rerun()` (`autosubmit.job.job_list.JobList` method), 63

`result()` (`autosubmit.config.log.Log` static method), 50

S

`save()` (`autosubmit.job.job_list.JobList` method), 63

`set_console_level()` (`autosubmit.config.log.Log` static method), 50

`set_expid()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`set_file()` (`autosubmit.config.log.Log` static method), 50

`set_file_level()` (`autosubmit.config.log.Log` static method), 50

`set_git_project_commit()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`set_platform()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`set_platform()` (`autosubmit.job.job.Job` method), 59

`set_queue()` (`autosubmit.job.job.Job` method), 59

`set_safetysleeptime()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`set_version()` (`autosubmit.config.config_common.AutosubmitConfig` method), 49

`short_name` (`autosubmit.job.job.Job` attribute), 59

`sort_by_id()` (`autosubmit.job.job_list.JobList` method), 64

`sort_by_name()` (`autosubmit.job.job_list.JobList` method), 64

`sort_by_status()` (`autosubmit.job.job_list.JobList` method), 64

`sort_by_type()` (`autosubmit.job.job_list.JobList` method), 64

`StatisticsSnippet` (class in `autosubmit.job.job_common`), 60

`Status` (class in `autosubmit.job.job_common`), 60

`sub_days()` (in module `autosubmit.date.chunk_date_lib`), 55

`subs_dates()` (in module `autosubmit.date.chunk_date_lib`), 55

U

`update_content()` (`autosubmit.job.job.Job` method), 59

`update_from_file()` (`autosubmit.job.job_list.JobList` method), 64

`update_genealogy()` (`autosubmit.job.job_list.JobList` method), 64

`update_list()` (`autosubmit.job.job_list.JobList` method), 64

`update_parameters()` (`autosubmit.job.job.Job` method), 59

`update_shortened_names()` (`autosubmit.job.job_list.JobList` method), 64

`update_status()` (`autosubmit.job.job.Job` method), 60

`user_warning()` (`autosubmit.config.log.Log` static method), 51

W

`warning()` (`autosubmit.config.log.Log` static method), 51

`write_end_time()` (`autosubmit.job.job.Job` method), 60

`write_start_time()` (`autosubmit.job.job.Job` method), 60

`write_submit_time()` (`autosubmit.job.job.Job` method), 60