autosubmit Documentation

Release 4.0.0b

Daniel Beltran

Apr 18, 2024

GETTING STARTED

1 Getting Started 1

2 Installation 7

3 User Guide 15

4 Databases 109

5 Error codes and solutions 113

6 Changelog 117

7 Troubleshooting 133

8 API 135

9 Contact Us 199

Python Module Index 201

Index 203

CHAPTER

ONE

GETTING STARTED

This tutorial is a starter's guide to run a dummy experiment with Autosubmit.

Dummy experiments run workflows with inexpensive empty tasks and therefore are ideal for teaching and testing purposes.

Real experiments instead run workflows with complex tasks. To read information about how to develop parameterizable tasks for Autosubmit workflows, refer to develproject.

1.1 Pre-requisites

Autosubmit needs to establish **password-less SSH connections** in order to run and monitor workflows on remote platforms.

Ensure that you have a **password-less** connection to all platforms you want to use in your experiment. If you are unsure how to do this, please follow these instructions:

- Open a terminal and prompt `ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM`
- Copy the resulting key to your platform of choice. Via SCP or ssh-copy-key.

```
# Generate a key pair for password-less ssh, PEM format is recommended as others can␣
↪cause problems
ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM
# Copy the public key to the remote machine
ssh-copy-id -i ~/.ssh/id_rsa.pub user@remotehost
# Add your key to ssh agent ( if encrypted )
# If not initialized, initialize it
eval `ssh-agent -s`
# Add the key
ssh-add ~/.ssh/id_rsa
# Where ~/.ssh/id_rsa is the path to your private key
```

1.2 Description of most used commands

Command	Short description
expid	Creates a new experiment and generates a new entry in the database by giving it a serial id composed of 4 letters. In addition, it also creates the folder experiment and the basic folder structure.
create <expid>	Generates the experiment workflow.
run <expid>	Runs the experiment workflow.
monitor <expid>	Shows the experiment workflow structure and status.
inspect <expid>	Generates Autosubmit scripts and batch scripts for inspection, by processing the tasks' templates with the experiment parameters.
refresh <expid>	Updates the project directory.
recovery <expid>	Recovers the experiment workflow obtaining the last run complete jobs.
setstatus <expid>	Sets one or multiple jobs status to a given value.

1.3 Create a new experiment

```
autosubmit expid -dm -H "local" -d "Tutorial"
```

- *-dm: Generates a dummy experiment.*
- *-H: Sets the principal experiment platform.*
- *-d: Sets a short description for the experiment.*

The output of the command will show the expid of the experiment and generate the following directory structure:

Experiment folder	Contains
conf	Experiment configuration files.
pkl	Workflow pkl files.
plot	Visualization output files
tmp	Logs, templates and misc files.
proj	User scripts and project code. (empty)

Then, execute `autosubmit create <expid> -np` and Autosubmit will generate the workflow graph.

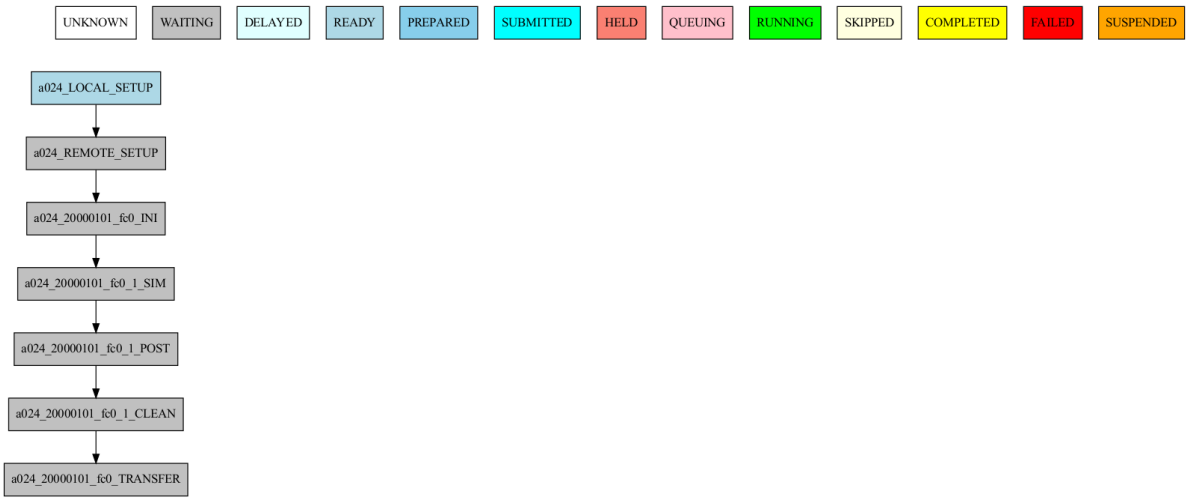
1.4 Run and monitoring

To run an experiment, use `` `autosubmit run <expid>` ``. Autosubmit runs experiments performing the following operations:

- First, it **checks the experiment configuration**. If it is wrong, it won't proceed further.
- Second, it **runs the experiment while retrieving all logs** from completed or failed tasks as they run.
- Third, it manages all the **workflow steps by following the dependencies defined by the user** until all jobs are in COMPLETED or FAILED status. There can be jobs left in **WAITING** status if their dependencies are in **FAILED** status.

While the experiment is running, it can be visualized via `autosubmit monitor <expid>`.

illustrates the output of the autosubmit monitor. It describes all workflow jobs' possible status and actual status.



Concurrently, the `<expid>/tmp` gets filled with the cmd scripts generated by Autosubmit to run the local and remote tasks (in this case, they are sent and submitted to the remote platform(s)).

Autosubmit keeps logs at `ASLOGS` and `LOG_a000` folders, which are filled up with Autosubmit's command logs and job logs.

1.5 Viewing the logs

The `autosubmit` commands such as `expid`, `run`, `monitor`, all may produce log files on the user's file system. To save the user from having to navigate to the log file, or to memorize the location of these files, Autosubmit provides the `autosubmit cat-log` command.

```
$ autosubmit cat-log a000
Autosubmit is running with 4.0.0b
2023-02-27 21:45:47,863 Autosubmit is running with 4.0.0b
2023-02-27 21:45:47,872
Checking configuration files...
2023-02-27 21:45:47,900 expdef_a000.yml OK
2023-02-27 21:45:47,904 platforms_a000.yml OK
2023-02-27 21:45:47,905 jobs_a000.yml OK
2023-02-27 21:45:47,906 autosubmit_a000.yml OK
2023-02-27 21:45:47,907 Configuration files OK
```

Note: The `-f` (`--file`) option is for the file type, not the file path. See the complete help and syntax with `autosubmit cat-log --help` for a list of supported types, depending on whether you choose a workflow log or a job log file. Note too that there is a `-i` (`--inspect`) flag in the command to tell Autosubmit you want job files generated by `autosubmit inspect`, instead of job files generated by `autosubmit run`.

1.6 Configuration summary

In the folder <expid>/conf there are different files that define the actual experiment configuration.

File	Content
expdef.yml	• It contains the default platform, the one set with -H. • Allows changing the start dates, members and chunks. • Allows changing the experiment project source (git, local, svn or dummy)
platforms.yml	• It contains the list of platforms to use in the experiment. • This file contains the definitions for managing clusters, fat-nodes and support computers. • This file must be filled-up with the platform(s) configuration(s). • Several platforms can be defined and used in the same experiment.
jobs.yml	• It contains the tasks' definitions in sections. Depending on the parameters, one section can generate multiple similar tasks. • This file must be filled-up with the tasks' definitions. • Several sections can be defined and used in the same experiment.
autosubmit.yml	• This file contains the definitions that impact the workflow behavior. • It changes workflow behavior with parameters such as job limitations, remote_dependencies and retrials. • It extends autosubmit functionalities with parameters such as wrappers and mail notification.
proj.yml	• This file contains the configuration used by the user scripts. • This file is fully customizable for the current experiment. Allows setting user- parameters that will be readable by the autosubmit jobs.

1.7 Final step: Modify and run

It is time to look into the configuration files of the dummy experiment and modify them with a remote platform to run a workflow with a few more chunks.

Open expdef.yml

```
DEFAULT:
    # Don't change
    EXPID: "a000"
    # Change for your new main platform name, ej. marenostrum4
    HPCARCH: "local"
    # Locate and  change these parameters, per ej. numchunks: 3
    EXPERIMENT:
        DATELIST: 20000101
        MEMBERS: fc0
```

(continues on next page)

(continued from previous page)

```
        NUMCHUNKS: 1
    (...)
```

Now open platforms.yml. Note: This will be an example for marenostrum4

```
PLATFORMS:
    marenostrum4:
        # Queue type. Options: ps, SGE, LSF, SLURM, PBS, eceaccess
        # scheduler type
        TYPE: slurm
        HOST: mn1.bsc.es,mn2.bsc.es,mn3.bsc.es
        # your project
        PROJECT: bsc32
        # <- your user
        USER: bsc32070
        SCRATCH_DIR: /gpfs/scratch
        ADD_PROJECT_TO_HOST: False
        # use 72:00 if you are using a PRACE account, 48:00 for the bsc account
        MAX_WALLCLOCK: 02:00
        # use 19200 if you are using a PRACE account, 2400 for the bsc account
        MAX_PROCESSORS: 2400
        PROCESSORS_PER_NODE: 48
        SERIAL_QUEUE: debug
        QUEUE: debug
```

`autosubmit create <expid>** (without -np)` will generate the new workflow and `autosubmit run <expid>` will run the experiment with the latest changes.

Warning: If you are using an encrypted key, you will need to add it to the ssh-agent before running the experiment. To do so, run `ssh-add <path_to_key>`.

CHAPTER

TWO

INSTALLATION

2.1 How to install

The Autosubmit code is hosted in Git, at the BSC GitLab public repository. The Autosubmit Python package is available through PyPI, the primary source for Python packages.

- Pre-requisites: bash, python3, sqlite3, git-scm > 1.8.2, subversion, dialog, curl, python-tk(tkinter in centOS), graphviz >= 2.41, pip3

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

- Python dependencies: configobj>=5.0.6, argparse>=1.4.0 , python-dateutil>=2.8.2, matplotlib==3.4.3, numpy==1.21.6, py3dotplus>=1.1.0, pyparsing>=3.0.7, paramiko>=2.9.2, mock>=4.0.3, six>=1.10, portalocker>=2.3.2, networkx==2.6.3, requests>=2.27.1, bscearth.utils>=0.5.2, cryptography>=36.0.1, setuptools>=60.8.2, xlib>=0.21, pip>=22.0.3, ruamel.yaml, pythondialog, pytest, nose, coverage, PyNaCl==1.4.0, six>=1.10.0, requests, xlib, Pygments, packaging==19, typing>=3.7, autosubmitconfigparser

Important: `dot -v` command should contain "dot", pdf, png, SVG, Xlib in the device section.

Important: The host machine has to be able to access HPCs/Clusters via password-less ssh. Ensure that the ssh key is in PEM format `ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM`.

To install autosubmit, execute the following:

```
pip install autosubmit
```

Or download, unpack and:

```
python3 setup.py install
```

Hint: To check if Autosubmit is 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`

2.1.1 The sequence of instructions to install Autosubmit and its dependencies with pip.

Warning: The following instructions are for Ubuntu 20.04 LTS. The instructions may vary for other UNIX distributions.

```
# Update repositories
apt update

# Avoid interactive stuff
export DEBIAN_FRONTEND=noninteractive

# Dependencies
apt install wget curl python3 python3-tk python3-dev graphviz -y -q

# Additional dependencies related with pycrypto
apt install build-essential libssl-dev libffi-dev -y -q

# Install Autosubmit using pip
pip3 install autosubmit

# Check that we can execute autosubmit commands
autosubmit -h
```

For a very quick test, you can follow the next instructions to configure and run Autosubmit at the user level. Otherwise, please go directly to How to configure Autosubmit .

```
# Quick-configure ( user-level database)
autosubmit configure

# Install
autosubmit install

# Quick-start

# Get expid
autosubmit expid -H "local" -d "Test exp in local."

# Create with
# Since it was a new install, the expid will be a000
autosubmit create a000

# In case you want to use a remote platform

# Generate a key pair for password-less ssh. PEM format is recommended as others can␣
↪cause problems
```

(continues on next page)

(continued from previous page)

```
ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM

# Copy the public key to the remote machine
ssh-copy-id -i ~/.ssh/id_rsa.pub user@remotehost


# Add your key to the ssh-agent ( if encrypted )

# If not initialized, initialize it
eval `ssh-agent -s`

# Add the key
ssh-add ~/.ssh/id_rsa
# Where ~/.ssh/id_rsa is the path to your private key

# run
autosubmit run a000
```

2.1.2 The sequence of instructions to install Autosubmit and its dependencies with conda.

Warning: The following instructions are for Ubuntu 20.04 LTS. The instructions may vary for other UNIX distributions.

Warning: This procedure is still WIP. You can follow the process at issue #864. We strongly recommend using the pip procedure.

If you don't have conda installed yet, we recommend following Installing Miniconda.

```
# Download git
apt install git -y -q
# Download autosubmit
git clone https://earth.bsc.es/gitlab/es/autosubmit.git -b v4.0.0b
cd autosubmit
# Create a Conda environment from YAML with autosubmit dependencies
conda env create -f environment.yml -n autosubmitenv
# Activate env
conda activate autosubmitenv
# Install autosubmit
pip install autosubmit
# Test autosubmit
autosubmit -v
```

For a very quick test, you can follow the next instructions to configure and run Autosubmit at the user level. Otherwise, please go directly to How to configure Autosubmit

```
# Quick-configure ( user-level database)
autosubmit configure

# Install
autosubmit install

# Quick-start
# Get expid
autosubmit expid -H "local" -d "Test exp in local."

# Create with
# Since it was a new install, the expid will be a000
autosubmit create a000

# In case you want to use a remote platform

# Generate a key pair for password-less ssh. PEM format is recommended as others can␣
↪cause problems
ssh-keygen -t rsa -b 4096 -C "email@email.com" -m PEM

# Copy the public key to the remote machine
ssh-copy-id -i ~/.ssh/id_rsa.pub user@remotehost

# Add your key to ssh agent ( if encrypted )
# If not initialized, initialize it
eval `ssh-agent -s`
# Add the key
ssh-add ~/.ssh/id_rsa
# Where ~/.ssh/id_rsa is the path to your private key

# run
autosubmit run a000
```

Hint: After installing the Conda, you may need to close the terminal and re-open it so the installation takes effect.

2.2 How to configure Autosubmit

There are two methods of configuring the Autosubmit main paths.

- `autosubmit configure` is suited for a personal/single user who wants to test Autosubmit in the scope of `$HOME`. It will generate an `$HOME/.autosubmitrc` file that overrides the machine configuration.

Manually generate an `autosubmitrc` file in one of these locations, which is the recommended method for a production environment with a shared database in a manner that multiple users can share and view others' experiments.

- `/etc/autosubmitrc`, System level configuration.
- Set the environment variable `AUTOSUBMIT_CONFIGURATION` to the path of the `autosubmitrc` file. This will override all other configuration files.

Important: *.autosubmitrc* user level precedes system configuration unless the environment variable is set. *AUTO-*

SUBMIT_CONFIGURATION > $HOME/.autosubmitrc > /etc/autosubmitrc

2.2.1 Quick Installation - Non-shared database (user level)

After the package installation, you have to configure at least the database and path for Autosubmit.

To use the default settings, create a directory called `autosubmit` (`mkdir $HOME/autosubmit`) in your home directory before running the `configure` command.

```
autosubmit configure
```

`autosubmit generate` will always generate a file called `.autosubmitrc` in your `$HOME`.

You can add `--advanced` to the configure command for advanced options.

```
autosubmit configure --advanced
```

It will allow you to choose different directories:

- Experiments path and database name (`$HOME/autosubmit/` by default) and database name (`$HOME/autosubmit/autosubmit.db` by default)
- Path for the global logs (those not belonging to any experiment). Default is `$HOME/autosubmit/logs`.
- Autosubmit metadata. Default is `$HOME/autosubmit/metadata/`

Additionally, it also provides the possibility of configuring an SMTP server and an email account to use the email notifications feature.

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

Example - Local - .autosubmitrc skeleton

```
[database]
path = /home/dbeltran/autosubmit
filename = autosubmit.db

[local]
path = /home/dbeltran/autosubmit

[globallogs]
path = /home/dbeltran/autosubmit/logs

[structures]
path = /home/dbeltran/autosubmit/metadata/structures

[historicdb]
path = /home/dbeltran/autosubmit/metadata/data

[historiclog]
path = /home/dbeltran/autosubmit/metadata/logs
```

2.2.2 Production environment installation - Shared-Filesystem database

Warning: Keep in mind the .autosubmitrc precedence. If you, as a user, have a .autosubmitrc generated in the quick-installation, you have to delete or rename it before using the production environment installation.

Create an `/etc/autosubmitrc` file or move it from `$HOME/.autosubmitrc` to `/etc/autosubmitrc` with the information as follows:

Mandatory parameters of /etc/autosubmit

```
[database]
# Accessible for all users of the filesystem
path = <database_path>
# Experiment database name can be whatever.
filename = autosubmit.db

# Accessible for all users of the filesystem, can be the same as database_path
[local]
path = <experiment_path>

# Global logs, logs without expid associated.
[globallogs]
path = /home/dbeltran/autosubmit/logs

# This depends on your email server and can be left empty if not applicable
[mail]
smtp_server = mail.bsc.es
mail_from = automail@bsc.es
```

Recommendable parameters of /etc/autosubmit

The following parameters are the Autosubmit metadata, it is not mandatory, but it is recommendable to have them set up as some of them can positively affect the Autosubmit performance.

```
[structures]
path = /home/dbeltran/autosubmit/metadata/structures

[historicdb]
path = /home/dbeltran/autosubmit/metadata/data

[historiclog]
path = /home/dbeltran/autosubmit/metadata/logs
```

Optional parameters of /etc/autosubmit

These parameters provide extra functionalities to Autosubmit.

```
[conf]
# Allows using a different jobs.yml default template on `autosubmit expid ``
jobs = <path_jobs>/jobs.yml
# Allows using a different platforms.yml default template on `autosubmit expid `
platforms = <path_platforms>platforms.yml> path to any jobs.yml

# Autosubmit API includes extra information for some Autosubmit functions. It is␣
↪optional to have access to it to use Autosubmit.
[autosubmitapi]
# Autosubmit API (The API is right now only provided inside the BSC network), which␣
↪enables extra features for the Autosubmit GUI
url = <url of the Autosubmit API>:<port>

# Used for controlling the traffic that comes from Autosubmit.
[hosts]
authorized =  [<command1,commandN> <machine1,machineN>]
forbidden =   [<command1,commandN> <machine1,machineN>]
```

About hosts parameters:

From 3.14+ onwards, the users can tailor Autosubmit commands to run on specific machines. Previously, only the run was affected by the deprecated whitelist parameter.

- authorized = [<command1,commandN> <machine1,machineN>] list of machines that can run given autosubmit commands. If the list is empty, all machines are allowed.
- forbidden = [<command1,commandN> <machine1,machineN>] list of machines that cannot run given autosubmit commands. If the list is empty, no machine is forbidden.

Example - BSC - /etc/autosubmitrc skeleton

```
[database]
path = /esarchive/autosubmit
filename = ecearth.db

[local]
path = /esarchive/autosubmit

[conf]
jobs = /esarchive/autosubmit/default
platforms = /esarchive/autosubmit/default

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

[hosts]
     authorized =  [run bscearth000,bscesautosubmit01,bscesautosubmit02] [stats, clean,␣
↪describe, check, report,dbfix,pklfix, upgrade,updateversion all]
     forbidden =  [expid, create, recovery, delete, inspect, monitor, recovery, migrate,␣
```

(continues on next page)

(continued from previous page)

```
↪configure,setstatus,testcase, test, refresh, archive, unarchive bscearth000,
↪bscesautosubmit01,bscesautosubmit02]
```

2.3 Experiments database installation

As the last step, ensure to install the Autosubmit database. To do so, execute `autosubmit install`.

```
autosubmit install
```

This command will generate a blank database in the specified configuration path.

CHAPTER

THREE

USER GUIDE

3.1 Create an Experiment

3.1.1 Create new 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 | -min [ -repo GIT_PATH -b BRANCH -config AS_
↪CONF ] ] [-p PATH] -H HPC -d DESCRIPTION

    -h, --help            show this help message and exit
    -y COPY, --copy COPY  makes a copy of the specified experiment
    -op, -operational     creates a new experiment, starting with "o"
    -dm, --dummy          creates a new experiment with default values, usually for␣
↪testing
    -min, --minimal_config
        creates a new experiment with minimal configuration files, usually for using a␣
↪custom configuration
    -repo GIT_PATH, --git_repo GIT_PATH
        sets the git_repository
    -b BRANCH, --git_branch BRANCH
        sets the branch to use for the git repository
    -config, --git_as_conf
        sets the configuration folder to use for the experiment, relative to repo root
    -local, --use_local_minimal obtains the minimal configuration for local files
    -H HPC, --HPC HPC     specifies the HPC to use for the experiment, default is␣
↪localhost
    -d DESCRIPTION, --description DESCRIPTION
        sets a description for the experiment to store in the database.
```

Example:

```
autosubmit expid --HPC marenostrum4 --description "experiment is about..."
autosubmit expid -min -repo https://earth.bsc.es/gitlab/ces/auto-advanced_config_example␣
```

(continues on next page)

(continued from previous page)

```
↪-b main -conf as_conf -d "minimal config example"
autosubmit expid -dm -d "dummy test"
```

If there is an autosubmitrc or .autosubmitrc file in your home directory (cd ~), you can setup a default file from where the contents of platforms_expid.yml should be copied.

In this autosubmitrc or .autosubmitrc file, include the configuration setting custom_platforms:

Example:

```
conf:
    custom_platforms: /home/Earth/user/custom.yml
```

Where the specified path should be complete, as something you would get when executing pwd, and also include the filename of your custom platforms content.

3.1.2 Copy another 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 -y COPY -H HPCname -d Description
autosubmit expid -y COPY -c PATH -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. *COPY* is the experiment identifier to copy from. *Description* is a brief experiment description. *CONFIG* is a folder that exists.

Example:

```
autosubmit expid -y cxxx -H ithaca -d "experiment is about..."
autosubmit expid -y cxxx -p "/esarchive/autosubmit/genericFiles/conf" -H marenostrum4 -d
↪"experiment is about..."
```

Warning: You can only copy experiments created with Autosubmit 3.11 or above.

If there is an autosubmitrc or .autosubmitrc file in your home directory (cd ~), you can setup a default file from where the contents of platforms_expid.yml should be copied.

In this autosubmitrc or .autosubmitrc file, include the configuration setting custom_platforms:

Example:

```
conf:
custom_platforms: /home/Earth/user/custom.yml
```

Where the specified path should be complete, as something you would get when executing pwd, and also include the filename of your custom platforms content.

3.1.3 Create a dummy experiment

It is useful to test if Autosubmit is properly configured with a inexpensive experiment. A Dummy experiment will check, test, and submit to the HPC platform, as any other experiment would.

The job submitted are only sleeps.

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..."
```

3.1.4 Create a test case experiment

Test case experiments are special experiments which have a reserved first letter "t" at the expid. They are meant to help differentiate testing suits of the automodels from normal runs.

This method is to create a test case experiment. It creates a new experiment for a test case with a given number of chunks, start date, member and HPC.

To create a test case experiment, use the command:

```
autosubmit testcase
```

Options:

```
usage: autosubmit testcase [-h] [-y COPY] -d DESCRIPTION [-c CHUNKS]
                           [-m MEMBER] [-s STARDATE] [-H HPC] [-b BRANCH]

    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 testcase -d "TEST CASE cca-intel auto-ecearth3 layer 0: T511L91-ORCA025L75-
↪LIM3 (cold restart) (a092-a09n)" -H cca-intel -b 3.2.0b_develop -y a09n
```

3.1.5 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
```

3.1.6 How to profile Autosubmit while creating an experiment

Autosubmit offers the possibility to profile the experiment creation process. To enable the profiler, just add the `--profile` (or `-p`) flag to your `autosubmit create` command, as in the following example:

```
autosubmit create --profile EXPID
```

Note: Remember that the purpose of this profiler is to measure the performance of Autosubmit, not the jobs it runs.

This profiler uses Python's `cProfile` and `psutil` modules to generate a report with simple CPU and memory metrics which will be displayed in your console after the command finishes, as in the example below:

The profiler output is also saved in `<EXPID>/tmp/profile`. There you will find two files, the report in plain text format and a `.prof` binary which contains the CPU metrics. We highly recommend using SnakeViz to visualize this file, as follows:

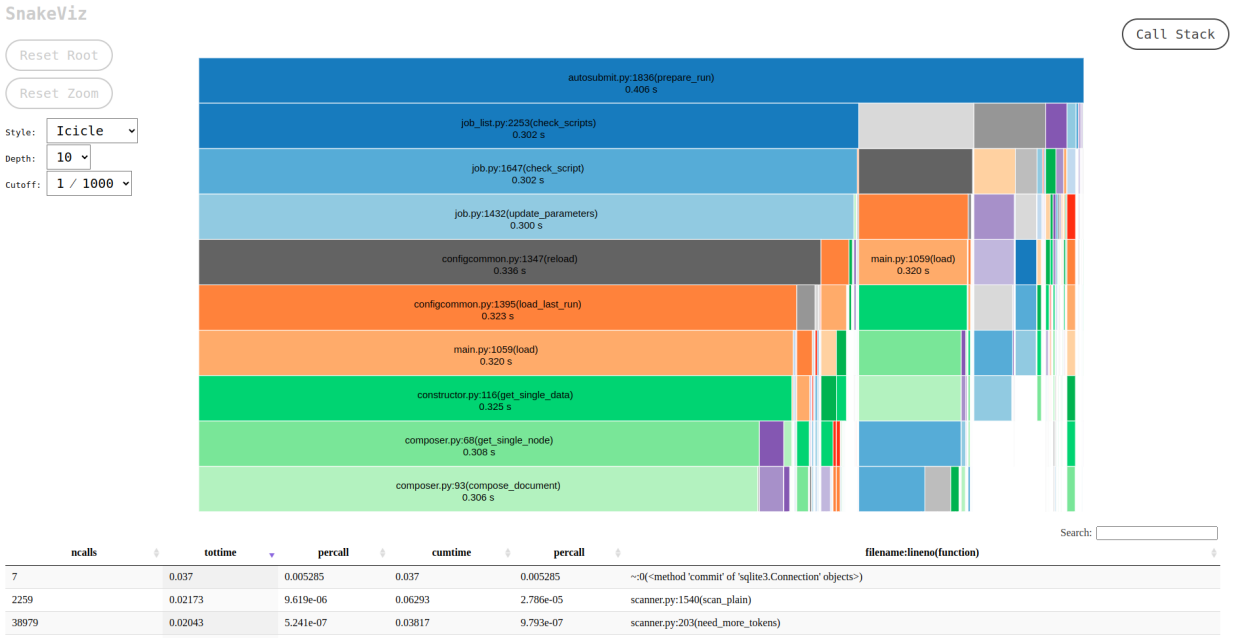
For more detailed documentation about the profiler, please visit this page.

```
No more jobs to run.
Run successful
================================================================================
                            Time & Calls Profiling
================================================================================

         1091713 function calls (1081212 primitive calls) in 0.581 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.564    0.564 autosubmit.py:1826(prepare_run)
        9    0.000    0.000    0.484    0.054 configcommon.py:1343(reload)
       17    0.000    0.000    0.470    0.028 constructor.py:116(get_single_data)
        9    0.000    0.000    0.467    0.052 configcommon.py:1391(load_last_run)
        9    0.000    0.000    0.463    0.051 main.py:1059(load)
        9    0.000    0.000    0.446    0.050 composer.py:68(get_single_node)
        9    0.000    0.000    0.445    0.049 composer.py:93(compose_document)
   2835/9    0.010    0.000    0.445    0.049 composer.py:111(compose_node)
    378/9    0.004    0.000    0.444    0.049 composer.py:199(compose_mapping_node)
        1    0.000    0.000    0.442    0.442 job_list.py:2253(check_scripts)
        8    0.000    0.000    0.441    0.055 job.py:1647(check_script)
        8    0.000    0.000    0.439    0.055 job.py:1432(update_parameters)
     8361    0.007    0.000    0.362    0.000 parser.py:141(check_event)
```



3.2 Configure Experiments

3.2.1 How to configure experiments

Edit `expdef_cxxx.yml`, `jobs_cxxx.yml` and `platforms_cxxx.yml` in the `conf` folder of the experiment.

***expdef_cxxx.yml* contains:**

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

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

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

***platforms_cxxx.yml* contains:**

- HPC, fat-nodes and supporting computers configuration.

Note: *platforms_cxxx.yml* 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.yml`.

***autosubmit_cxxx.yml* 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.yml` 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 [-group_by {date,member,chunk,split} -expand -expand_status] [-
↪h] [-np] [-cw] expid

  expid          experiment identifier

  -h, --help     show this help message and exit
  -np, --noplot  omit plot creation
  --hide,        hide the plot
  -group_by {date,member,chunk,split,automatic}
```

(continues on next page)

(continued from previous page)

```
                        criteria to use for grouping jobs
  -expand,              list of dates/members/chunks to expand
  -expand_status,       status(es) to expand
  -nt                   --notransitive
                            prevents doing the transitive reduction when plotting the␣
↪workflow
  -cw                   --check_wrapper
                            Generate the wrapper in the current workflow
  -d                    --detail
                            Shows Job List view in terminal
```

Example:

```
autosubmit create cxxx
```

In order to understand more the grouping options, which are used for visualization purposes, please check *Grouping jobs*.

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

3.2.2 How to add a new job

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

```
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.yml 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 start-date, 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 start-dates, use -(DISTANCE). For example, for a job "SIM" waiting for the previous "SIM" job to finish, you have to add "DEPENDENCIES: SIM-1".
  - For dependencies that are not mandatory for the normal workflow behaviour, you must add the char '?' at the end of the dependency.

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.
- NODES: nodes number to be submitted to the HPC. If not specified, the directive is not added.
- HYPERTHREADING: Enables Hyper-threading, this will double the max amount of threads. defaults to false. (Not available on slurm platforms)
- QUEUE: queue to add the job to. If not specified, uses PLATFORM default.
- RETRIALS: Number of retrials if job fails
- DELAY_RETRY_TIME: Allows to put a delay between retries. Triggered when a job fails. If not specified, Autosubmit will retry the job as soon as possible. Accepted formats are: plain number (there will be a constant delay between retrials, of as many seconds as specified), plus (+) sign followed by a number (the delay will steadily increase by the addition of these number of seconds), or multiplication (*) sign follows by a number (the delay after n retries will be the number multiplied by 10*n). Having this in mind, the ideal scenario is to use +(number) or plain(number) in case that the HPC has little issues or the experiment will run for a little time. Otherwise, is better to use the *(number) approach.

```
#DELAY_RETRY_TIME: 11
#DELAY_RETRY_TIME: +11 # will wait 11 + number specified
#DELAY_RETRY_TIME:*11 # will wait 11,110,1110,11110...* by 10 to prevent a too big number
```

There are also other, 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
- SYNCHRONIZE: specifies that a job with RUNNING: chunk, has to synchronize its dependencies chunks at a 'date' or 'member' level, which means that the jobs will be unified: one per chunk for all members or dates. If not specified, the synchronization is for each chunk of all the experiment.
- RERUN_ONLY: determines if a job is only to be executed in reruns. If not specified, defaults to false.
- CUSTOM_DIRECTIVES: Custom directives for the HPC resource manager headers of the platform used for that job.
- SKIPPABLE: When this is true, the job will be able to skip it work if there is an higher chunk or member already ready, running, queuing or in complete status.
- EXPORT: Allows to run an env script or load some modules before running this job.
- EXECUTABLE: Allows to wrap a job for be launched with a set of env variables.
- QUEUE: queue to add the job to. If not specified, uses PLATFORM default.
- EXTENDED_HEADER_PATH: specify the path relative to the project folder where the extension to the autosubmit's header is
- EXTENDED_TAILER_PATH: specify the path relative to the project folder where the extension to the autosubmit's tailer is

3.2.3 How to add a new heterogeneous job (hetjob)

A hetjob, is a job in which each component has virtually all job options available including partition, account and QOS (Quality Of Service).For example, part of a job might require four cores and 4 GB for each of 128 tasks while another part of the job would require 16 GB of memory and one CPU.

This feature is only available for SLURM platforms. And it is automatically enabled when the processors or nodes paramater is a yaml list

To add a new hetjob, open the <experiments_directory>/cxxx/conf/jobs_cxxx.yml file where cxxx is the experiment

```
JOBS:
    new_hetjob:
        FILE: <new_job_template>
        PROCESSORS: # Determines the amount of components that will be created
            - 4
            - 1
        MEMORY: # Determines the amount of memory that will be used by each component
            - 4096
            - 16384
        WALLCLOCK: 00:30
        PLATFORM: <platform_name> # Determines the platform where the job will be␣
↪executed
        PARTITION: # Determines the partition where the job will be executed
            - <partition_name>
            - <partition_name>
        TASKS: 128 # Determines the amount of tasks that will be used by each component
```

This will create a new job named "new_hetjob" with two components that will be executed once.

- EXTENDED_HEADER_PATH: specify the path relative to the project folder where the extension to the autosubmit's header is
- EXTENDED_TAILER_PATH: specify the path relative to the project folder where the extension to the autosubmit's tailer is

3.2.4 How to configure email notifications

To configure the email notifications, you have to follow two configuration steps:

1. First you have to enable email notifications and set the accounts where you will receive it.

Edit `autosubmit_cxxx.yml` in the `conf` folder of the experiment.

Hint: Remember that you can define more than one email address divided by a whitespace.

Example:

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

```
mail:
    # Enable mail notifications for remote_failures
    # Default:True
    NOTIFY_ON_REMOTE_FAIL: True
```

(continues on next page)

(continued from previous page)

```
    # Enable mail notifications
    # Default: False
    NOTIFICATIONS: True
    # Mail address where notifications will be received
    TO:   jsmith@example.com  rlewis@example.com
```

2. Then you have to define for which jobs you want to be notified.

Edit `jobs_cxxx.yml` in the `conf` folder of the experiment.

Hint: You will be notified every time the job changes its status to one of the statuses defined on the parameter `NOTIFY_ON`

Hint: Remember that you can define more than one job status divided by a whitespace.

Example:

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

```
JOBS:
    LOCAL_SETUP:
        FILE: LOCAL_SETUP.sh
        PLATFORM: LOCAL
        NOTIFY_ON: FAILED COMPLETED
```

3.2.5 How to add a new platform

Hint: If you are interested in changing the communications library, go to the section below.

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

```
PLATFORMS:
    new_platform:
        # MANDATORY
        TYPE: <platform_type>
        HOST: <host_name>
        PROJECT: <project>
        USER: <user>
        SCRATCH: <scratch_dir>
        MAX_WALLCLOCK: <HH:MM>
        QUEUE: <hpc_queue>
        # OPTIONAL
        ADD_PROJECT_TO_HOST: False
        MAX_PROCESSORS: <N>
        EC_QUEUE : <ec_queue> # only when type == ecaccess
        VERSION: <version>
```

(continues on next page)

(continued from previous page)

```
        2FA: False
        2FA_TIMEOUT: <timeout> # default 300
        2FA_METHOD: <method>
        SERIAL_PLATFORM: <platform_name>
        SERIAL_QUEUE: <queue_name>
        BUDGET: <budget>
        TEST_SUITE: False
        MAX_WAITING_JOBS: <N>
        TOTAL_JOBS: <N>
        CUSTOM_DIRECTIVES: "[ 'my_directive' ]"
```

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
- MAX_WALLCLOCK: maximum wallclock time allowed for a job in the platform
- MAX_PROCESSORS: maximum number of processors allowed for a job in the platform
- EC_QUEUE: queue for the ecaccess platform. (hpc, ecs)

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, slurm).

- VERSION: determines de version of the platform type

Warning: With some platforms, 2FA authentication is required. If this is the case, you have to add the parameter 2FA. These platforms are ecaccess (options: True, False). There may be some autosubmit functions that are not avaliable when using an interactive auth method.

- 2FA: determines if the platform requires 2FA authentication. (default: False)
- 2FA_TIMEOUT: determines the timeout for the 2FA authentication. (default: 300)
- 2FA_METHOD: determines the method for the 2FA authentication. (default: token)

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 may 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.
- CUSTOM_DIRECTIVES: Custom directives for the resource manager of this platform.

3.2.6 How to request exclusivity or reservation

To request exclusivity or reservation for your jobs, you can configure two platform variables:

Edit `platforms_cxxx.yml` in the `conf` folder of the experiment.

Hint: Until now, it is only available for Marenostrum.

Hint: To define some jobs with exclusivity/reservation and some others without it, you can define twice a platform, one with this parameters and another one without it.

Example:

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

```
PLATFORMS:
    marenostrum3:
        TYPE: LSF
        HOST: mn-bsc32
        PROJECT: bsc32
        ADD_PROJECT_TO_HOST: false
        USER: bsc32XXX
        SCRATCH_DIR: /gpfs/scratch
        TEST_SUITE: True
        EXCLUSIVITY: True
```

Of course, you can configure only one or both. For example, for reservation it would be:

Example:

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

```
PLATFORMS:
    marenostrum3:
        TYPE: LSF
        ...
        RESERVATION: your-reservation-id
```

3.2.7 How to set a custom interpreter for your job

If the remote platform does not implement the interpreter you need, you can customize the shebang of your job script so it points to the relative path of the interpreter you want.

In the file:

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

```
JOBS:
    # Example job with all options specified

    ## Job name
    # JOBNAME:
    ## Script to execute. If not specified, job will be omitted from workflow. You can␣
↪also specify additional files separated by a ",".
    # Note: The post processed additional_files will be sent to %HPCROOT%/LOG_%EXPID%
    ## Path relative to the project directory
    # FILE :
    ## Platform to execute the job. If not specified, defaults to HPCARCH in expdef file.
    ## LOCAL is always defined and refers to current machine
    # PLATFORM :
    ## Queue to add the job to. If not specified, uses PLATFORM default.
    # QUEUE :
    ## Defines dependencies from job as a list of parents jobs separated by spaces.
    ## Dependencies to jobs in previous chunk, member o startdate, use -(DISTANCE)
    # DEPENDENCIES:  INI SIM-1 CLEAN-2
    ## Define if jobs runs once, once per stardate, once per member or once per chunk.␣
↪Options: once, date, member, chunk.
    ## If not specified, defaults to once
    # RUNNING:  once
    ## Specifies that job has only to be run after X dates, members or chunk. A job will␣
↪always be created for the last
    ## If not specified, defaults to 1
    # FREQUENCY:  3
    ## On a job with FREQUENCY > 1, if True, the dependencies are evaluated against all
    ## jobs in the frequency interval, otherwise only evaluate dependencies against␣
↪current
    ## iteration.
    ## If not specified, defaults to True
    # WAIT:  False
    ## Defines if job is only to be executed in reruns. If not specified, defaults to␣
↪false.
    # RERUN_ONLY:  False
    ## Wallclock to be submitted to the HPC queue in format HH:MM
    # WALLCLOCK:  00:05
    ## Processors number to be submitted to the HPC. If not specified, defaults to 1.
    ## Wallclock chunk increase (WALLCLOCK will be increased according to the formula␣
↪WALLCLOCK + WCHUNKINC * (chunk - 1)).
    ## Ideal for sequences of jobs that change their expected running time according to␣
↪the current chunk.
    # WCHUNKINC:  00:01
    # PROCESSORS:  1
    ## Threads number to be submitted to the HPC. If not specified, defaults to 1.
```

(continues on next page)

(continued from previous page)

```
    # THREADS:  1
    ## Tasks number to be submitted to the HPC. If not specified, defaults to 1.
    # Tasks:  1
    ## Enables hyper-threading. If not specified, defaults to false.
    # HYPERTHREADING:  false
    ## Memory requirements for the job in MB
    # MEMORY:  4096
    ##  Number of retrials if a job fails. If not specified, defaults to the value given␣
↪on experiment's autosubmit.yml
    # RETRIALS:  4
    ##  Allows to put a delay between retries, of retrials if a job fails. If not␣
↪specified, it will be static
    # The ideal is to use the +(number) approach or plain(number) in case that the hpc␣
↪platform has little issues or the experiment will run for a short period of time
    # And *(10) in case that the filesystem is having large  delays or the experiment␣
↪will run for a lot of time.
    # DELAY_RETRY_TIME:  11
    # DELAY_RETRY_TIME:  +11 # will wait 11 + number specified
    # DELAY_RETRY_TIME:  *11 # will wait 11,110,1110,11110...* by 10 to prevent a too␣
↪big number
    ## Some jobs can not be checked before running previous jobs. Set this option to␣
↪false if that is the case
    # CHECK:  False
    ## Select the interpreter that will run the job. Options: bash, python, r Default:␣
↪bash
    # TYPE:  bash
    ## Specify the path to the interpreter. If empty, use system default based on job␣
↪type  . Default: empty
    # EXECUTABLE:  /my_python_env/python3
```

You can give a path to the `EXECUTABLE` setting of your job. Autosubmit will replace the `shebang` with the path you provided.

Example:

```
JOBS:
    POST:
        FILE:  POST.sh
        DEPENDENCIES:  SIM
        RUNNING:  chunk
        WALLCLOCK:  00:05
        EXECUTABLE:  /my_python_env/python3
```

This job will use the python interpreter located in the relative path `/my_python_env/python3/`

It is also possible to use variables in the `EXECUTABLE` path.

Example:

```
JOBS:
    POST:
        FILE: POST.sh
        DEPENDENCIES: SIM
        RUNNING: chunk
```

(continues on next page)

(continued from previous page)

```
        WALLCLOCK: 00:05
        EXECUTABLE: "%PROJDIR%/my_python_env/python3"
```

The result is a shebang line `#!/esarchive/autosubmit/my_python_env/python3`.

3.2.8 How to create and run only selected members

Your experiment is defined and correctly configured, but you want to create it only considering some selected members, and also to avoid creating the whole experiment to run only the members you want. Then, you can do it by configuring the setting **RUN_ONLY_MEMBERS** in the file:

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

```
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
    # List of members that can be included in this run. Optional.
    # RUN_ONLY_MEMBERS: fc0 fc1 fc2 fc3 fc4
    # RUN_ONLY_MEMBERS: fc[0-4]
    RUN_ONLY_MEMBERS:
```

You can set the **RUN_ONLY_MEMBERS** value as shown in the format examples above it. Then, `Job List` generation is performed as usual. However, an extra step is performed that will filter the jobs according to **RUN_ONLY_MEMBERS**. It discards jobs belonging to members not considered in the value provided, and also we discard these jobs from the dependency tree (parents and children). The filtered `Job List` is returned.

The necessary changes have been implemented in the API so you can correctly visualize experiments implementing this new setting in **Autosubmit GUI**.

Important: Wrappers are correctly formed considering the resulting jobs.

3.2.9 Remote Dependencies - Presubmission feature

There is also the possibility of setting the option **PRESUBMISSION** to True in the config directive. This allows more than one package containing simple or wrapped jobs to be submitted at the same time, even when the dependencies between jobs aren't yet satisfied.

This is only useful for cases when the job scheduler considers the time a job has been queuing to determine the job's priority (and the scheduler understands the dependencies set between the submitted packages). New packages can be created as long as the total number of jobs are below than the number defined in the **TOTALJOBS** variable.

The jobs that are waiting in the remote platform, will be marked as HOLD.

How to configure

In `autosubmit_cxxx.yml`, regardless of the how your workflow is configured.

For example:

```
config:
    EXPID: ....
    AUTOSUBMIT_VERSION: 4.0.0
    ...
    MAXWAITINGJOBS: 100
    TOTALJOBS: 100
    ...
```

3.3 Defining the workflow

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

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

3.3.1 Simple workflow

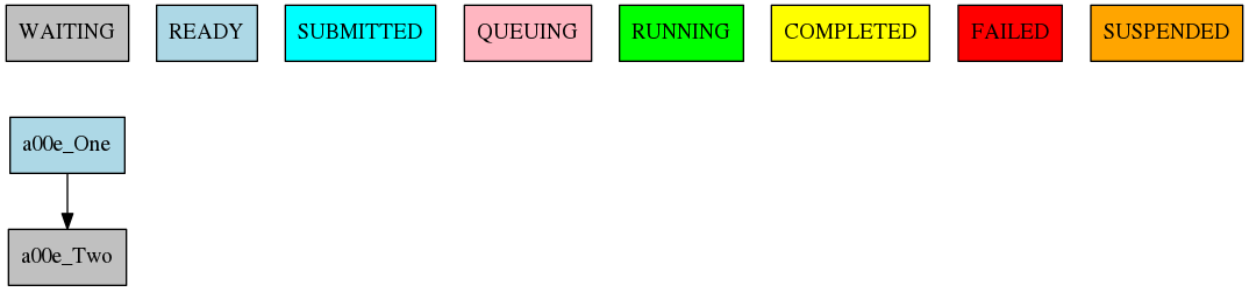
The simplest workflow that can be defined it is a sequence of two jobs, with the second one triggering at the end of the first. To define it, we define the two jobs and then add a DEPENDENCIES attribute on the second job referring to the first one.

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

```
One:
    FILE: one.sh

Two:
    FILE: two.sh
    DEPENDENCIES: One
```

The resulting workflow can be seen in Fi165gure 3.1



3.1: Example showing a simple workflow with two sequential jobs

3.3.2 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.

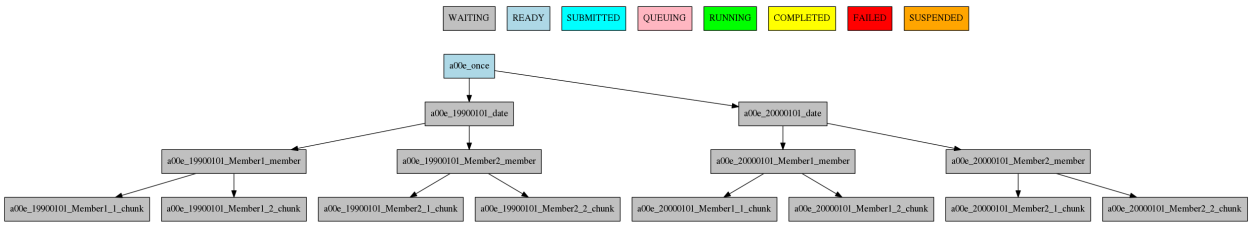
```
once:
    FILE: Once.sh

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 in Figure 3.2 for a experiment with 2 startdates, 2 members and 2 chunks.



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

3.3.3 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.

Dependencies with previous jobs

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

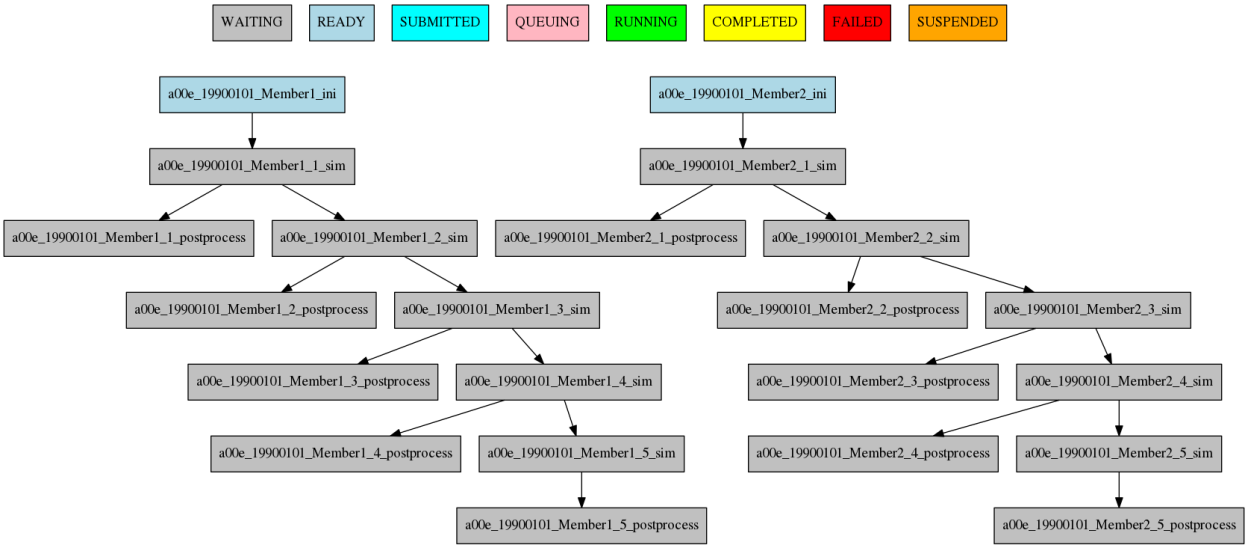
```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: chunk

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

The resulting workflow can be seen in Figure 3.3

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



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

Dependencies between running levels

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

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

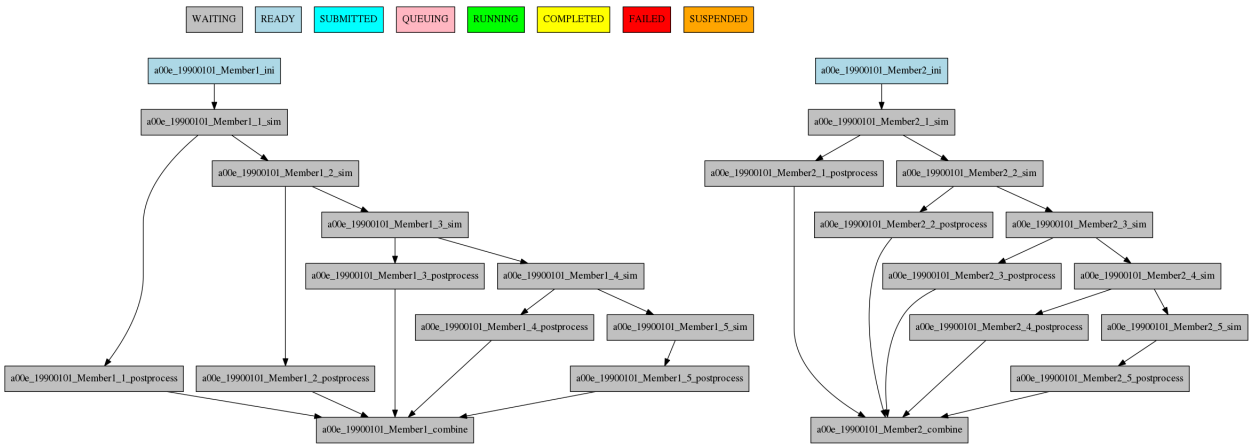
```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: chunk

postprocess:
    FILE: postprocess.sh
    DEPENDENCIES: sim
    RUNNING: chunk

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

The resulting workflow can be seen in Figure 3.4



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

Dependencies rework

The DEPENDENCIES key is used to define the dependencies of a job. It can be used in the following ways:

- Basic: The dependencies are a list of jobs, separated by " ", that runs before the current task is submitted.
- New: The dependencies is a list of YAML sections, separated by "n", that runs before the current job is submitted.
  - For each dependency section, you can designate the following keywords to control the current job-affected tasks:
    - DATES_FROM: Selects the job dates that you want to alter.
    - MEMBERS_FROM: Selects the job members that you want to alter.
    - CHUNKS_FROM: Selects the job chunks that you want to alter.
  - For each dependency section and *_FROM keyword, you can designate the following keywords to control the destination of the dependency:
    - DATES_TO: Links current selected tasks to the dependency tasks of the dates specified.
    - MEMBERS_TO: Links current selected tasks to the dependency tasks of the members specified.
    - CHUNKS_TO: Links current selected tasks to the dependency tasks of the chunks specified.
  - Important keywords for [DATES|MEMBERS|CHUNKS]_TO:
    - "natural": Will keep the default linkage. Will link if it would be normally. Example, SIM_FC00_CHUNK_1 -> DA_FC00_CHUNK_1.
    - "all": Will link all selected tasks of the dependency with current selected tasks. Example, SIM_FC00_CHUNK_1 -> DA_FC00_CHUNK_1, DA_FC00_CHUNK_2, DA_FC00_CHUNK_3…
    - "none": Will unlink selected tasks of the dependency with current selected tasks.

For the new format, consider that the priority is hierarchy and goes like this DATES_FROM -(includes)-> MEMBERS_FROM -(includes)-> CHUNKS_FROM.

- You can define a DATES_FROM inside the DEPENDENCY.
- You can define a MEMBERS_FROM inside the DEPENDENCY and DEPENDENCY.DATES_FROM.
- You can define a CHUNKS_FROM inside the DEPENDENCY, DEPENDENCY.DATES_FROM, DEPENDENCY.MEMBERS_FROM, DEPENDENCY.DATES_FROM.MEMBERS_FROM

Start conditions

Sometimes you want to run a job only when a certain condition is met. For example, you may want to run a job only when a certain task is running. This can be achieved using the START_CONDITIONS feature based on the dependencies rework.

Start conditions are achieved by adding the keyword `STATUS` and optionally `FROM_STEP` keywords into any dependency that you want.

The `STATUS` keyword can be used to select the status of the dependency that you want to check. The possible values (case-insensitive) are:

- "WAITING": The task is waiting for its dependencies to be completed.
- "DELAYED": The task is delayed by a delay condition.
- "PREPARED": The task is prepared to be submitted.
- "READY": The task is ready to be submitted.
- "SUBMITTED": The task is submitted.
- "HELD": The task is held.
- "QUEUING": The task is queuing.
- "RUNNING": The task is running.
- "SKIPPED": The task is skipped.
- "FAILED": The task is failed.
- "UNKNOWN": The task is unknown.
- "COMPLETED": The task is completed. # Default
- "SUSPENDED": The task is suspended.

The status are ordered, so if you select "RUNNING" status, the task will be run if the parent is in any of the following statuses: "RUNNING", "QUEUING", "HELD", "SUBMITTED", "READY", "PREPARED", "DELAYED", "WAITING".

```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: chunk

postprocess:
    FILE: postprocess.sh
    DEPENDENCIES:
        SIM:
            STATUS: "RUNNING"
    RUNNING: chunk
```

The `FROM_STEP` keyword can be used to select the **internal** step of the dependency that you want to check. The possible value is an integer. Additionally, the target dependency, must call to *%AS_CHECKPOINT%* inside their scripts. This will create a checkpoint that will be used to check the amount of steps processed.

```
A:
  FILE: a.sh
  RUNNING: once
  SPLITS: 2
A_2:
  FILE: a_2.sh
  RUNNING: once
  DEPENDENCIES:
    A:
      SPLIT_TO: "2"
      STATUS: "RUNNING"
      FROM_STEP: 2
```

There is now a new function that is automatically added in your scripts which is called `as_checkpoint`. This is the function that is generating the checkpoint file. You can see the function below:

```
###################
# AS CHECKPOINT FUNCTION
###################
# Creates a new checkpoint file upon call based on the current numbers of calls to the␣
↪function

AS_CHECKPOINT_CALLS=0
function as_checkpoint {
    AS_CHECKPOINT_CALLS=$((AS_CHECKPOINT_CALLS+1))
    touch ${job_name_ptrn}_CHECKPOINT_${AS_CHECKPOINT_CALLS}
}
```

And what you would have to include in your target dependency or dependencies is the call to this function which in this example is a.sh.

The amount of calls is strongly related to the `FROM_STEP` value.

`$expid/proj/$projname/as.sh`

```
##compute somestuff
as_checkpoint
## compute some more stuff
as_checkpoint
```

To select an specific task, you have to combine the `STATUS` and `CHUNKS_TO` , `MEMBERS_TO` and `DATES_TO`, `SPLITS_TO` keywords.

```
A:
  FILE: a
  RUNNING: once
  SPLITS: 1
B:
  FILE: b
  RUNNING: once
  SPLITS: 2
  DEPENDENCIES: A
C:
  FILE: c
```

(continues on next page)

(continued from previous page)

```
  RUNNING: once
  SPLITS: 1
  DEPENDENCIES: B
RECOVER_B_2:
  FILE: fix_b
  RUNNING: once
  DEPENDENCIES:
    B:
      SPLIT_TO: "2"
      STATUS: "RUNNING"
```

Job frequency

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

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

---

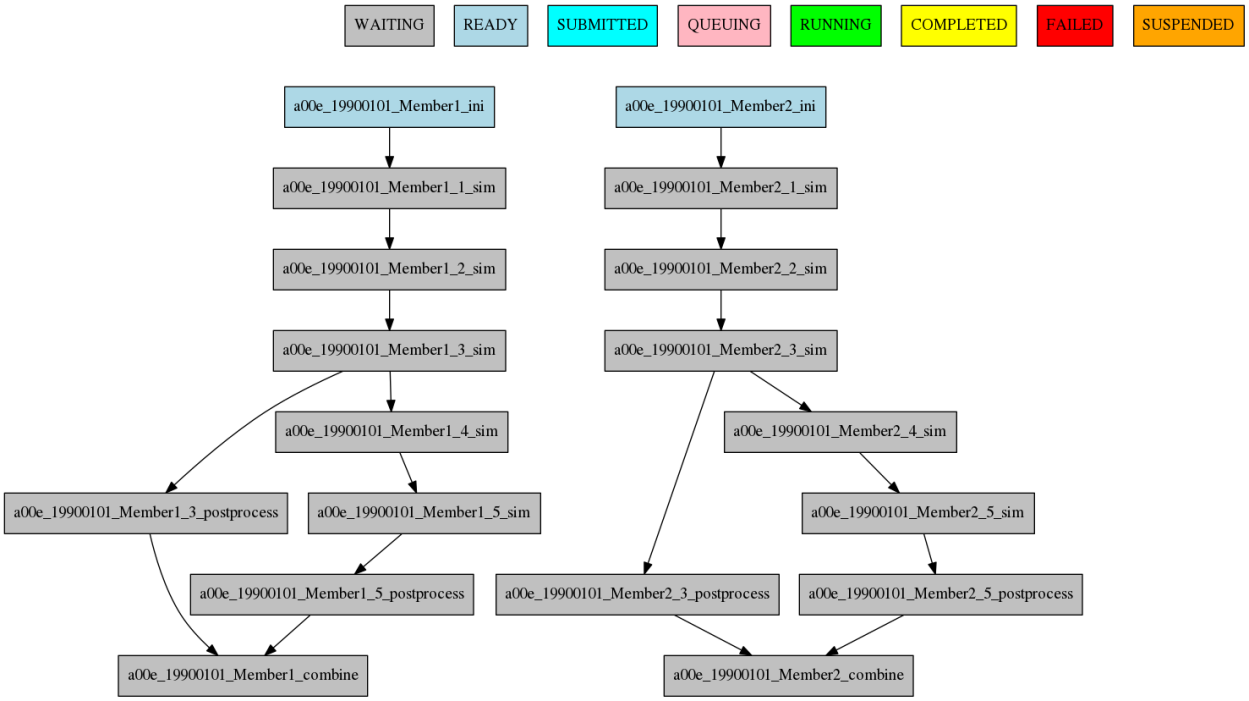
```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: chunk

postprocess:
    FILE: postprocess.sh
    DEPENDENCIES: sim
    RUNNING: chunk
    FREQUENCY: 3

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

The resulting workflow can be seen in Figure 3.5



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

Job synchronize

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

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

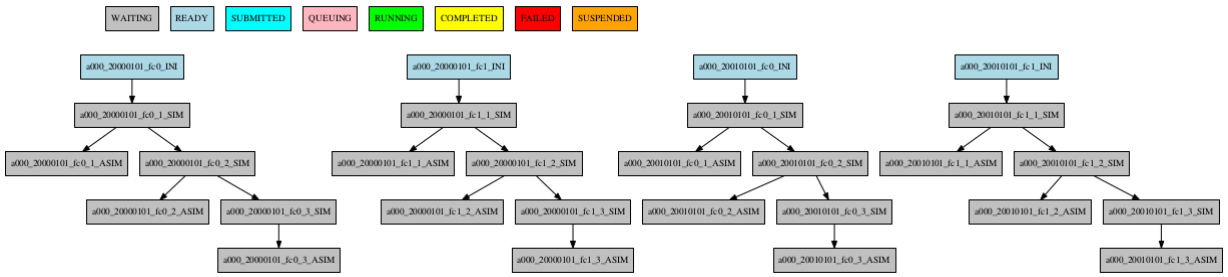
```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: INI SIM-1
    RUNNING: chunk

ASIM:
    FILE: asim.sh
    DEPENDENCIES: SIM
    RUNNING: chunk
```

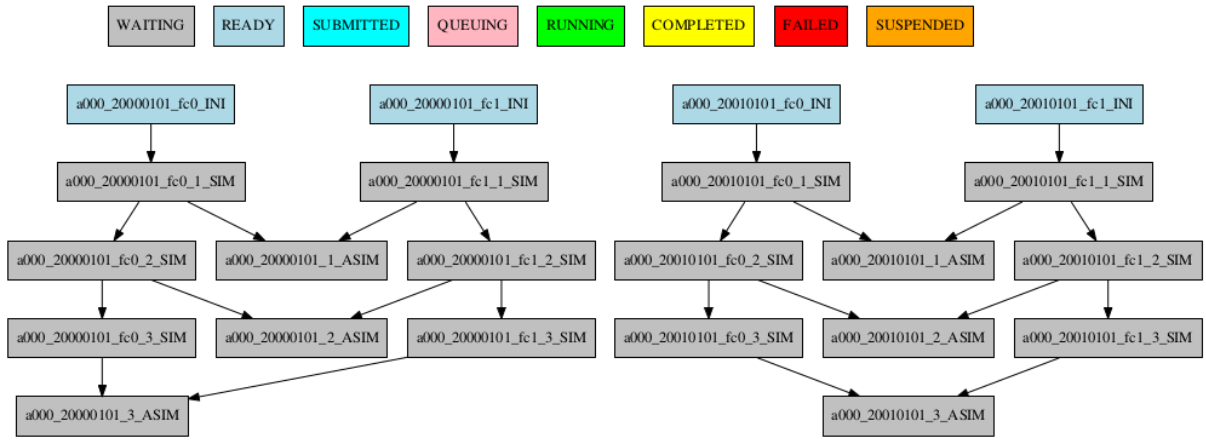
The resulting workflow can be seen in Figure 3.6

```
ASIM:
    SYNCHRONIZE: member
```



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

The resulting workflow of setting SYNCHRONIZE parameter to 'member' can be seen in Figure 3.7



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

```
ASIM:
    SYNCHRONIZE: date
```

The resulting workflow of setting SYNCHRONIZE parameter to 'date' can be seen in Figure 3.8

Job split

For jobs running at any level, it may be useful to split each task into different parts. This behaviour can be achieved using the SPLITS attribute to specify the number of parts.

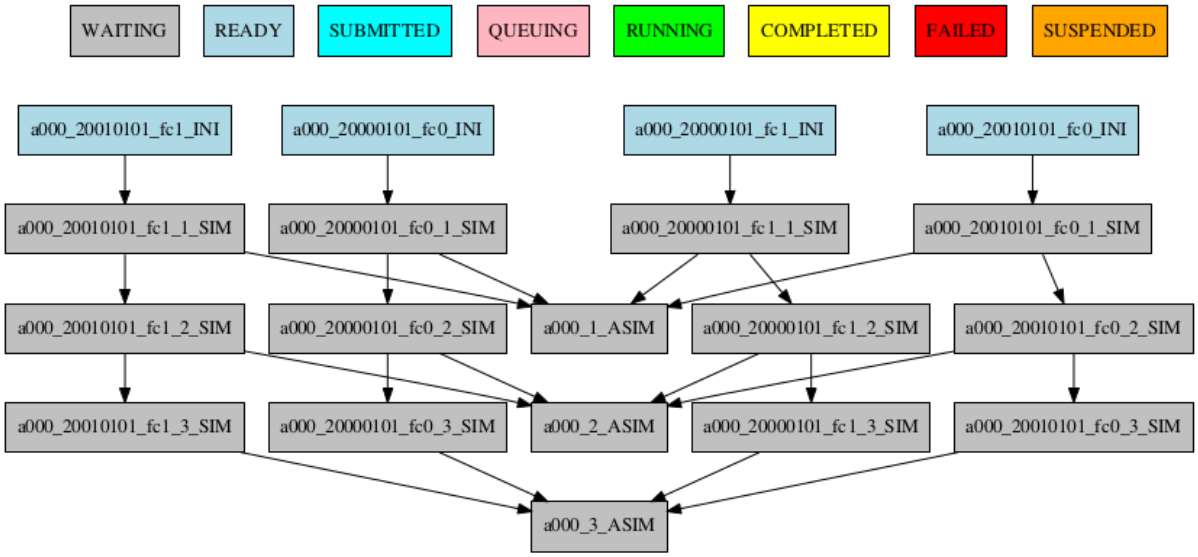
It is also possible to specify the splits for each task using the SPLITS_FROM and SPLITS_TO attributes.

There is also an special character '*' that can be used to specify that the split is 1-to-1 dependency. In order to use this character, you have to specify both SPLITS_FROM and SPLITS_TO attributes.

```
ini:
    FILE: ini.sh
    RUNNING: once

sim:
```

(continues on next page)



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

(continued from previous page)

```
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: once

asim:
    FILE: asim.sh
    DEPENDENCIES: sim
    RUNNING: once
    SPLITS: 3

post:
    FILE: post.sh
    RUNNING: once
    DEPENDENCIES:
        asim:
            SPLITS_FROM:
                2,3: # [2:3] is also valid
                    splits_to: 1,2*,3* # 1,[2:3]* is also valid, you can also specify␣
↪the step with [2:3:step]
    SPLITS: 3
```

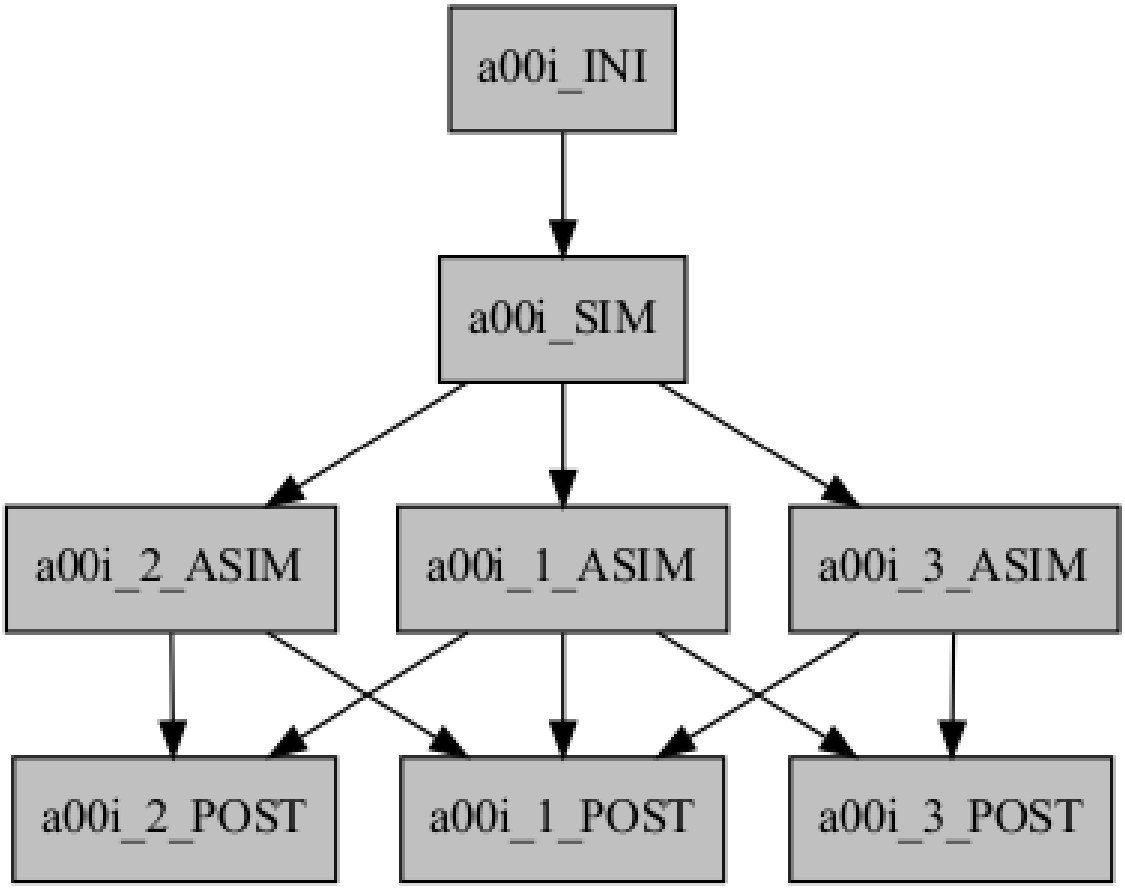
In this example:

Post job will be split into 2 parts. Each part will depend on the 1st part of the asim job. The 2nd part of the post job will depend on the 2nd part of the asim job. The 3rd part of the post job will depend on the 3rd part of the asim job.

Example2: N-to-1 dependency

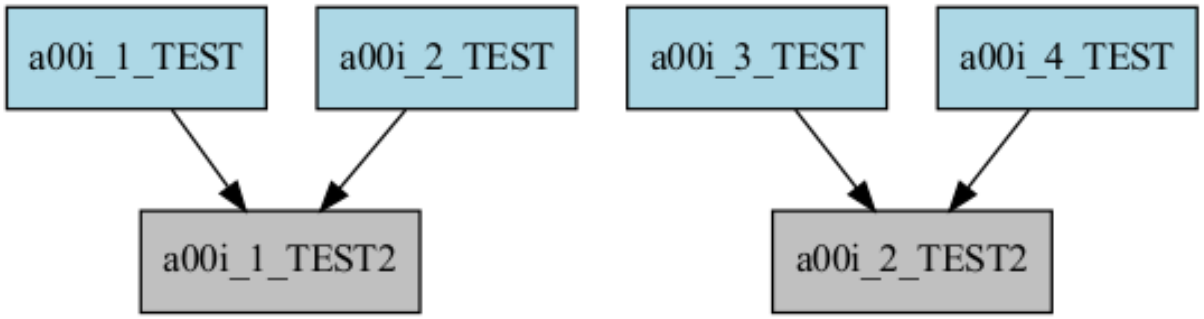
```
TEST:
  FILE: TEST.sh
  RUNNING: once
```

(continues on next page)


a00i_INI
a00i_SIM
a00i_2_ASIM
a00i_1_ASIM
a00i_3_ASIM
a00i_2_POST
a00i_1_POST
a00i_3_POST

(continued from previous page)

```
  SPLITS: '4'
TEST2:
  FILE: TEST2.sh
  DEPENDENCIES:
    TEST:
      SPLITS_FROM:
        "[1:2]":
          SPLITS_TO: "[1:4]*\\2"
  RUNNING: once
  SPLITS: '2'
```



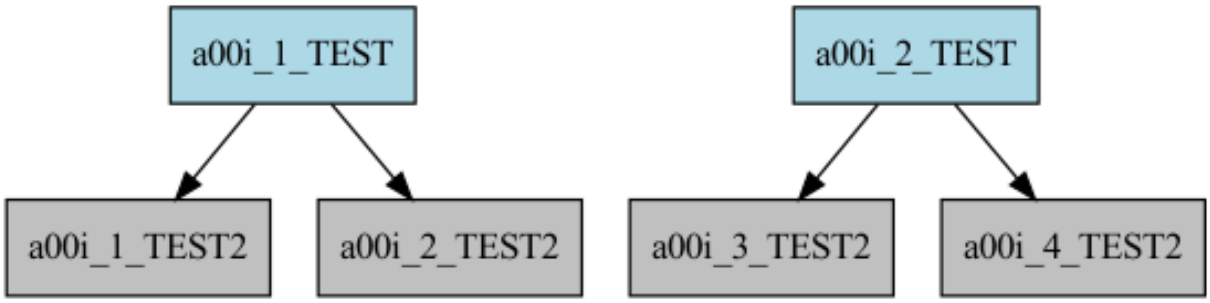
Example3: 1-to-N dependency

```
TEST:
  FILE: TEST.sh
  RUNNING: once
  SPLITS: '2'
TEST2:
  FILE: TEST2.sh
  DEPENDENCIES:
    TEST:
      SPLITS_FROM:
        "[1:4]":
          SPLITS_TO: "[1:2]*\\2"
  RUNNING: once
  SPLITS: '4'
```

Job delay

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

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




```
ini:
    FILE: ini.sh
    RUNNING: member

sim:
    FILE: sim.sh
    DEPENDENCIES: ini sim-1
    RUNNING: chunk

asim:
    FILE: asim.sh
    DEPENDENCIES:  sim asim-1
    RUNNING:  chunk
    DELAY:  2

post:
    FILE:  post.sh
    DEPENDENCIES:  sim asim
    RUNNING:  chunk
```

The resulting workflow can be seen in Figure 3.9

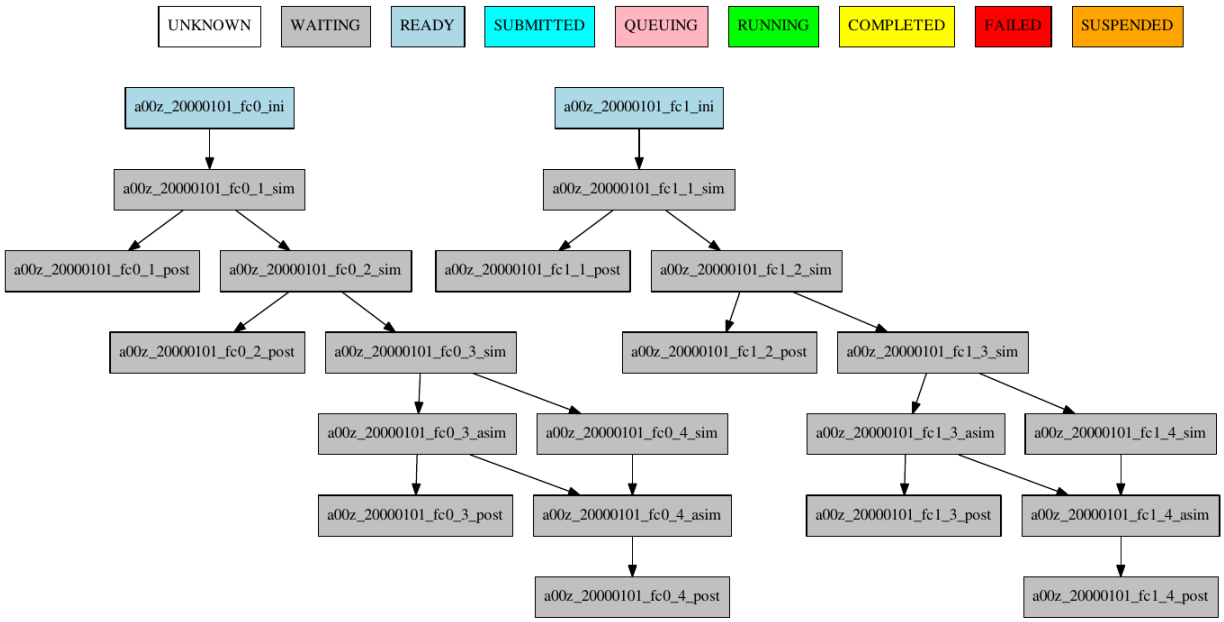
3.3.4 Workflow examples:

Example 1: How to select an specific chunk

Warning: This example illustrates the old select_chunk.

```
SIM:
    FILE: templates/sim.tmpl.sh
    DEPENDENCIES: INI SIM-1 POST-1 CLEAN-5
        INI:
        SIM-1:
        POST-1:
          CHUNKS_FROM:
```

(continues on next page)



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

(continued from previous page)

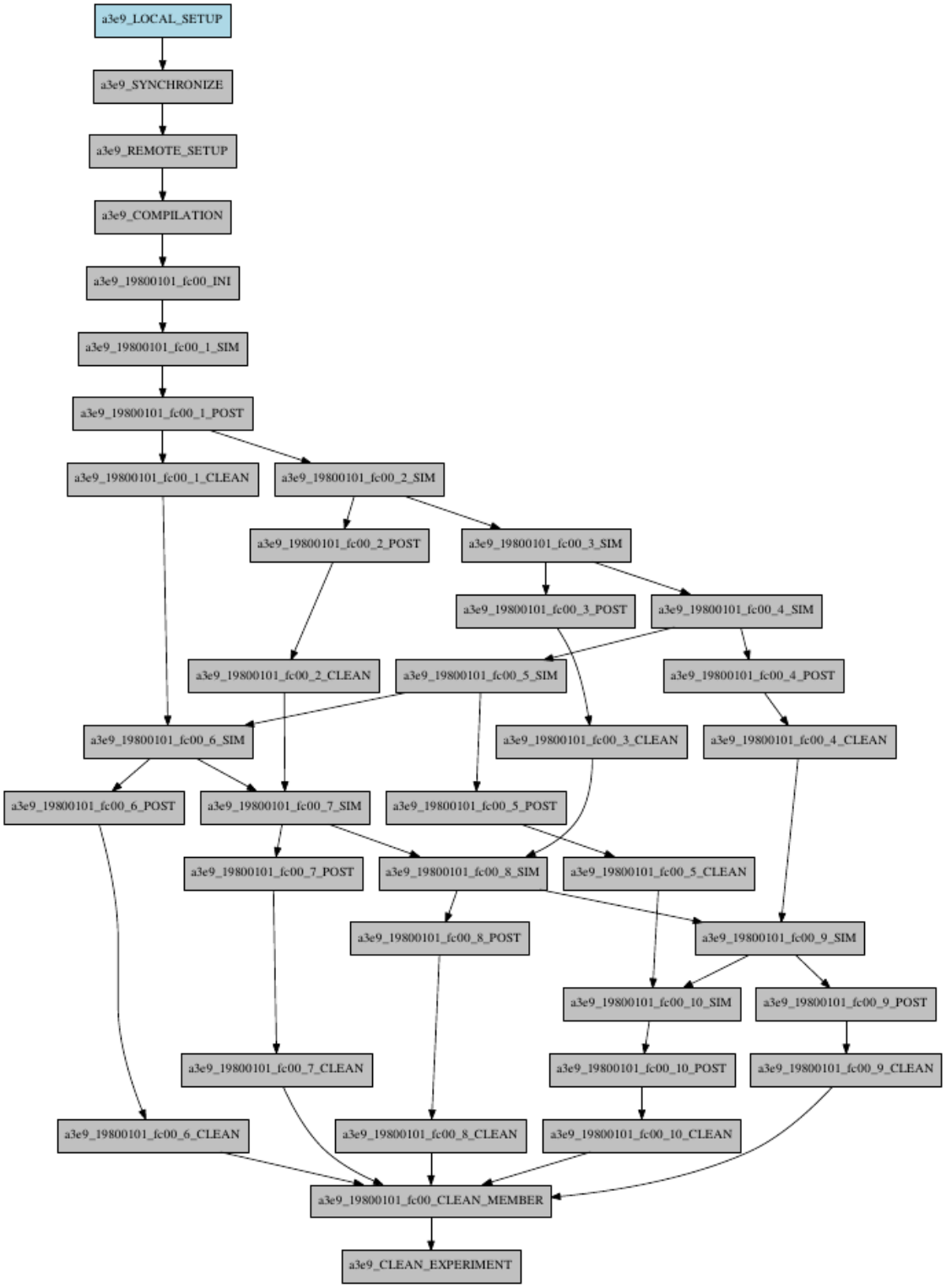
```
            all:
                chunks_to: 1
        CLEAN-5:
    RUNNING: chunk
    WALLCLOCK: 0:30
    PROCESSORS: 768
```

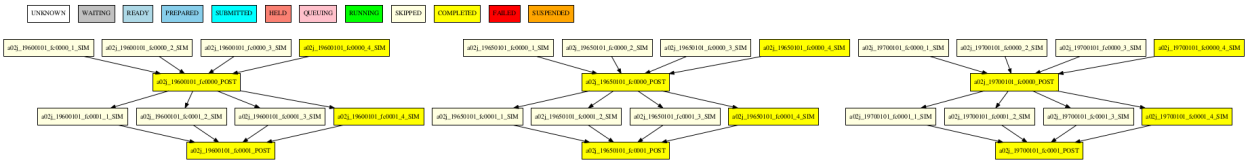
Example 2: SKIPPABLE

In this workflow you can see an illustrated example of SKIPPABLE parameter used in an dummy workflow.

```
JOBS:
    SIM:
        FILE: sim.sh
        DEPENDENCIES: INI POST-1
        WALLCLOCK: 00:15
        RUNNING: chunk
        QUEUE: debug
        SKIPPABLE: TRUE

    POST:
        FILE: post.sh
        DEPENDENCIES: SIM
        WALLCLOCK: 00:05
        RUNNING: member
        #QUEUE: debug
```


a3e9_LOCAL_SETUP
a3e9_SYNCHRONIZE
a3e9_REMOTE_SETUP
a3e9_COMPILATION
a3e9_19800101_fc00_INI
a3e9_19800101_fc00_1_SIM
a3e9_19800101_fc00_1_POST
a3e9_19800101_fc00_1_CLEAN
a3e9_19800101_fc00_2_SIM
a3e9_19800101_fc00_2_POST
a3e9_19800101_fc00_3_SIM
a3e9_19800101_fc00_3_POST
a3e9_19800101_fc00_4_SIM
a3e9_19800101_fc00_2_CLEAN
a3e9_19800101_fc00_5_SIM
a3e9_19800101_fc00_4_POST
a3e9_19800101_fc00_6_SIM
a3e9_19800101_fc00_3_CLEAN
a3e9_19800101_fc00_4_CLEAN
a3e9_19800101_fc00_6_POST
a3e9_19800101_fc00_7_SIM
a3e9_19800101_fc00_5_POST
a3e9_19800101_fc00_7_POST
a3e9_19800101_fc00_8_SIM
a3e9_19800101_fc00_5_CLEAN
a3e9_19800101_fc00_8_POST
a3e9_19800101_fc00_9_SIM
a3e9_19800101_fc00_10_SIM
a3e9_19800101_fc00_9_POST
a3e9_19800101_fc00_7_CLEAN
a3e9_19800101_fc00_10_POST
a3e9_19800101_fc00_9_CLEAN
a3e9_19800101_fc00_6_CLEAN
a3e9_19800101_fc00_8_CLEAN
a3e9_19800101_fc00_10_CLEAN
a3e9_19800101_fc00_CLEAN_MEMBER
a3e9_CLEAN_EXPERIMENT



Example 3: Weak dependencies

In this workflow you can see an illustrated example of weak dependencies.

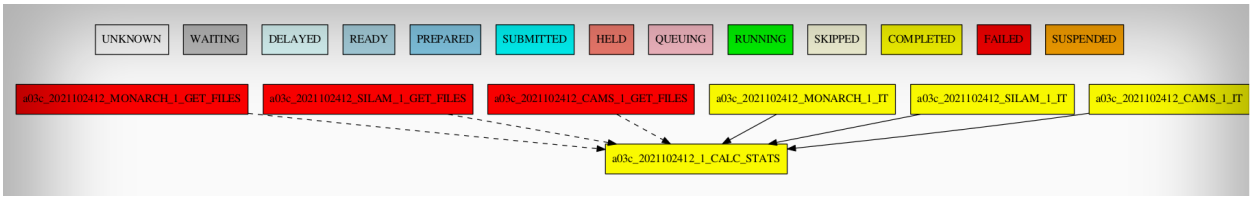
Weak dependencies, work like this way:

- X job only has one parent. X job parent can have "COMPLETED or FAILED" as status for current job to run.
- X job has more than one parent. One of the X job parent must have "COMPLETED" as status while the rest can be "FAILED or COMPLETED".

```
JOBS:
    GET_FILES:
        FILE: templates/fail.sh
        RUNNING: chunk

    IT:
        FILE: templates/work.sh
        RUNNING: chunk
        QUEUE: debug

    CALC_STATS:
        FILE: templates/work.sh
        DEPENDENCIES: IT GET_FILES?
        RUNNING: chunk
        SYNCHRONIZE: member
```



Example 4: Select Member

In this workflow you can see an illustrated example of select member. Using 4 members 1 datelist and 4 different job sections.

Expdef:

```
experiment:
    DATELIST: 19600101
    MEMBERS: "00 01 02 03"
    CHUNKSIZE: 1
    NUMCHUNKS: 2
```

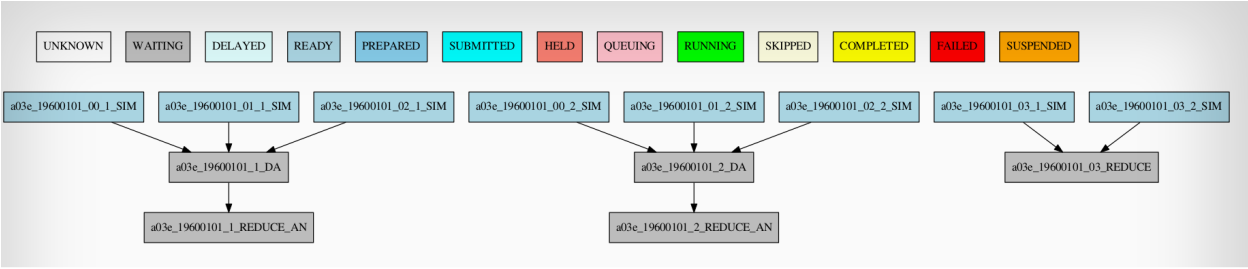
Jobs_conf:

```
JOBS:
    SIM:
        ...
        RUNNING: chunk
        QUEUE: debug

    DA:
        ...
        DEPENDENCIES:
            SIM:
                members_from:
                    all:
                        members_to: 00,01,02
        RUNNING: chunk
        SYNCHRONIZE: member

    REDUCE:
        ...
        DEPENDENCIES:
            SIM:
                members_from:
                    all:
                        members_to: 03
        RUNNING: member
        FREQUENCY: 4

    REDUCE_AN:
        ...
        FILE: templates/05b_sim.sh
        DEPENDENCIES: DA
        RUNNING: chunk
        SYNCHRONIZE: member
```



Loops definition

You need to use the FOR and NAME keys to define a loop.

To generate the following jobs:

```
experiment:
  DATELIST: 19600101
  MEMBERS: "00"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '2'
  CALENDAR: standard
JOBS:
  POST_20:

    DEPENDENCIES:
      POST_20:
      SIM_20:
    FILE: POST.sh
    PROCESSORS: '20'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  POST_40:

    DEPENDENCIES:
      POST_40:
      SIM_40:
    FILE: POST.sh
    PROCESSORS: '40'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  POST_80:

    DEPENDENCIES:
      POST_80:
      SIM_80:
    FILE: POST.sh
    PROCESSORS: '80'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  SIM_20:

    DEPENDENCIES:
      SIM_20-1:
    FILE: POST.sh
    PROCESSORS: '20'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  SIM_40:
```

(continues on next page)

(continued from previous page)

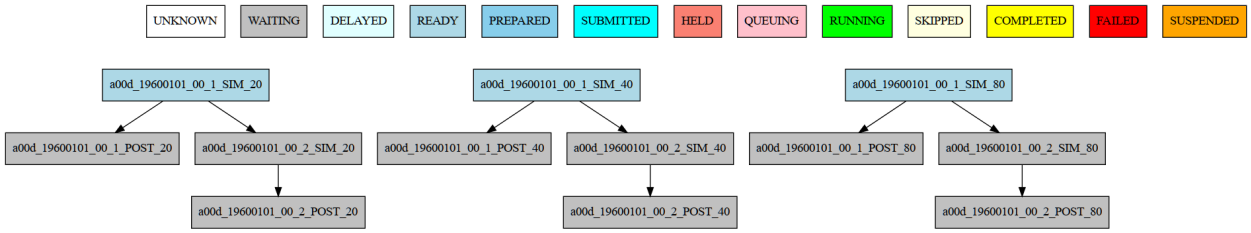
```
    DEPENDENCIES:
      SIM_40-1:
    FILE: POST.sh
    PROCESSORS: '40'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  SIM_80:

    DEPENDENCIES:
      SIM_80-1:
    FILE: POST.sh
    PROCESSORS: '80'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
```

One can use now the following configuration:

```
experiment:
  DATELIST: 19600101
  MEMBERS: "00"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '2'
  CALENDAR: standard
JOBS:
  SIM:
    FOR:
      NAME: [ 20,40,80 ]
      PROCESSORS: [ 20,40,80 ]
      THREADS: [ 1,1,1 ]
      DEPENDENCIES: [ SIM_20-1,SIM_40-1,SIM_80-1 ]
    FILE: POST.sh
    RUNNING: chunk
    WALLCLOCK: '00:05'
  POST:
      FOR:
        NAME: [ 20,40,80 ]
        PROCESSORS: [ 20,40,80 ]
        THREADS: [ 1,1,1 ]
        DEPENDENCIES: [ SIM_20 POST_20,SIM_40 POST_40,SIM_80 POST_80 ]
      FILE: POST.sh
      RUNNING: chunk
      WALLCLOCK: '00:05'
```

Warning: The mutable parameters must be inside the *FOR* key.



3.4 Wrappers

Job packages, or "wrappers", are jobs created as bundles of different tasks (submitted at once in a single script to the platform) assembled by Autosubmit to maximize the usage of platforms managed by a scheduler (by minimizing the queuing time between consecutive or concurrent tasks). Autosubmit supports four wrapper types that can be used depending on the experiment's workflow.

- *Horizontal*
- *Vertical*
- *Horizontal-vertical*
- *Vertical-horizontal*

Note: To have a preview of wrappers, you must use the parameter *-cw* available on inspect, monitor, and create.

```
autosubmit create  <expid>  -cw  # Unstarted experiment
autosubmit monitor <expid> -cw # Ongoing experiment
autosubmit inspect <expid> -cw -f # Visualize wrapper cmds
```

3.4.1 Basic configuration

To configure a new wrapper, the user has to define a *WRAPPERS* section in any configuration file. When using the standard configuration, this one is autosubmit.yml.

```
WRAPPERS:
    WRAPPER_0:
        TYPE: "horizontal"
```

By default, Autosubmit will try to bundle jobs of the same type. The user can alter this behavior by setting the *JOBS_IN_WRAPPER* parameter directive in the wrapper section.

When using multiple wrappers or 2-dim wrappers is essential to define the *JOBS_IN_WRAPPER* parameter.

```
WRAPPERS:
  WRAPPER_H:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
  WRAPPER_V:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM2"
  WRAPPER_VH:
    TYPE: "vertical-horizontal"
```

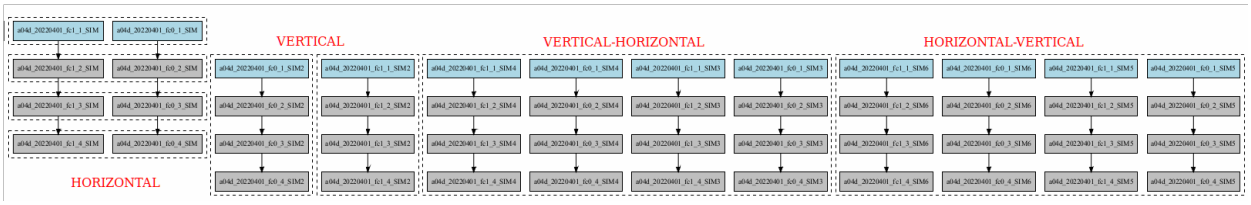
(continues on next page)

(continued from previous page)

```
    JOBS_IN_WRAPPER: "SIM3 SIM4"
  WRAPPER_HV:
    TYPE: "horizontal-vertical"
    JOBS_IN_WRAPPER: "SIM5 SIM6"

experiment:
  DATELIST: 20220101
  MEMBERS: "fc0 fc1"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '4'
  CALENDAR: standard
JOBS:
    SIM:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM-1
      WALLCLOCK: 00:15
    SIM2:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM2-1
      WALLCLOCK: 00:15
    SIM3:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM3-1
      WALLCLOCK: 00:15
    SIM4:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM4-1
      WALLCLOCK: 00:15
    SIM5:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM5-1
      WALLCLOCK: 00:15
    SIM6:
      FILE: sim.sh
      RUNNING: chunk
      QUEUE: debug
      DEPENDENCIES: SIM6-1
      WALLCLOCK: 00:15
```

Important: Autosubmit will not wrap tasks with external and non-fulfilled dependencies.



Wrapper parameters description

Type

The type parameter allow the user to determine the wrapper algorithm.

It affects tasks in wrapper order executions, and in hybrid cases, it adds some internal logic.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
```

Jobs_in_wrapper

The jobs_in_wrapper parameter allow the user to determine the tasks inside a wrapper by giving the job_section name. It can group multiple tasks by providing more than one job_section name.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
```

Method

The method parameter allow the user to determine if the wrapper will use machine files or threads.

This allows to form a wrapper with that relies on machinefiles to work.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    METHOD: ASTHREAD
```

or

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
```

This allows to form a wrapper with shared-memory paradigm instead of rely in machinefiles to work in parallel.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    METHOD: SRUN
```

Extend_wallclock

The extend_wallclock parameter allow the users to provide extra headroom for the wrapper. The accepted value is an integer. Autosubmit will translate this value automatically to the max_wallclock of the sum of wrapper inner-tasks wallclock at the horizontal level.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    extend_wallclock: 1
```

Retrials

The retrials parameter allows the users to enable or disable the wrapper's retrial mechanism. This value overrides the general tasks defined.

Vertical wrappers will retry the jobs without resubmitting the wrapper.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    RETRIALS: 2
```

Queue

The queue parameter allows the users to define a different queue for the wrapper. This value overrides the platform queue and job queue.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    QUEUE: BSC_ES
```

Export

The queue parameter allows the users to define a path to a script that will load environment scripts before running the wrapper tasks. This value overrides the job queue.

```
WRAPPERS:
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
    EXPORT: "%CURRENT_ROOTDIR%/envmodules.sh"
```

Check_time_wrapper

The CHECK_TIME_WRAPPER parameter defines the frequency, in seconds, on which Autosubmit will check the remote platform status of all the wrapper tasks. This affects all wrappers.

```
WRAPPERS:
  CHECK_TIME_WRAPPER: 10
  WRAPPER_0:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
  WRAPPER_1:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM1"
```

Number of jobs in a wrapper({MIN/MAX}_WRAPPED{_H/_V}

Users can configure the maximum and the minimum number of jobs in each wrapper by configuring MAX_WRAPPED and MIN_WRAPPED inside the wrapper section. If the user doesn't set them, Autosubmit will default to MAX_WRAPPED: "infinite" and MIN_WRAPPED: 2.

```
WRAPPERS:
  MIN_WRAPPED: 2
  MAX_WRAPPED: 999999
  WRAPPER_0:
    MAX_WRAPPED: 2
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
  WRAPPER_1:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM1"
```

For 2-dim wrappers, {MAX_MIN}_WRAPPED_{V/H} must be used instead of the general one.

```
WRAPPERS:
 MIN_WRAPPED: 2
 MAX_WRAPPED: 999999
 WRAPPER_0:
  MAX_WRAPPED_H: 2
  MAX_WRAPPED_V: 4
  MIN_WRAPPED_H: 2
```

(continues on next page)

(continued from previous page)

```
  MIN_WRAPPED_V: 2
  TYPE: "horizontal-vertical"
  JOBS_IN_WRAPPER: "SIM SIM1"
```

Policy

Autosubmit will wrap as many tasks as possible while respecting the limits set in the configuration(MAX_WRAPPED, MAX_WRAPPED_H, MAX_WRAPPED_V, MIN_WRAPPED, MIN_WRAPPED_V, and MIN_WRAPPED_H parameters). However, users have three different policies available to tune the behavior in situations where there aren't enough tasks in general, or there are uncompleted tasks remaining from a failed wrapper job:

- Flexible: if there aren't at least MIN_WRAPPED tasks to be grouped, Autosubmit will submit them as individual jobs.
- Mixed: will wait for MIN_WRAPPED jobs to be available to create a wrapper, except if one of the wrapped tasks had failed beforehand. In this case, Autosubmit will submit them individually.
- Strict: will always wait for MIN_WRAPPED tasks to be ready to create a wrapper.

```
WRAPPERS:
  POLICY: "flexible"
  WRAPPER_0:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM SIM1"
```

3.4.2 Vertical wrapper

Vertical wrappers are suited for sequential dependent jobs (e.x. chunks of SIM tasks that depend on the previous chunk). Defining the platform's *MAX_WALLCLOCK* is essential since the wrapper's total wallclock time will be the sum of each job and will be a limiting factor for the creation of the wrapper, which will not bundle more jobs than the ones fitting in the wallclock time.

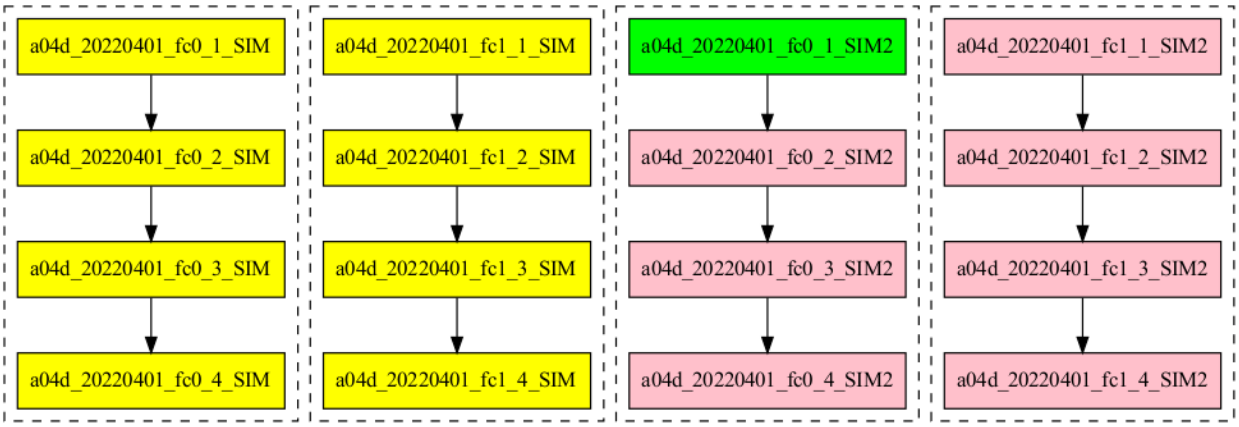
Autosubmit supports wrapping together vertically jobs of different types.

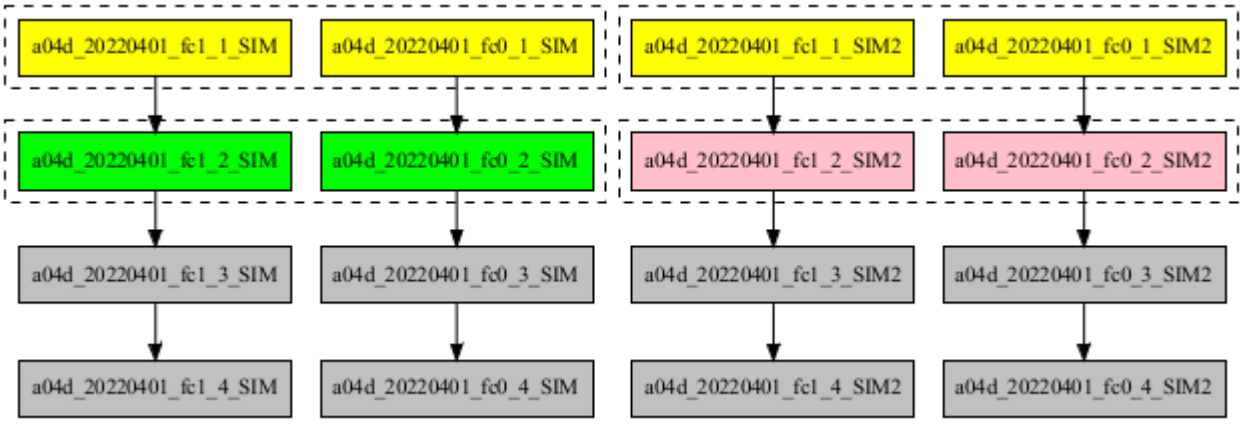
```
WRAPPERS:
  WRAPPER_V:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM"
```

3.4.3 Horizontal wrapper

Horizontal wrappers are suited for jobs that must run parallel (e.x. members of SIM tasks). Defining the platform's *MAX_PROCESSORS* is essential since the wrapper processor amount will be the sum of each job and will be a limiting factor for the creation of the wrapper, which will not bundle more jobs than the ones fitting in the *MAX_PROCESSORS* of the platform.

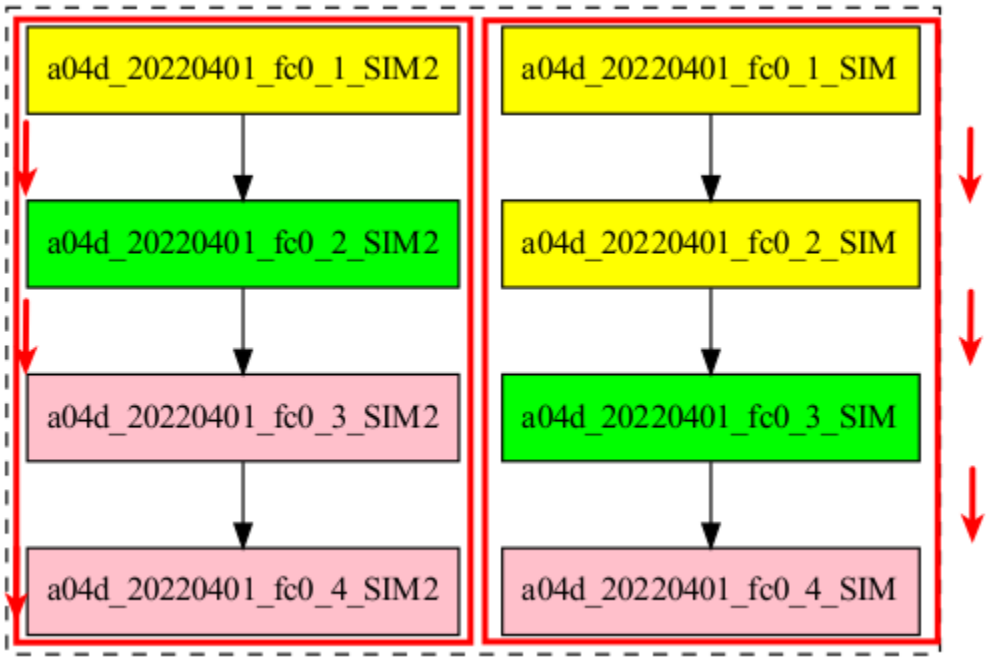
```
WRAPPERS:
  WRAPPER_H:
    TYPE: "horizontal"
    JOBS_IN_WRAPPER: "SIM"
```


a04d_20220401_fc0_1_SIM
a04d_20220401_fc0_2_SIM
a04d_20220401_fc0_3_SIM
a04d_20220401_fc0_4_SIM
a04d_20220401_fc1_1_SIM
a04d_20220401_fc1_2_SIM
a04d_20220401_fc1_3_SIM
a04d_20220401_fc1_4_SIM
a04d_20220401_fc0_1_SIM2
a04d_20220401_fc0_2_SIM2
a04d_20220401_fc0_3_SIM2
a04d_20220401_fc0_4_SIM2
a04d_20220401_fc1_1_SIM2
a04d_20220401_fc1_2_SIM2
a04d_20220401_fc1_3_SIM2
a04d_20220401_fc1_4_SIM2


a04d_20220401_fc1_1_SIM
a04d_20220401_fc0_1_SIM
a04d_20220401_fc1_1_SIM2
a04d_20220401_fc0_1_SIM2
a04d_20220401_fc1_2_SIM
a04d_20220401_fc0_2_SIM
a04d_20220401_fc1_2_SIM2
a04d_20220401_fc0_2_SIM2
a04d_20220401_fc1_3_SIM
a04d_20220401_fc0_3_SIM
a04d_20220401_fc1_3_SIM2
a04d_20220401_fc0_3_SIM2
a04d_20220401_fc1_4_SIM
a04d_20220401_fc0_4_SIM
a04d_20220401_fc1_4_SIM2
a04d_20220401_fc0_4_SIM2

3.4.4 Vertical-horizontal wrapper

The vertical-horizontal wrapper allows bundling together a vertical sequence of tasks independent of the horizontal ones. Therefore, all horizontal tasks do not need to finish to progress to the next horizontal level.



3.4.5 Horizontal-vertical wrapper

The horizontal-vertical wrapper allows bundling together tasks that could run simultaneously but need to communicate before progressing to the next horizontal level.

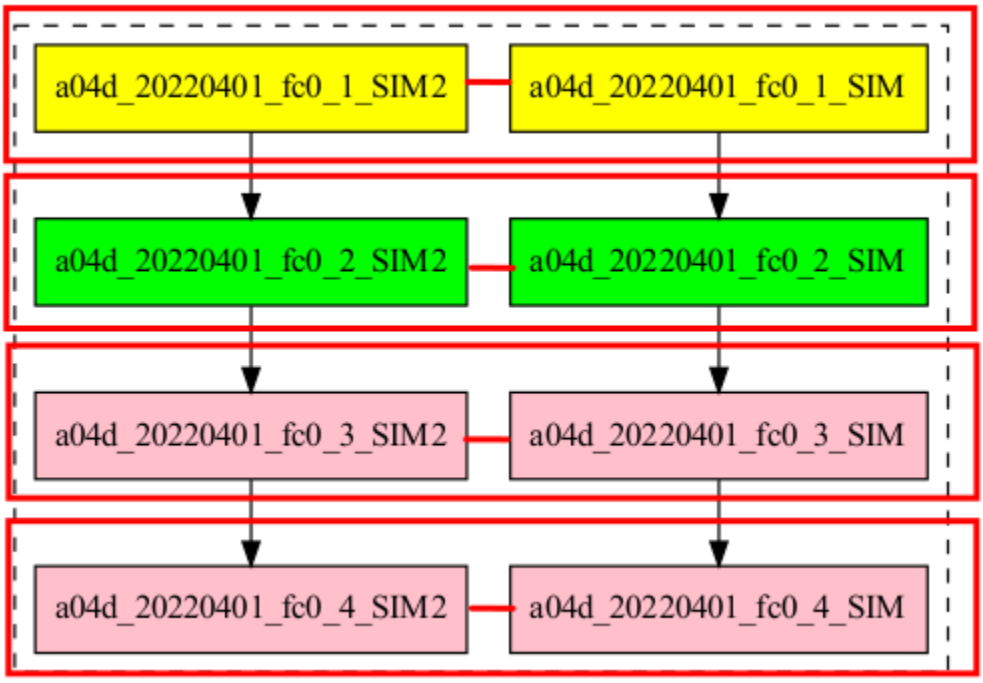
Advanced example: Set-up an crossdate wrapper

Considering the following configuration:

```
experiment:
  DATELIST: 20120101 20120201
  MEMBERS: "000 001"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '3'

JOBS:
  LOCAL_SETUP:
    FILE: templates/local_setup.sh
    PLATFORM: marenostrum_archive
    RUNNING: once
    NOTIFY_ON: COMPLETED
  LOCAL_SEND_SOURCE:
    FILE: templates/01_local_send_source.sh
```

(continues on next page)



(continued from previous page)

```
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
    NOTIFY_ON: FAILED
  LOCAL_SEND_STATIC:
    FILE: templates/01b_local_send_static.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
    NOTIFY_ON: FAILED
  REMOTE_COMPILE:
    FILE: templates/02_compile.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    PROCESSORS: '4'
    WALLCLOCK: 00:50
    NOTIFY_ON: COMPLETED
  SIM:
    FILE: templates/05b_sim.sh
    DEPENDENCIES:
      LOCAL_SEND_STATIC:
      REMOTE_COMPILE:
      SIM-1:
      DA-1:
    RUNNING: chunk
    PROCESSORS: '68'
    WALLCLOCK: 00:12
    NOTIFY_ON: FAILED
```

(continues on next page)

(continued from previous page)

```
  LOCAL_SEND_INITIAL_DA:
    FILE: templates/00b_local_send_initial_DA.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP LOCAL_SEND_INITIAL_DA-1
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
  COMPILE_DA:
    FILE: templates/02b_compile_da.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    WALLCLOCK: 00:20
    NOTIFY_ON: FAILED
  DA:
    FILE: templates/05c_da.sh
    DEPENDENCIES:
      SIM:
      LOCAL_SEND_INITIAL_DA:
        CHUNKS_TO: "all"
        DATES_TO: "all"
        MEMBERS_TO: "all"
      COMPILE_DA:
      DA:
        DATES_FROM:
          "20120201":
            CHUNKS_FROM:
              1:
                DATES_TO: "20120101"
                CHUNKS_TO: "1"
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
    WALLCLOCK: 00:12
    PROCESSORS: '256'
    NOTIFY_ON: FAILED
```

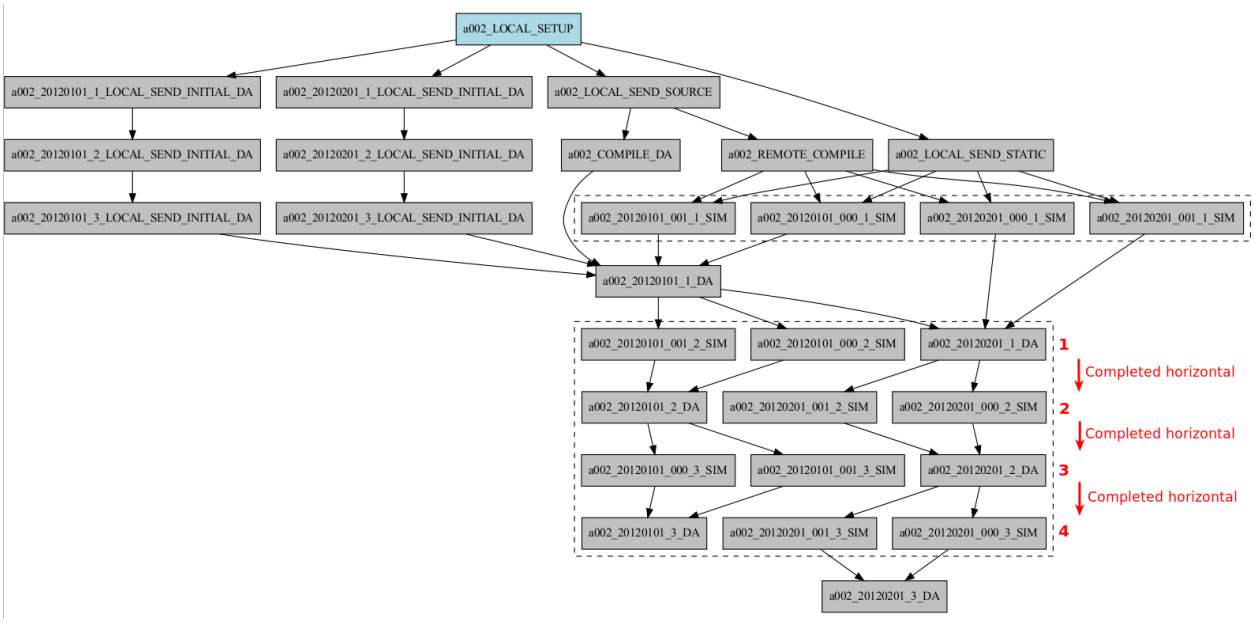
```
wrappers:
  wrapper_simda:
    TYPE: "horizontal-vertical"
    JOBS_IN_WRAPPER: "SIM DA"
```

3.5 Running Experiments

3.5.1 Run an experiment

Launch Autosubmit with the command:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run EXPID
```



In the previous command output `EXPID` is the experiment identifier. The command exits with `0` when the workflow finishes with no failed jobs, and with `1` otherwise.

Options:

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

  expid       experiment identifier
  -nt                   --notransitive
                            prevents doing the transitive reduction when plotting the␣
↪workflow
  -v                    --update_version
                            update the experiment version to match the actual autosubmit␣
↪version
  -st                   --start_time
                            Sets the starting time for the experiment. Accepted format:
↪'yyyy-mm-dd HH:MM:SS' or 'HH:MM:SS' (defaults to current day).
  -sa                   --start_after
                            Sets a experiment expid that will be tracked for completion.␣
↪When this experiment is completed, the current instance of Autosubmit run will start.
  -rom,--run_only_members  --run_members
                            Sets a list of members allowed to run. The list must have␣
↪the format '### ###' where '###' represents the name of the member as set in the conf␣
↪files.
  -h, --help  show this help message and exit
```

Example:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run cxxx
```

Important: If the autosubmit version is set on `autosubmit.yml` it must match the actual autosubmit version

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

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
nohup autosubmit run cxxx &
```

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

Important: Add encryption key to ssh agent for each session (if your ssh key is encrypted)

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

`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.

How to run an experiment that was created with another version

Important: First of all you have to stop your Autosubmit instance related with the experiment

Once you've already loaded / installed the Autosubmit version do you want:

```
autosubmit create $EXPID -np
autosubmit recovery $EXPID -s --all -f -np
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run $EXPID -v
or
autosubmit updateversion $EXPID
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run $EXPID -v
```

EXPID is the experiment identifier. The most common problem when you change your Autosubmit version is the apparition of several Python errors. This is due to how Autosubmit saves internally the data, which can be incompatible

between versions. The steps above represent the process to re-create (1) these internal data structures and to recover (2) the previous status of your experiment.

How to run an experiment that was created with version <= 4.0.0

Important: First of all you have to stop your Autosubmit instance related with the experiment.

Once you've already loaded / installed the Autosubmit version do you want:

```
autosubmit upgrade $expid
autosubmit create $EXPID -np
autosubmit recovery $EXPID -s --all -f -np
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run $EXPID -v
or
autosubmit updateversion $EXPID
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run $EXPID -v
```

EXPID is the experiment identifier. The most common problem when you upgrade an experiment with INI configuration to YAML is that some variables may be not automatically translated. Ensure that all your $EXPID/conf/*.yml files are correct and also revise the templates in $EXPID/proj/$proj_name.

How to run only selected members

To run only a subset of selected members you can execute the command:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run EXPID -rom MEMBERS
```

EXPID is the experiment identifier, the experiment you want to run.

MEMBERS is the selected subset of members. Format *"member1 member2 member2"*, example: *"fc0 fc1 fc2"*.

Then, your experiment will start running jobs belonging to those members only. If the experiment was previously running and autosubmit was stopped when some jobs belonging to other members (not the ones from your input) where running, those jobs will be tracked and finished in the new exclusive run.

Furthermore, if you wish to run a sequence of only members execution; then, instead of running *autosubmit run -rom "member_1"* ... *autosubmit run -rom "member_n"*, you can make a bash file with that sequence and run the bash file. Example:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run EXPID -rom MEMBER_1
autosubmit run EXPID -rom MEMBER_2
autosubmit run EXPID -rom MEMBER_3
...
autosubmit run EXPID -rom MEMBER_N
```

3.5.2 How to start an experiment at a given time

To start an experiment at a given time, use the command:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run EXPID -st INPUT
```

EXPID is the experiment identifier

***INPUT* is the time when your experiment will start. You can provide two formats:**

- *H:M:S*: For example *15:30:00* will start your experiment at 15:30 in the afternoon of the present day.
- *yyyy-mm-dd H:M:S*: For example *2021-02-15 15:30:00* will start your experiment at 15:30 in the afternoon on February 15th.

Then, your terminal will show a countdown for your experiment start.

This functionality can be used together with other options supplied by the *run* command.

The *-st* command has a long version *–start_time*.

3.5.3 How to start an experiment after another experiment is finished

To start an experiment after another experiment is finished, use the command:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
autosubmit run EXPID -sa EXPIDB
```

EXPID is the experiment identifier, the experiment you want to start.

EXPIDB is the experiment identifier of the experiment you are waiting for before your experiment starts.

Warning: Both experiments must be using Autosubmit version *3.13.0* or later.

Then, your terminal will show the current status of the experiment you are waiting for. The status format is *COMPLETED/QUEUING/RUNNING/SUSPENDED/FAILED*.

This functionality can be used together with other options supplied by the *run* command.

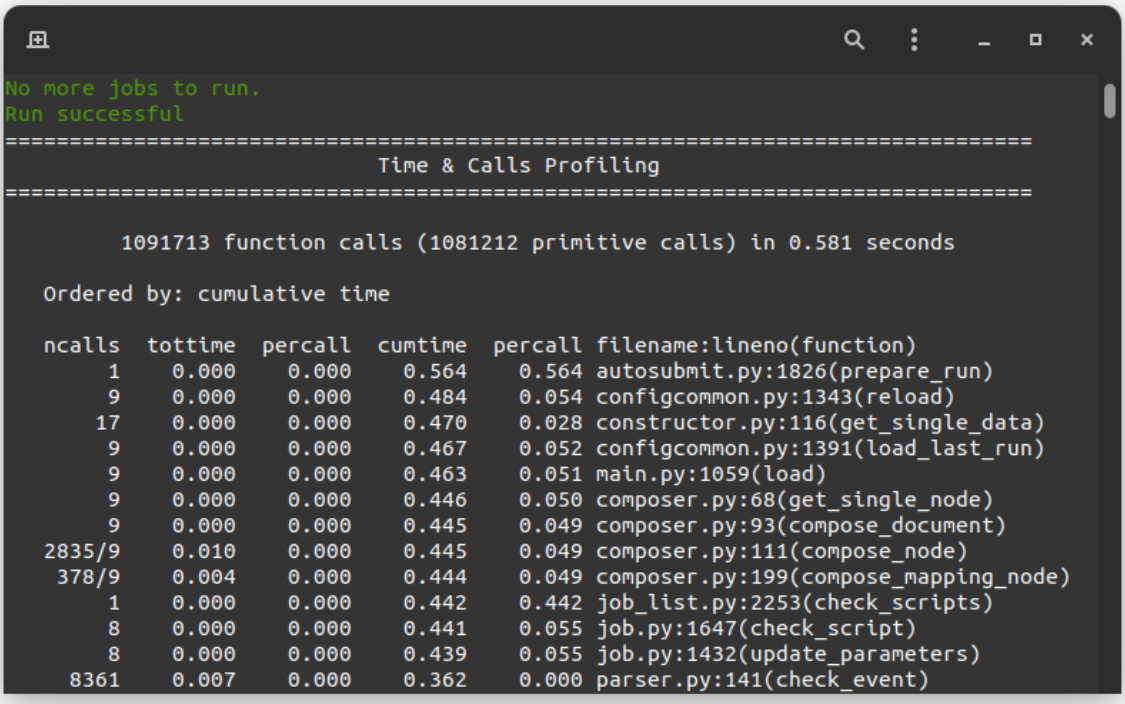
The *-sa* command has a long version *–start_after*.

3.5.4 How to profile Autosubmit while running an experiment

Autosubmit offers the possibility to profile an experiment execution. To enable the profiler, just add the `--profile` (or `-p`) flag to your `autosubmit run` command, as in the following example:

```
autosubmit run --profile EXPID
```

Note: Remember that the purpose of this profiler is to measure the performance of Autosubmit, not the jobs it runs.



```
No more jobs to run.
Run successful
================================================================================
                            Time & Calls Profiling
================================================================================

         1091713 function calls (1081212 primitive calls) in 0.581 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.564    0.564 autosubmit.py:1826(prepare_run)
        9    0.000    0.000    0.484    0.054 configcommon.py:1343(reload)
       17    0.000    0.000    0.470    0.028 constructor.py:116(get_single_data)
        9    0.000    0.000    0.467    0.052 configcommon.py:1391(load_last_run)
        9    0.000    0.000    0.463    0.051 main.py:1059(load)
        9    0.000    0.000    0.446    0.050 composer.py:68(get_single_node)
        9    0.000    0.000    0.445    0.049 composer.py:93(compose_document)
   2835/9    0.010    0.000    0.445    0.049 composer.py:111(compose_node)
    378/9    0.004    0.000    0.444    0.049 composer.py:199(compose_mapping_node)
        1    0.000    0.000    0.442    0.442 job_list.py:2253(check_scripts)
        8    0.000    0.000    0.441    0.055 job.py:1647(check_script)
        8    0.000    0.000    0.439    0.055 job.py:1432(update_parameters)
     8361    0.007    0.000    0.362    0.000 parser.py:141(check_event)
```

This profiler uses Python's `cProfile` and `psutil` modules to generate a report with simple CPU and memory metrics which will be displayed in your console after the command finishes, as in the example below:

The profiler output is also saved in `<EXPID>/tmp/profile`. There you will find two files, the report in plain text format and a `.prof` binary which contains the CPU metrics. We highly recommend using SnakeViz to visualize this file, as follows:

For more detailed documentation about the profiler, please visit this page.

3.5.5 How to prepare an experiment to run in two independent job_list. (Priority jobs, Two-step-run) (OLD METHOD)

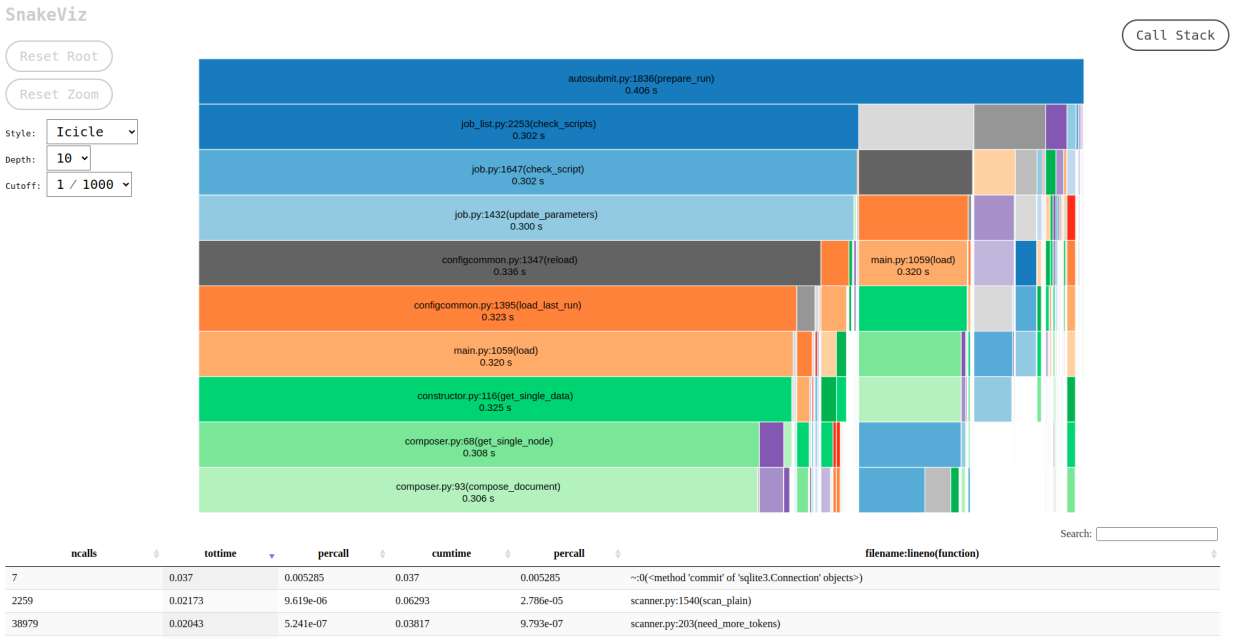
This feature allows to run an experiment in two separated steps without the need of do anything manually.

To achieve this, you will have to use an special parameter called TWO_STEP_START in which you will put the list of the jobs that you want to run in an exclusive mode. These jobs will run until all of them finishes and once it finishes, the rest of the jobs will begun the execution.

It can be activated through TWO_STEP_START and it is set on expdef_a02n.yml, under the experiment: section.

```
experiment:
    DATELIST: 20120101 20120201
    MEMBERS: fc00[0-3]
    CHUNKSIZEUNIT: day
    CHUNKSIZE: 1
    NUMCHUNKS: 10
    CHUNKINI :
```

(continues on next page)



(continued from previous page)

```
    CALENDAR: standard
    # To run before the rest of experiment:
    TWO_STEP_START: <job_names&section,dates,member_or_chunk(M/C),chunk_or_member(C/M)>
```

In order to be easier to use, there are Three modes for use this feature: job_names and section,dates,member_or_chunk(M/C),chunk_or_member(C/M).

- By using job_names alone, you will need to put all jobs names one by one divided by the char , .
- By using section,dates,member_or_chunk(M/C),chunk_or_member(C/M). You will be able to select multiple jobs at once combining these filters.
- Use both options, job_names and section,dates,member_or_chunk(M/C),chunk_or_member(C/M). You will have to put & between the two modes.

There are 5 fields on TWO_STEP_START, all of them are optional but there are certain limitations:

- **Job_name**: [Independent] List of job names, separated by ‘,’ char. Optional, doesn’t depend on any field. Separated from the rest of fields by ‘&’ must be the first field if specified
- **Section**: [Independent] List of sections, separated by ‘,’ char. Optional, can be used alone. Separated from the rest of fields by ‘;’
- **Dates**: [Depends on section] List of dates, separated by ‘,’ char. Optional, but depends on Section field. Separated from the rest of fields by ‘;’
- **member_or_chunk**: [Depends on Dates(OR)] List of chunk or member, must start with C or M to indicate the filter type. Jobs are selected by [1,2,3..] or by a range [0-9] Optional, but depends on Dates field. Separated from the rest of fields by ‘;’
- **chunk_or_member**: [Depends on Dates(OR)] List of member or chunk, must start with M or C to indicate the filter type. Jobs are selected by [1,2,3..] or by a range [0-9] Optional, but depends on Dates field. Separated from the rest of fields by ‘;’

Example using the old method

Guess the expdef configuration as follow:

```
experiment:
    DATELIST: 20120101
    MEMBERS: 00[0-1]
    CHUNKSIZEUNIT: day
    CHUNKSIZE: 1
    NUMCHUNKS: 2
    TWO_STEP_START: a02n_20120101_000_1_REDUCE&COMPILE_DA,SIM;20120101;c[1]
```

Given this job_list (jobs_conf has REMOTE_COMPILE(once),DA,SIM,REDUCE)

['a02n_REMOTE_COMPILE', 'a02n_20120101_000_1_SIM', 'a02n_20120101_000_2_SIM', 'a02n_20120101_001_1_SIM', 'a02n_20120101_001_2_SIM', 'a02n_COMPILE_DA', 'a02n_20120101_1_DA', 'a02n_20120101_2_DA', 'a02n_20120101_000_1_REDUCE', 'a02n_20120101_000_2_REDUCE', 'a02n_20120101_001_1_REDUCE', 'a02n_20120101_001_2_REDUCE']

The priority jobs will be (check TWO_STEP_START from expdef conf):

['a02n_20120101_000_1_SIM', 'a02n_20120101_001_1_SIM', 'a02n_COMPILE_DA', 'a02n_20120101_000_1_REDUCE']

3.5.6 How to prepare an experiment to run in two independent job_list. (New method)

From AS4, TWO_STEP_START is not longer needed since the users can now specify exactly which tasks of a job are needed to run the current task in the DEPENDENCIES parameter.

Simplified example using the new method

This example is based on the previous one, but using the new method and without the reduce job.

```
experiment:
    DATELIST: 20120101
    MEMBERS: "00[0-1]"
    CHUNKSIZEUNIT: day
    CHUNKSIZE: 1
    NUMCHUNKS: 2
JOBS:
    REMOTE_COMPILE:
        FILE: remote_compile.sh
        RUNNING: once
    DA:
        FILE: da.sh
        DEPENDENCIES:
            SIM:
            DA:
                DATES_FROM:
                 "20120201":
                   CHUNKS_FROM:
                     1:
```

(continues on next page)

(continued from previous page)

```
                    DATES_TO: "20120101"
                    CHUNKS_TO: "1"
    SIM:
        FILE: sim.sh
        DEPENDENCIES:
            LOCAL_SEND_STATIC:
            REMOTE_COMPILE:
            SIM-1:
            DA-1:
```

Example 2: Crossdate wrappers using the the new dependencies

```
experiment:
  DATELIST: 20120101 20120201
  MEMBERS: "000 001"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '3'
wrappers:
    wrapper_simda:
        TYPE: "horizontal-vertical"
        JOBS_IN_WRAPPER: "SIM DA"

JOBS:
  LOCAL_SETUP:
    FILE: templates/local_setup.sh
    PLATFORM: marenostrum_archive
    RUNNING: once
    NOTIFY_ON: COMPLETED
  LOCAL_SEND_SOURCE:
    FILE: templates/01_local_send_source.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
    NOTIFY_ON: FAILED
  LOCAL_SEND_STATIC:
    FILE: templates/01b_local_send_static.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
    NOTIFY_ON: FAILED
  REMOTE_COMPILE:
    FILE: templates/02_compile.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    PROCESSORS: '4'
    WALLCLOCK: 00:50
    NOTIFY_ON: COMPLETED
  SIM:
    FILE: templates/05b_sim.sh
```

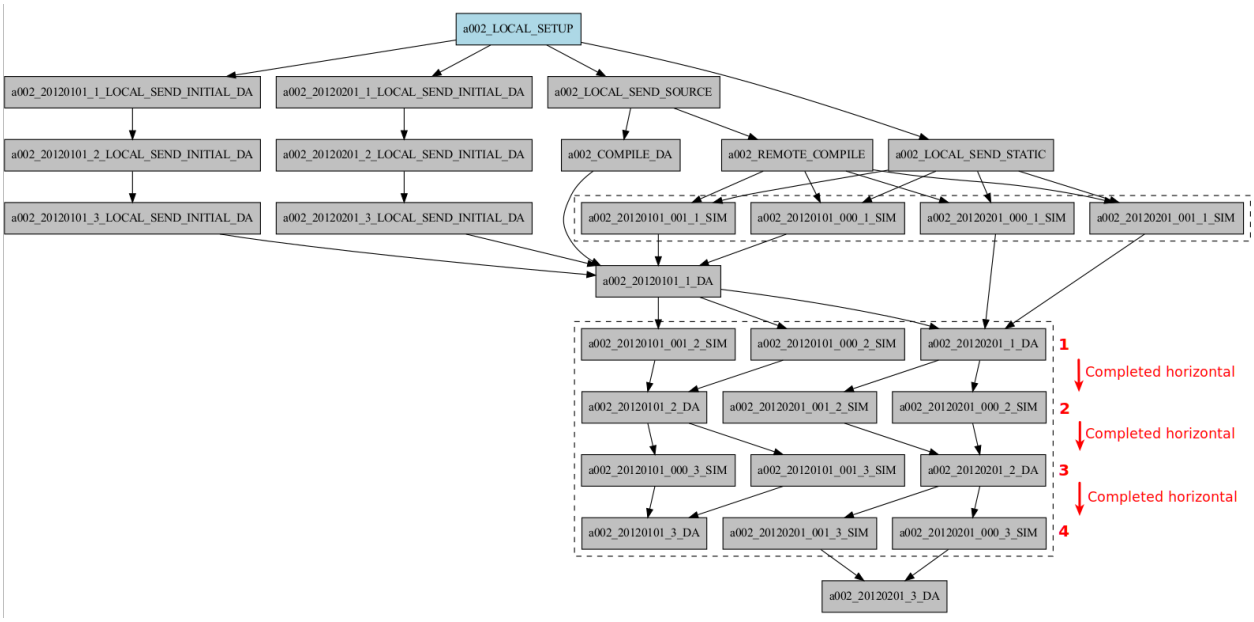
(continues on next page)

(continued from previous page)

```
    DEPENDENCIES:
      LOCAL_SEND_STATIC:
      REMOTE_COMPILE:
      SIM-1:
      DA-1:
    RUNNING: chunk
    PROCESSORS: '68'
    WALLCLOCK: 00:12
    NOTIFY_ON: FAILED
  LOCAL_SEND_INITIAL_DA:
    FILE: templates/00b_local_send_initial_DA.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP LOCAL_SEND_INITIAL_DA-1
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
  COMPILE_DA:
    FILE: templates/02b_compile_da.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    WALLCLOCK: 00:20
    NOTIFY_ON: FAILED
  DA:
    FILE: templates/05c_da.sh
    DEPENDENCIES:
      SIM:
      LOCAL_SEND_INITIAL_DA:
        CHUNKS_TO: "all"
        DATES_TO: "all"
        MEMBERS_TO: "all"
      COMPILE_DA:
      DA:
        DATES_FROM:
         "20120201":
           CHUNKS_FROM:
            1:
             DATES_TO: "20120101"
             CHUNKS_TO: "1"
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
    WALLCLOCK: 00:12
    PROCESSORS: '256'
    NOTIFY_ON: FAILED
```

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

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
nohup autosubmit run cxxx &
```



3.5.7 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
dbeltran  22835     1  1 May04 ?        00:45:35 autosubmit run cxxy
dbeltran  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.

3.6 How to restart the experiment

This procedure allows you to restart an experiment. Autosubmit looks for the COMPLETED file for jobs that are considered active (SUBMITTED, QUEUING, RUNNING), UNKNOWN or READY.

Warning: You can only restart the experiment if there are not active jobs. You can use -f flag to cancel running jobs automatically.

You must execute:

```
autosubmit recovery EXPID
```

EXPID is the experiment identifier.

Options:

```
usage: autosubmit recovery [-h] [-np] [--all] [-s] [-group_by {date,member,chunk,split} -
↪expand -expand_status] expid

    expid       experiment identifier

    -h, --help  show this help message and exit
    -np, --noplot  omit plot
    -f             Allows to perform the recovery even if there are active jobs
    --all        Get all completed files to synchronize pkl
    -s, --save  Save changes to disk
    -group_by {date,member,chunk,split,automatic}
                         criteria to use for grouping jobs
    -expand,               list of dates/members/chunks to expand
    -expand_status,        status(es) to expand
    -nt                    --notransitive
                                 prevents doing the transitive reduction when␣
↪plotting the workflow
    -nl                    --no_recover_logs
                                 prevents the recovering of log files from remote␣
↪platforms
    -d                     --detail
                             Shows Job List view in terminal
```

Example:

```
autosubmit recovery cxxx -s
```

In order to understand more the grouping options, which are used for visualization purposes, please check *Grouping jobs*.

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
```

3.6.1 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>.yml` the `create` command will generate the rerun if the variable RERUN is set to TRUE and a RERUN_JOBLIST is provided.

Additionally, you can have re-run only jobs that won't be include in the default job_list. In order to do that, you have to set RERUN_ONLY in the jobs conf of the corresponding job.

```
autosubmit create cxxx
```

It will read the list of jobs specified in the RERUN_JOBLIST and will generate a new plot.

Example:

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

```
...

rerun:
    RERUN: TRUE
    RERUN_JOBLIST: RERUN_TEST_INI;SIM[19600101[C:3]],RERUN_TEST_INI_chunks[19600101[C:3]]
...

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

```
PREPROCVAR:
    FILE: templates/04_preproc_var.sh
    RUNNING: chunk
    PROCESSORS: 8

RERUN_TEST_INI_chunks:
    FILE: templates/05b_sim.sh
    RUNNING: chunk
    RERUN_ONLY: true

RERUN_TEST_INI:
    FILE: templates/05b_sim.sh
    RUNNING: once
    RERUN_ONLY: true

SIM:
    DEPENDENCIES: RERUN_TEST_INI RERUN_TEST_INI_chunks PREPROCVAR SIM-1
    RUNNING: chunk
    PROCESSORS: 10

.. figure:: fig/rerun.png
   :name: rerun_result
   :align: center
   :alt: rerun_result
```

Run the command:

```
# Add your key to ssh agent ( if encrypted )
ssh-add ~/.ssh/id_rsa
nohup autosubmit run cxxx &
```

3.7 Manage Experiments

3.7.1 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 particularly 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.yml` file.

3.7.2 How to archive an experiment

When you archive an experiment in Autosubmit, it automatically *cleans* the experiment as well. This means the experiment will not be available for use, unless it is unarchived.

```
autosubmit archive <EXPID>
```

Options:

3.1: `autosubmit archive` options

```
$ autosubmit archive -h

[Errno 2] No such file or directory: 'source/'
```

The archived experiment will be stored as a `tar.gz` file, under a directory named after the year of the last ``_COMPLETED` file date or, if no _COMPLETED job is present, it will use the year of the date the `autosubmit archive` was run (e.g. for the selected year `2023`, the location will be `$HOME/autosubmit/2023/<EXPID>.tar.gz`).

3.7.3 How to unarchive an experiment

To unarchive an experiment, use the command:

```
autosubmit unarchive <EXPID>
```

Options:

3.2: `autosubmit unarchive` options

```
$ autosubmit unarchive -h

[Errno 2] No such file or directory: 'source/'
```

3.7.4 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.

3.7.5 How to migrate an experiment

To migrate an experiment from one user to another, you need to add two parameters for each platform in the platforms configuration file:

- USER_TO: <target_user> # Mandatory
- TEMP_DIR: <hpc_temporary_directory> # Mandatory, can be left empty if there are no files on that platform
- SAME_USER: false|true # Default False
- PROJECT_TO: <project> # Optional, if not specified project will remain the same
- HOST_TO: <cluster_ip> # Optional, avoid alias if possible, try use direct ip.

Warning: The USER_TO must be a different user , in case you want to maintain the same user, put SAME_USER: True.

Warning: The temporary directory must be readable by both users (old owner and new owner) Example for a RES account to BSC account the tmp folder must have rwx|rwx|— permissions. The temporary directory must be in the same filesystem.

User A, To offer the experiment:

```
autosubmit migrate --offer expid
```

Local files will be archived and remote files put in the HPC temporary directory.

User A To only offer the remote files

```
autosubmit migrate expid --offer --onlyremote
```

Only remote files will be put in the HPC temporary directory.

Warning: Be sure that there is no folder named as the expid before do the pick. The old owner might need to remove temporal files and archive. To Run the experiment the queue may need to be change.

Warning: If onlyremote option is selected, the pickup must maintain the flag otherwise the command will fail.

Now to pick the experiment, the user B, must do

```
autosubmit migrate --pickup expid
```

Local files will be unarchived and remote files copied from the temporal location.

To only pick the remote files, the user B, must do

```
autosubmit migrate --pickup expid --onlyremote
```

3.7.6 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 (<expid>/proj 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
```

3.7.7 How to update the description of your experiment

Use the command:

```
autosubmit updatedescrip EXPID DESCRIPTION
```

EXPID is the experiment identifier.

DESCRIPTION is the new description of your experiment.

Autosubmit will validate the provided data and print the results in the command line.

Example:

```
autosubmit a29z "Updated using Autosubmit updatedescrip"
```

3.7.8 How to change the job status

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] [-np] [-s] [-t] [-o {pdf,png,ps,svg}] [-fl] [-fc] [-fs]␣
↪[-ft] [-group_by {date,member,chunk,split} -expand -expand_status] [-cw] 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
    -np, --noplot           omit plot
    -s, --save              Save changes to disk
    -t, --status_final    Target status
    -fl FILTER_LIST, --list
                          List of job names to be changed
    -fc FILTER_CHUNK, --filter_chunk
                          List of chunks to be changed
    -fs FILTER_STATUS, --filter_status
                          List of status to be changed
    -ft FILTER_TYPE, --filter_type
                          List of types to be changed
    -ftc FILTER_TYPE_CHUNK --filter_type_chunk
                          Accepts a string with the formula: "[ 19601101 [ fc0 [1 2 3 4]␣
↪Any [1] ] 19651101 [ fc0 [16 30] ] ],SIM,SIM2"
                          Where SIM, SIM2 are section (job types) names that also accept␣
↪the keyword "Any" so the changes apply to all sections.
                          Starting Date (19601101) does not accept the keyword "Any", so␣
↪you must specify the starting dates to be changed.
                          You can also specify date ranges to apply the change to a range␣
↪on dates.
                          Member names (fc0) accept the keyword "Any", so the chunks ([1 2␣
↪3 4]) given will be updated for all members.
                          Chunks must be in the format "[1 2 3 4]" where "1 2 3 4"␣
↪represent the numbers of the chunks in the member,
                          no range format is allowed.
    -d                      When using the option -ftc and sending this flag, a tree view␣
↪of the experiment with markers indicating which jobs
                          have been changed will be generated.
    --hide,                 hide the plot
```

(continues on next page)

(continued from previous page)

```
    -group_by {date,member,chunk,split,automatic}
                        criteria to use for grouping jobs
    -expand,              list of dates/members/chunks to expand
    -expand_status,       status(es) to expand
    -nt                   --notransitive
                            prevents doing the transitive reduction when plotting the␣
↪workflow
    -cw                   --check_wrapper
                            Generate the wrapper in the current workflow
```

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
autosubmit setstatus cxxx -ftc "[ 19601101 [ fc1 [1] ] ], SIM" -t SUSPENDED -s
```

Date (month) range example:

```
autosubmit setstatus cxxx -ftc "[ 1960(1101-1201) [ fc1 [1] ] ], SIM" -t SUSPENDED -s
```

This example will result changing the following jobs:

```
cxxx_19601101_fc1_1_SIM
cxxx_19601201_fc1_1_SIM
```

Date (day) range example:

```
autosubmit setstatus cxxx -ftc "[ 1960(1101-1105) [ fc1 [1] ] ], SIM" -t SUSPENDED -s
```

Result:

```
cxxx_19601101_fc1_1_SIM
cxxx_19601102_fc1_1_SIM
cxxx_19601103_fc1_1_SIM
cxxx_19601104_fc1_1_SIM
cxxx_19601105_fc1_1_SIM
```

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 job names 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 the following status for job:**

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

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

The variable -ftc acts similar to -fc but also accepts the job types. It does not accept chunk ranges e.g. "1-10", but accepts the wildcard "Any" for members and job types. Let's look at some examples.

- **Using -ftc to change the chunks "1 2 3 4" of member "fc0" and chunk "1" of member "fc1" for the starting date "19601101", where these changes apply only for the "SIM" jobs:**

```
[ 19601101 [ fc0 [1 2 3 4] fc1 [1] ] ],SIM
```

- **Using -ftc to change the chunks "1 2 3 4" of all members for the starting date "19601101", where these changes apply only for the "SIM" jobs:**

```
[ 19601101 [ Any [1 2 3 4] ] ],SIM
```

- **Using -ftc to change the chunks "1 2 3 4" of "fc0" members for the starting date "19601101", where these changes apply to all jobs:**

```
[ 19601101 [ fc0 [1 2 3 4] ] ],Any
```

Try the combinations you come up with. Autosubmit will supply with proper feedback when a wrong combination is supplied.

Hint: When we are satisfied with the results we can use the parameter -s, which will save the change to the pkl file. In order to understand more the grouping options, which are used for visualization purposes, please check *Grouping jobs*.

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 adobe 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 an rename it when you have finished.

3.8 Monitor and Check Experiments

3.8.1 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. It is used to check if the script is well-formed. If any template has an inconsistency it will replace them for an empty value on the cmd generated. Options:

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

  expid                 experiment identifier
  -nt                   --notransitive
                            prevents doing the transitive reduction when plotting the␣
↪workflow
  -h, --help            show this help message and exit
```

Example:

```
autosubmit check cxxx
```

How to use check in running time:

In `jobs_cxxx.yml` , you can set check(default true) to check the scripts during autosubmit run cxx.

There are two parameters related to check:

- CHECK: Controls the mechanism that allows replacing an unused variable with an empty string (%_% substitution). It is TRUE by default.
- SHOW_CHECK_WARNINGS: For debugging purposes. It will print a lot of information regarding variables and substitution if it is set to TRUE.

```
CHECK: TRUE or FALSE or ON_SUBMISSION # Default is TRUE
SHOW_CHECK_WARNINGS: TRUE or FALSE # Default is FALSE
```

```
CHECK: TRUE # Static templates (existing on `autosubmit create`). Used to substitute␣
↪empty variables

CHECK: ON_SUBMISSION # Dynamic templates (generated on running time). Used to substitute␣
↪empty variables.

CHECK: FALSE # Used to disable this substitution.
```

```
SHOW_CHECK_WARNINGS: TRUE # Shows a LOT of information. Disabled by default.
```

For example:

```
LOCAL_SETUP:
    FILE: filepath_that_exists
    PLATFORM: LOCAL
    WALLCLOCK: 05:00
    CHECK: TRUE
    SHOW_CHECK_WARNINGS: TRUE
    ...
SIM:
    FILE: filepath_that_no_exists_until_setup_is_processed
    PLATFORM: bsc_es
    DEPENDENCIES: LOCAL_SETUP SIM-1
    RUNNING: chunk
    WALLCLOCK: 05:00
    CHECK: ON_SUBMISSION
    SHOW_CHECK_WARNINGS: FALSE
    ...
```

3.8.2 How to generate cmd files

To generate the cmd files of the current non-active jobs experiment, it is possible to use the command:

```
autosubmit inspect EXPID
```

EXPID is the experiment identifier.

Usage

Options:

```
usage: autosubmit inspect [-h]  [-fl] [-fc] [-fs] [-ft]  [-cw] expid

  expid                 experiment identifier

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

  -fl FILTER_LIST, --list
```

(continues on next page)

(continued from previous page)

```
                        List of job names to be generated
  -fc FILTER_CHUNK, --filter_chunk
                        List of chunks to be generated
  -fs FILTER_STATUS, --filter_status
                        List of status to be generated
  -ft FILTER_TYPE, --filter_type
                        List of types to be generated

  -cw                   --checkwrapper
                            Generate the wrapper cmd with the current filtered jobs

  -f                    --force
                            Generate all cmd files
```

Example

with autosubmit.lock present or not:

```
autosubmit inspect expid
```

with autosubmit.lock present or not:

```
autosubmit inspect expid -f
```

without autosubmit.lock:

```
autosubmit inspect expid -fl [-fc,-fs or ft]
```

To generate cmd for wrappers:

```
autosubmit inspect expid -cw -f
```

With autosubmit.lock and no (-f) force, it will only generate all files that are not submitted.

Without autosubmit.lock, it will generate all unless filtered by -fl,fc,fs or ft.

To generate cmd only for a single job of the section :

```
autosubmit inspect expid -q
```

3.8.3 How to monitor an 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,txt}] [-group_by {date,member,chunk,
↪split} -expand -expand_status] [-fl] [-fc] [-fs] [-ft] [-cw] expid [-txt] [-txtlog]

  expid                 Experiment identifier.

  -h, --help            Show this help message and exit.
  -o {pdf,png,ps,svg}, --output {pdf,png,ps,svg,txt}
                        Type of output for generated plot (or text file).
  -group_by {date,member,chunk,split,automatic}
                        Criteria to use for grouping jobs.
  -expand,              List of dates/members/chunks to expand.
  -expand_status,       Status(es) to expand.
  -fl FILTER_LIST, --list
                        List of job names to be filtered.
  -fc FILTER_CHUNK, --filter_chunk
                        List of chunks to be filtered.
  -fs FILTER_STATUS, --filter_status
                        Status to be filtered.
  -ft FILTER_TYPE, --filter_type
                        Type to be filtered.
  --hide,               Hide the plot.
  -txt, --text
                        Generates a tree view format that includes job name, children␣
↪number, and status in a file in the /status/ folder. If possible, shows the results in␣
↪the terminal.
  -txtlog, --txt_logfiles
                        Generates a list of job names, status, .out path, and .err path␣
↪as a file in /status/ (AS <3.12 behaviour).
  -nt                   --notransitive
                            Prevents doing the transitive reduction when plotting the␣
↪workflow.
  -cw                   --check_wrapper
                            Generate the wrapper in the current workflow.
```

Example:

```
autosubmit monitor cxxx
```

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

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

The location where the user can find the txt output containing the status of each job and the path to out and err log files.

```
<experiments_directory>/cxxx/status/cxxx_<date>_<time>.txt
```

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.

In order to understand more the grouping options, please check *Grouping jobs*.

3.8.4 Grouping jobs

Other than the filters, another option for large workflows is to group jobs. This option is available with the `group_by` keyword, which can receive the values `{date,member,chunk,split,automatic}`.

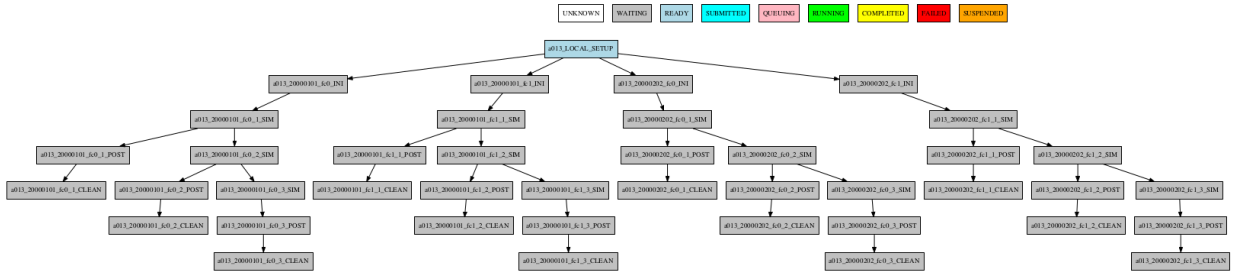
For the first 4 options, the grouping criteria is explicitly defined `{date,member,chunk,split}`. In addition to that, it is possible to expand some dates/members/chunks that would be grouped either/both by status or/and by specifying the date/member/chunk not to group. The syntax used in this option is almost the same as for the filters, in the format of `[ date1 [ member1 [ chunk1 chunk2 ] member2 [ chunk3 ... ] ... ] date2 [ member3 [ chunk1 ] ] ... ]`

Important: The grouping option is also in autosubmit monitor, create, setstatus and recovery

---

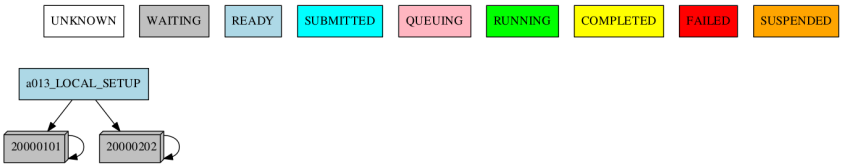
Examples:

Consider the following workflow:

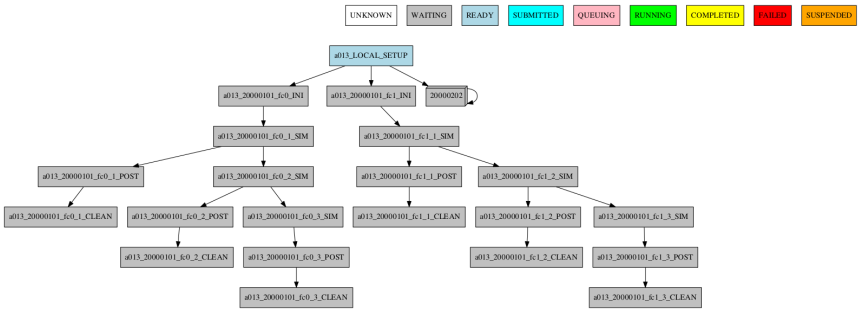


Group by date

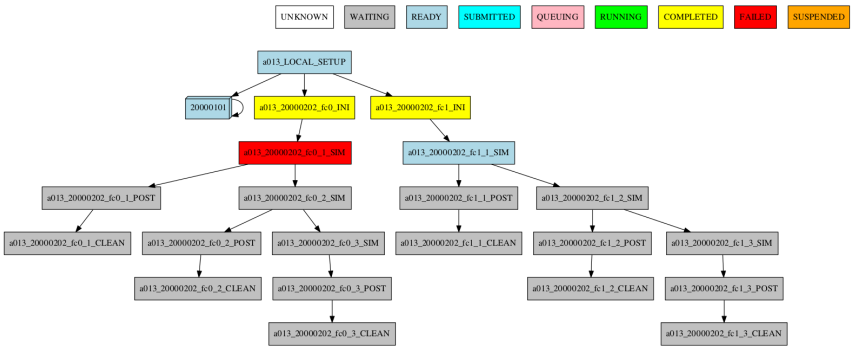
```
-group_by=date
```



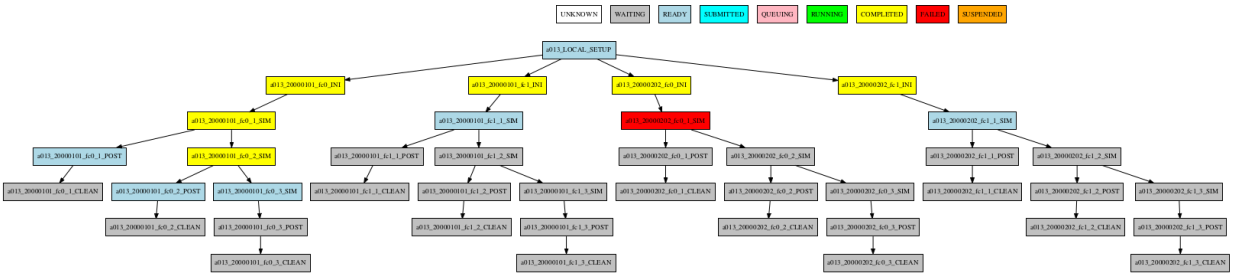

```
-group_by=date -expand="[ 20000101 ]"
```



```
-group_by=date -expand_status="FAILED RUNNING"
```

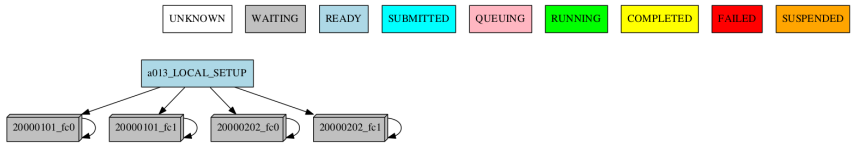



```
-group_by=date -expand="[ 20000101 ]" -expand_status="FAILED RUNNING"
```



Group by member

```
-group_by=member
```




```
-group_by=member -expand="[ 20000101 [ fc0 fc1 ] 20000202 [ fc0 ] ]"
```

```
-group_by=member -expand_status="FAILED QUEUING"
```

```
-group_by=member -expand="[ 20000101 [ fc0 fc1 ] 20000202 [ fc0 ] ]" -expand_status=
↪"FAILED QUEUING"
```

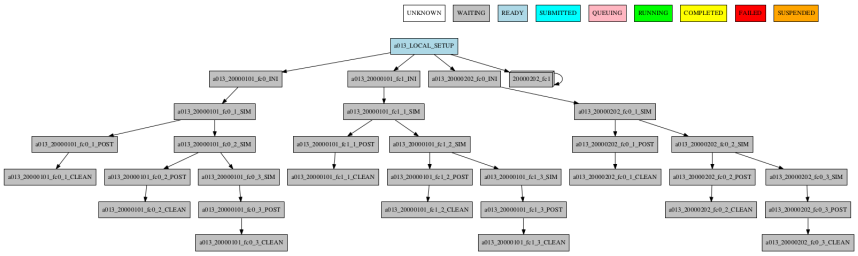
Group by chunk

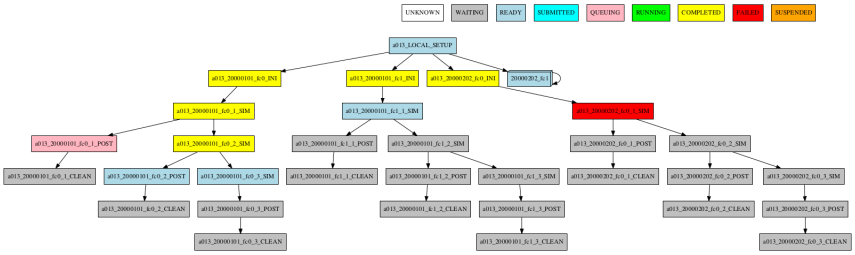
```
-group_by=chunk
```

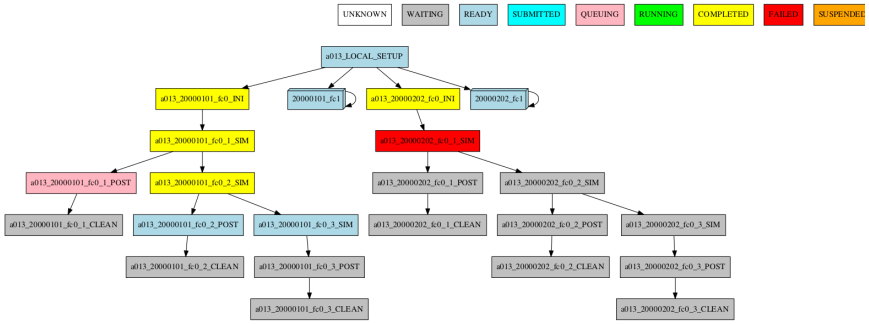
TODO: Add `group_chunk.png` figure.

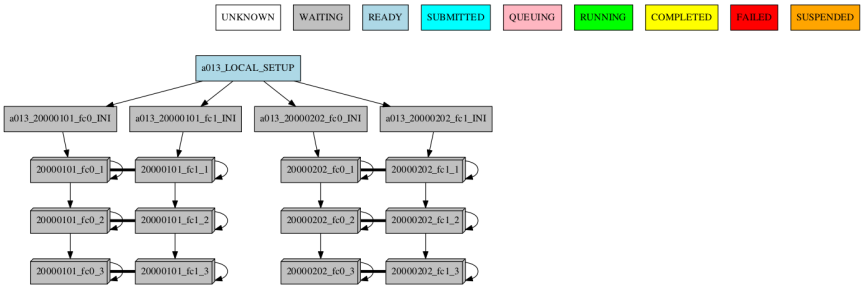
Synchronize jobs

If there are jobs synchronized between members or dates, then a connection between groups is shown:

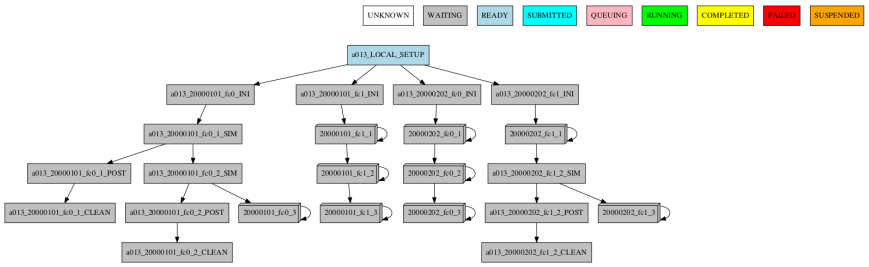

UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
a013_20000101_fc1_INI
a013_20000202_fc0_INI
20000202_fc1
a013_20000101_fc0_1_SIM
a013_20000101_fc1_1_SIM
a013_20000202_fc0_1_SIM
a013_20000101_fc0_1_POST
a013_20000101_fc0_2_SIM
a013_20000101_fc1_1_POST
a013_20000101_fc1_2_SIM
a013_20000202_fc0_1_POST
a013_20000202_fc0_2_SIM
a013_20000101_fc0_1_CLEAN
a013_20000101_fc0_2_POST
a013_20000101_fc0_3_SIM
a013_20000101_fc1_1_CLEAN
a013_20000101_fc1_2_POST
a013_20000101_fc1_3_SIM
a013_20000202_fc0_1_CLEAN
a013_20000202_fc0_2_POST
a013_20000202_fc0_3_SIM
a013_20000101_fc0_2_CLEAN
a013_20000101_fc0_3_POST
a013_20000101_fc1_2_CLEAN
a013_20000101_fc1_3_POST
a013_20000202_fc0_2_CLEAN
a013_20000202_fc0_3_POST
a013_20000101_fc0_3_CLEAN
a013_20000101_fc1_3_CLEAN
a013_20000202_fc0_3_CLEAN


UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
a013_20000101_fc1_INI
a013_20000202_fc0_INI
20000202_fc1
a013_20000101_fc0_1_SIM
a013_20000101_fc1_1_SIM
a013_20000202_fc0_1_SIM
a013_20000101_fc0_1_POST
a013_20000101_fc0_2_SIM
a013_20000101_fc1_1_POST
a013_20000101_fc1_2_SIM
a013_20000202_fc0_1_POST
a013_20000202_fc0_2_SIM
a013_20000101_fc0_1_CLEAN
a013_20000101_fc0_2_POST
a013_20000101_fc0_3_SIM
a013_20000101_fc1_1_CLEAN
a013_20000101_fc1_2_POST
a013_20000101_fc1_3_SIM
a013_20000202_fc0_1_CLEAN
a013_20000202_fc0_2_POST
a013_20000202_fc0_3_SIM
a013_20000101_fc0_2_CLEAN
a013_20000101_fc0_3_POST
a013_20000101_fc1_2_CLEAN
a013_20000101_fc1_3_POST
a013_20000202_fc0_2_CLEAN
a013_20000202_fc0_3_POST
a013_20000101_fc0_3_CLEAN
a013_20000101_fc1_3_CLEAN
a013_20000202_fc0_3_CLEAN

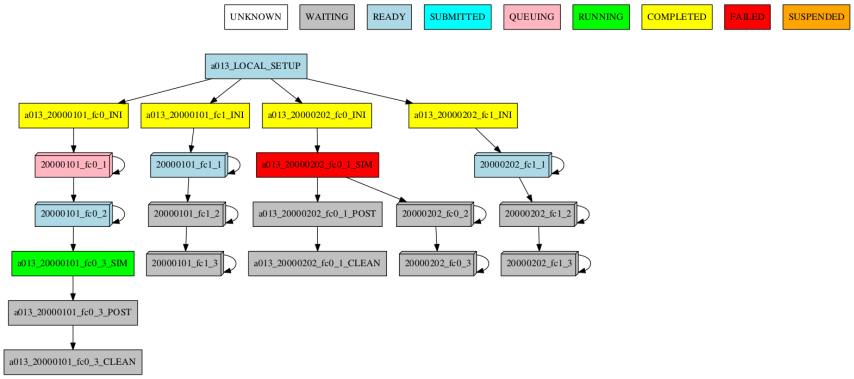

UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
20000101_fc1
a013_20000202_fc0_INI
20000202_fc1
a013_20000101_fc0_1_SIM
a013_20000202_fc0_1_SIM
a013_20000101_fc0_1_POST
a013_20000101_fc0_2_SIM
a013_20000202_fc0_1_POST
a013_20000202_fc0_2_SIM
a013_20000101_fc0_1_CLEAN
a013_20000101_fc0_2_POST
a013_20000101_fc0_3_SIM
a013_20000202_fc0_1_CLEAN
a013_20000202_fc0_2_POST
a013_20000202_fc0_3_SIM
a013_20000101_fc0_2_CLEAN
a013_20000101_fc0_3_POST
a013_20000202_fc0_2_CLEAN
a013_20000202_fc0_3_POST
a013_20000101_fc0_3_CLEAN
a013_20000202_fc0_3_CLEAN


UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
a013_20000101_fc1_INI
a013_20000202_fc0_INI
a013_20000202_fc1_INI
20000101_fc0_1
20000101_fc1_1
20000202_fc0_1
20000202_fc1_1
20000101_fc0_2
20000101_fc1_2
20000202_fc0_2
20000202_fc1_2
20000101_fc0_3
20000101_fc1_3
20000202_fc0_3
20000202_fc1_3

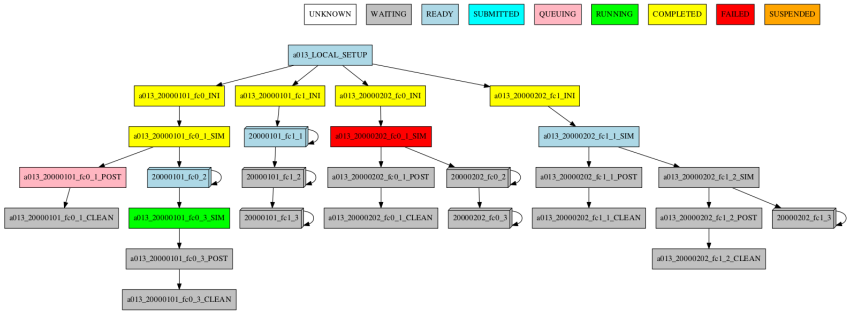
```
-group_by=chunk -expand="[ 20000101 [ fc0 [1 2] ] 20000202 [ fc1 [2] ] ]"
```




```
-group_by=chunk -expand_status="FAILED RUNNING"
```




```
-group_by=chunk -expand="[ 20000101 [ fc0 [1] ] 20000202 [ fc1 [1 2] ] ]" -expand_status=
↪"FAILED RUNNING"
```



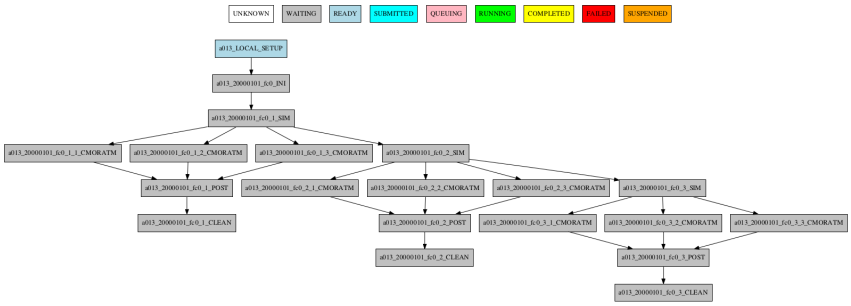
Group by split

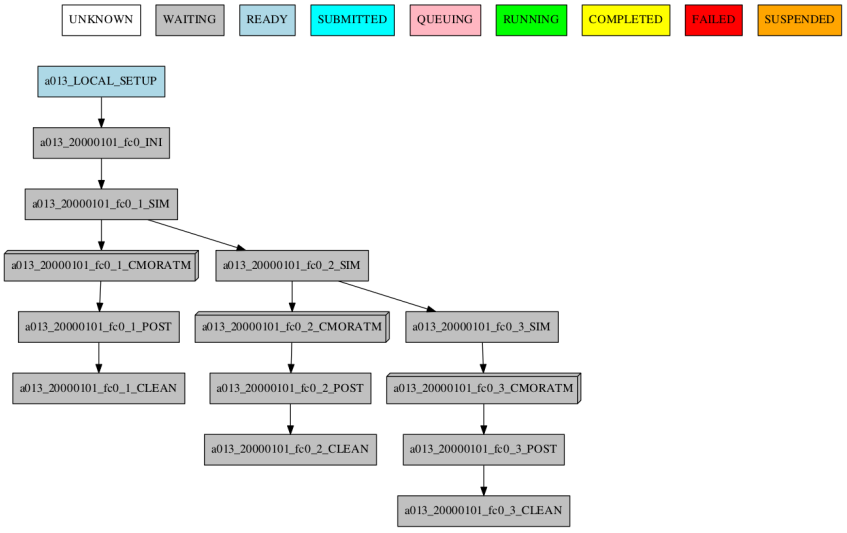
If there are chunk jobs that are split, the splits can also be grouped.

```
-group_by=split
```

Understanding the group status

If there are jobs with different status grouped together, the status of the group is determined as follows: If there is at least one job that failed, the status of the group will be FAILED. If there are no failures, but if there is at least one job running, the status will be RUNNING. The same idea applies following the hierarchy: SUBMITTED, QUEUING,


UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
a013_20000101_fc0_1_SIM
a013_20000101_fc0_1_1_CMORATM
a013_20000101_fc0_1_2_CMORATM
a013_20000101_fc0_1_3_CMORATM
a013_20000101_fc0_2_SIM
a013_20000101_fc0_1_POST
a013_20000101_fc0_2_1_CMORATM
a013_20000101_fc0_2_2_CMORATM
a013_20000101_fc0_2_3_CMORATM
a013_20000101_fc0_3_SIM
a013_20000101_fc0_1_CLEAN
a013_20000101_fc0_2_POST
a013_20000101_fc0_3_1_CMORATM
a013_20000101_fc0_3_2_CMORATM
a013_20000101_fc0_3_3_CMORATM
a013_20000101_fc0_2_CLEAN
a013_20000101_fc0_3_POST
a013_20000101_fc0_3_CLEAN

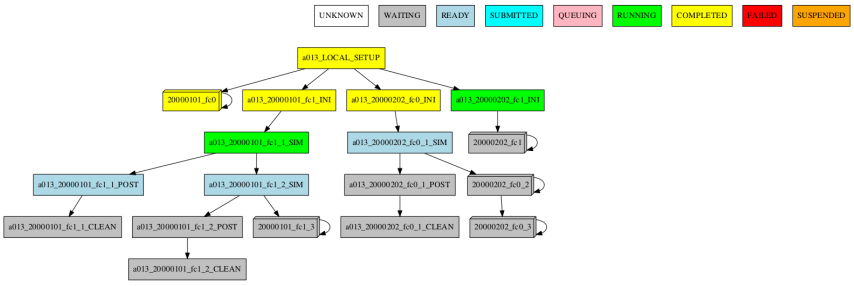

UNKNOWN
WAITING
READY
SUBMITTED
QUEUING
RUNNING
COMPLETED
FAILED
SUSPENDED
a013_LOCAL_SETUP
a013_20000101_fc0_INI
a013_20000101_fc0_1_SIM
a013_20000101_fc0_1_CMORATM
a013_20000101_fc0_2_SIM
a013_20000101_fc0_1_POST
a013_20000101_fc0_2_CMORATM
a013_20000101_fc0_3_SIM
a013_20000101_fc0_1_CLEAN
a013_20000101_fc0_2_POST
a013_20000101_fc0_3_CMORATM
a013_20000101_fc0_2_CLEAN
a013_20000101_fc0_3_POST
a013_20000101_fc0_3_CLEAN

READY, WAITING, SUSPENDED, UNKNOWN. If the group status is COMPLETED, it means that all jobs in the group were completed.

Automatic grouping

For the automatic grouping, the groups are created by collapsing the split->chunk->member->date that share the same status (following this hierarchy). The following workflow automatic created the groups 20000101_fc0, since all the jobs for this date and member were completed, 20000101_fc1_3, 20000202_fc0_2, 20000202_fc0_3 and 20000202_fc1, as all the jobs up to the respective group granularity share the same - waiting - status.

For example:



Especially in the case of monitoring an experiment with a very large number of chunks, it might be useful to hide the groups created automatically. This allows to better visualize the chunks in which there are jobs with different status, which can be a good indication that there is something currently happening within such chunks (jobs ready, submitted, running, queueing or failed).

```
-group_by=automatic --hide_groups
```

3.8.5 How to profile Autosubmit while monitoring an experiment

Autosubmit offers the possibility to profile the execution of the monitoring process. To enable the profiler, just add the `--profile` (or `-p`) flag to your `autosubmit monitor` command, as in the following example:

```
autosubmit monitor --profile EXPID
```

Note: Remember that the purpose of this profiler is to measure the performance of Autosubmit, not the jobs it runs.

This profiler uses Python's `cProfile` and `psutil` modules to generate a report with simple CPU and memory metrics which will be displayed in your console after the command finishes, as in the example below:

The profiler output is also saved in `<EXPID>/tmp/profile`. There you will find two files, the report in plain text format and a `.prof` binary which contains the CPU metrics. We highly recommend using SnakeViz to visualize this file, as follows:

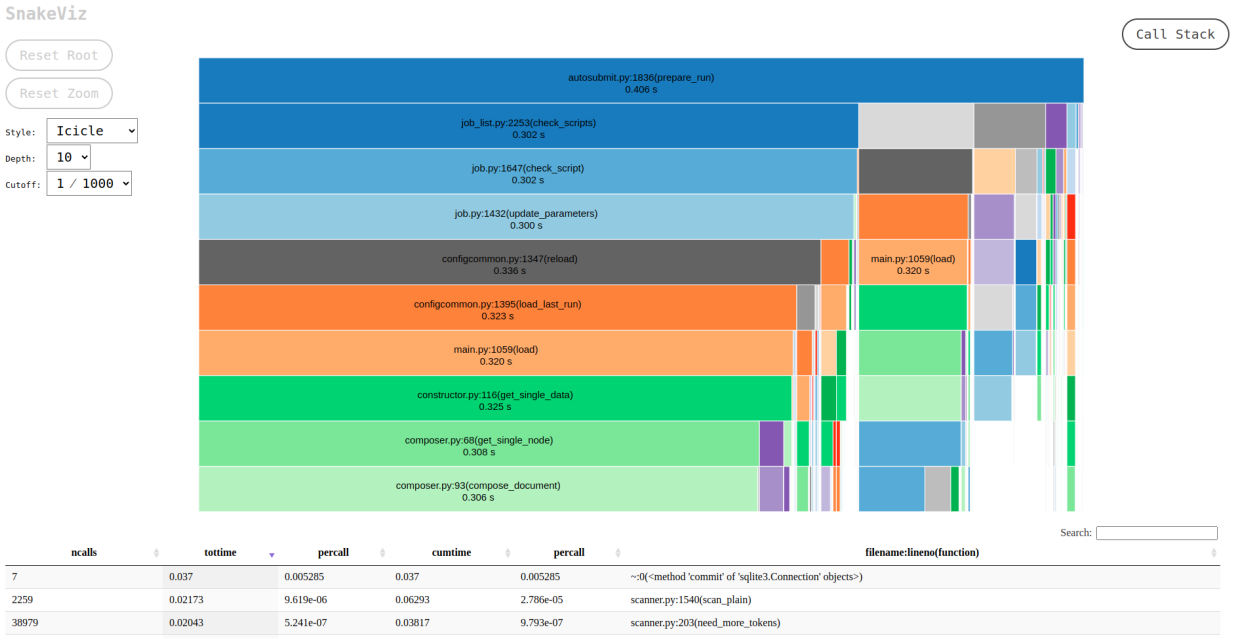
For more detailed documentation about the profiler, please visit this page.

```
No more jobs to run.
Run successful
================================================================================
                            Time & Calls Profiling
================================================================================

         1091713 function calls (1081212 primitive calls) in 0.581 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.564    0.564 autosubmit.py:1826(prepare_run)
        9    0.000    0.000    0.484    0.054 configcommon.py:1343(reload)
       17    0.000    0.000    0.470    0.028 constructor.py:116(get_single_data)
        9    0.000    0.000    0.467    0.052 configcommon.py:1391(load_last_run)
        9    0.000    0.000    0.463    0.051 main.py:1059(load)
        9    0.000    0.000    0.446    0.050 composer.py:68(get_single_node)
        9    0.000    0.000    0.445    0.049 composer.py:93(compose_document)
   2835/9    0.010    0.000    0.445    0.049 composer.py:111(compose_node)
    378/9    0.004    0.000    0.444    0.049 composer.py:199(compose_mapping_node)
        1    0.000    0.000    0.442    0.442 job_list.py:2253(check_scripts)
        8    0.000    0.000    0.441    0.055 job.py:1647(check_script)
        8    0.000    0.000    0.439    0.055 job.py:1432(update_parameters)
     8361    0.007    0.000    0.362    0.000 parser.py:141(check_event)
```



3.8.6 How to get details about the experiment

To get details about the experiment, use the command:

```
autosubmit describe {EXPID} {-u USERNAME}
```

EXPID is the experiment identifier, can be a list of expid separated by comma or spaces *-u USERNAME* is the username of the user who submitted the experiment.

It displays information about the experiment. Currently it describes owner,description_date,model,branch and hpc

Options:

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

  expid                 experiment identifier
  -u USERNAME, --user USERNAME username of the user who submitted the experiment
  -h, --help            show this help message and exit
```

Examples:

```
.. code-block:: bash
```

autosubmit describe cxxx autosubmit describe “cxxx cyyy” autosubmit describe cxxx,cyyy autosubmit describe -u dbeltran

3.8.7 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
  --hide,               hide the plot
  -nt                   --notransitive
                            prevents doing the transitive reduction when plotting the␣
↪workflow
```

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
```

Console output description

Example:

```
Period: 2021-04-25 06:43:00 ~ 2021-05-07 18:43:00
Submitted (#): 37
Run  (#): 37
Failed  (#): 3
Completed (#): 34
Queueing time (h): 1.61
Expected consumption real (h): 2.75
Expected consumption CPU time (h): 3.33
Consumption real (h): 0.05
Consumption CPU time (h): 0.06
Consumption (%): 1.75
```

Where:

- Period: Requested time frame.
- Submitted: Total number of attempts that reached the SUBMITTED status.
- Run: Total number of attempts that reached the RUNNING status.
- Failed: Total number of FAILED attempts of running a job.
- Completed: Total number of attempts that reached the COMPLETED status.
- Queueing time (h): Sum of the time spent queuing by attempts that reached the COMPLETED status, in hours.
- Expected consumption real (h): Sum of wallclock values for all jobs, in hours.
- Expected consumption CPU time (h): Sum of the products of wallclock value and number of requested processors for each job, in hours.
- Consumption real (h): Sum of the time spent running by all attempts of jobs, in hours.
- Consumption CPU time (h): Sum of the products of the time spent running and number of requested processors for each job, in hours.
- Consumption (%): Percentage of *Consumption CPU time* relative to *Expected consumption CPU time*.

Diagram output description

The main *stats* output is a bar diagram. On this diagram, each job presents these values:

- Queued (h): Sum of time spent queuing for COMPLETED attempts, in hours.
- Run (h): Sum of time spent running for COMPLETED attempts, in hours.
- Failed jobs (#): Total number of FAILED attempts.
- Fail Queued (h): Sum of time spent queuing for FAILED attempts, in hours.
- Fail Run (h): Sum of time spent running for FAILED attempts, in hours.
- Max wallclock (h): Maximum wallclock value for all jobs in the plot.

Notice that the left scale of the diagram measures the time in hours, and the right scale measures the number of attempts.

Custom statistics

Although Autosubmit saves several statistics about your experiment, as the queueing time for each job, how many failures per job, etc. The user also might be interested in adding his particular statistics to the Autosubmit stats report (`autosubmit stats EXPID`). The allowed format for this feature is the same as the Autosubmit configuration files: INI style. For example:

```
COUPLING:
LOAD_BALANCE: 0.44
RECOMMENDED_PROCS_MODEL_A: 522
RECOMMENDED_PROCS_MODEL_B: 418
```

The location where user can put this stats is in the file:

```
<experiments_directory>/cxxx/tmp/cxxx_GENERAL_STATS
```

Hint: If it is not yet created, you can manually create the file: `expid_GENERAL_STATS` inside the `tmp` folder.

3.8.8 How to extract information about the experiment parameters

This procedure allows you to extract the experiment variables that you want.

The command can be called with:

```
autosubmit report EXPID -t "absolute_file_path"
```

Alternatively it also can be called as follows:

```
autosubmit report expid -all
```

Or combined as follows:

```
autosubmit report expid -all -t "absolute_file_path"
```

Options:

```
usage: autosubmit report [-all] [-t] [-fp] [-p] expid

    expid                                 Experiment identifier

    -t, --template <path_to_template>     Allows to select a set of parameters to be␣
↪extracted

    -fp, --show_all_parameters            All parameters will be extracted to a different␣
↪file

    -fp, --folder_path                    By default, all parameters will be put into␣
↪experiment tmp folder

    -p, --placeholders                    disable the replacement by - if the variable␣
↪doesn't exist
```

Autosubmit parameters are encapsulated by %_%, once you know how the parameter is called you can create a template similar to the one as follows:

3.3: Template format and example.

```
 - **CHUNKS:** %NUMCHUNKS% - %CHUNKSIZE% %CHUNKSIZEUNIT%
 - **VERSION:** %VERSION%
 - **MODEL_RES:** %MODEL_RES%
 - **PROCS:** %XIO_NUMPROC% / %NEM_NUMPROC% / %IFS_NUMPROC% / %LPJG_NUMPROC% / %TM5_
↪NUMPROC_X% / %TM5_NUMPROC_Y%
 - **PRODUCTION_EXP:** %PRODUCTION_EXP%
 - **OUTCLASS:** %BSC_OUTCLASS% / %CMIP6_OUTCLASS%
```

This will be understood by Autosubmit and the result would be similar to:

```
- CHUNKS: 2 - 1 month
- VERSION: trunk
- MODEL_RES: LR
- PROCS: 96 / 336 / - / - / 1 / 45
- PRODUCTION_EXP: FALSE
- OUTCLASS: reduced /  -
```

Although it depends on the experiment.

If the parameter doesn't exists, it will be returned as '-' while if the parameter is declared but empty it will remain empty

3.4: List of all parameters example.

```
 HPCQUEUE: bsc_es
 HPCARCH: marenostrum4
 LOCAL_TEMP_DIR: /home/dbeltran/experiments/ASlogs
 NUMCHUNKS: 1
 PROJECT_ORIGIN: https://earth.bsc.es/gitlab/es/auto-ecearth3.git
 MARENOSTRUM4_HOST: mn1.bsc.es
 NORD3_QUEUE: bsc_es
 NORD3_ARCH: nord3
 CHUNKSIZEUNIT: month
```

(continues on next page)

(continued from previous page)

```
MARENOSTRUM4_LOGDIR: /gpfs/scratch/bsc32/bsc32070/a01w/LOG_a01w
PROJECT_COMMIT:
SCRATCH_DIR: /gpfs/scratch
HPCPROJ: bsc32
NORD3_BUDG: bsc32
```

3.9 Configuration details, setup and sharing

3.9.1 Experiment configuration

Since the beginning, Autosubmit has always been composed of five files in the folder `$expid/conf` that define the experiment configuration.

However, from Autosubmit 4, the configuration is no longer bound to one specific location. And it is composed of YAML files.

This document will teach you how to set up an experiment configuration using the different available methods and what Autosubmit expects to find in the configuration.

3.9.2 Standard configuration structure

The standard configuration is the one that is used by default. It is composed of five files in the folder `$expid/conf` that define the experiment configuration.

This configuration is generated by the `expid` command without any opcional flag or when using the -dm flag.

The following table summarizes what configuration files Autosubmit expects and what parameters you can define.

File	Content
`expdef.yml`	• It contains the default platform, the one set with -H. • Allows changing the start dates, members and chunks. • Allows changing the experiment project source (git, local, svn or dummy)
`platforms.yml`	• It contains the list of platforms to use in the experiment. • This file must be filled-up with the platform(s) configuration(s). • Several platforms can be defined and used in the same experiment.
`jobs.yml`	• It contains the tasks' definitions in sections. • This file must be filled-up with the tasks' definitions. • Several sections can be defined and used in the same experiment.
`autosubmit.yml`	• Parameters that control workflow behavior. • Parameters that activate extra functionalities.
`proj.yml`	• Project-dependent parameters.
`version.yml`	• Current AS version for this experiment.

It is worth mentioning that for Autosubmit 4, these files are seen as one. Therefore, the sections and parameters can be defined in any of the files.

Note: The `version.yml` file is automatically generated by Autosubmit. It is not necessary to create it.

Note: Autosubmit only admits the use of the `.yml` and `.yaml` (lowercase) extensions for the configuration files.

3.9.3 Advanced configuration structure and restrictions

From Autosubmit4, the configuration structure can be split into multiple locations and different files.

The experiment must have a **-minimal.yml* file in *$expid/conf* in the *$expid/conf* folder. This file is used to define the location of the configuration files and can be generated by the *expid* command when using with -min flag. This location can be defined by the user in the *DEFAULT.CONFIG_DIR* parameter inside a file ending with *minimal.yml* or *minimal.yaml* file.

- You would define the model-specific parameters inside your git or local repository. So when you push/pull the changes from git, they will be updated automatically.
- You would define the experiment-specific parameters under *$expid/conf*.
- You would define your user-specific parameters, for example, platform user, in a different location.

3.9.4 How to create and share the configuration

This section contains examples of creating a standard configuration and an advanced one from a newly made experiment.

Standard Configuration

The expid command can generate a sample structure containing all the parameters that Autosubmit needs to work correctly.

```
#Create a new experiment.
autosubmit expid -H "LOCAL" -d "Standard configuration."
# Get the expid from the output. Ex. expid=a000
cd $autosubmit_experiment_folder/a000
ls conf
autosubmit_a01y.yml  expdef_a01y.yml  platforms_a01y.yml
     jobs_a01y.yml    proj_a01y.yml
```

Sharing a standard Configuration

The expid command can copy another user's existing expid to work correctly.

```
#Create a new experiment.
autosubmit expid  --copy a000 -H "LOCAL" -d "Standard configuration. --copy of a000"
# Get the expid from the output. Ex. expid=a001
cd $autosubmit_experiment_folder/a001
ls conf
autosubmit_a001.yml  expdef_a001.yml  platforms_a001.yml
 jobs_a001.yml    proj_a001.yml
```

Warning: You must share the same Autosubmit experiment database for this to work.

Advanced Configuration

Autosubmit is now able to find the configuration files in different locations. The user can define the location of the configuration files in the *DEFAULT.CONFIG_DIR* parameter inside a file ending with *minimal.yml* or *minimal.yaml* file.

An skeleton of the advanced configuration can be generated through the *expid* command when using the *-min* flag.

```
#Create a new experiment.
autosubmit expid -min -d "Test minimal conf"
# Get the expid from the output. Ex. expid=a002
cd $autosubmit_experiment_folder/a002
ls conf
minimal.yml
```

To give a practical example, we will show an example using git. However, using a non-git folder is also possible.

Edit or generate a *minimal.yml* with the following parameters, leaving the rest untouched.

```
DEFAULT:
    #ADD, note that %PROJDIR% is an special AS_PLACEHOLDER that points to the `$expid/
↪proj/proj_destination` folder.
    CUSTOM_CONFIG:
        PRE: "%PROJDIR%/<path_to_model_as_conf>"
        POST: <path_to_user_conf>
PROJECT:
    PROJECT_TYPE: "git"
    PROJECT_DESTINATION: "git_project"
GIT:
    PROJECT_ORIGIN: "TO_FILL"
    PROJECT_BRANCH: "TO_FILL"
    PROJECT_COMMIT: "TO_FILL"
    PROJECT_SUBMODULES: "TO_FILL"
    FETCH_SINGLE_BRANCH: True
```

Important: The final configuration will be loaded in the following order: *PRE:$expid/%PROJDIR%/$as_proj_config_path -> $expid/conf -> POST*. Overwriting the parameters in the

order they are loaded.

CUSTOM_CONFIG: Syntax

The %DEFAULT.CUSTOM_CONFIG% parameter is used to define the location of the model/project or user files. The paths can be absolute or relative to the `%PROJDIR%`.

It has two different syntaxes:

- *Simple* a list of paths to the model or project yaml files. This can be a file or a folder. If it is a folder, all the files inside will be loaded in a non-recursive way.
- *Advanced* a dictionary with two keys: `PRE` and `POST`. The `PRE` key is used to define the files that will be loaded before the `$EXPID/CONF` ones. The `POST` key is used to define user configuration.

Note: With the simple syntax, the outcome is the same as the advanced one, but with the `POST` key empty.

Note: If a list of path is provided, the paths will be loaded in the order they are provided and in a recursive way. Meaning that in the case there are additional `DEFAULT.CUSTOM_CONFIG` parameter inside the files, they will be also loaded.

```
# Download the git project
autosubmit create a002
autosubmit refresh a002
```

Warning: Keep in mind that no parameters are disabled when custom_config is activated, including the jobs definitions.

3.9.5 Advanced configuration - Full dummy example (reproducible)

```
#Create a new experiment.
autosubmit expid -min -repo https://earth.bsc.es/gitlab/ces/auto-advanced_config_example␣
↪-b main -conf as_conf -d "Test minimal conf"
# expid=a04b
dbeltran@bsces107894: cd ~/autosubmit/a04b
dbeltran@bsces107894:~/autosubmit/a04b$ ls conf
minimal.yml
```

```
cat ~/autosubmit/conf/minimal.yml
```

```
CONFIG:
  AUTOSUBMIT_VERSION: "4.0.0b"
DEFAULT:
    EXPID: "a04b"
    HPCARCH: "local"
    #ADD, note that %PROJDIR% is an special AS_PLACEHOLDER that points to the expid␣
```

(continues on next page)

(continued from previous page)

```
↪folder.
    #hint: use %PROJDIR% to point to the project folder (where the project is cloned)
    CUSTOM_CONFIG: "%PROJDIR%/as_conf"
PROJECT:
    PROJECT_TYPE: "git"
    PROJECT_DESTINATION: "git_project"
GIT:
    PROJECT_ORIGIN: "https://earth.bsc.es/gitlab/ces/auto-advanced_config_example"
    PROJECT_BRANCH: "main"
    PROJECT_COMMIT: ""
    PROJECT_SUBMODULES: ""
```

```
# Download the git project to obtain the distributed configuration
dbeltran@bsces107894: autosubmit refresh a04b
# Check the downloaded model-configuration
dbeltran@bsces107894:~/autosubmit/a04b$ ls proj/git_project/as_conf/
autosubmit.yml  expdef.yml  jobs.yml  platforms.yml
```

Model configuration is distributed at git.

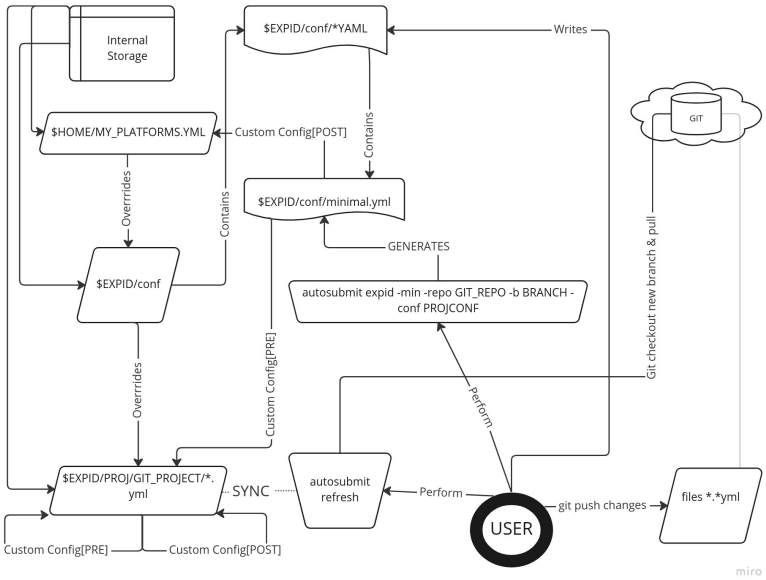
```
dbeltran@bsces107894:~/autosubmit/a04b$ cat ~/as_user_conf/platforms.yml
```

```
Platforms:
  MARENOSTRUM4:
    USER: bsc32xxx
    QUEUE: debug
    MAX_WALLCLOCK: "02:00"
  marenostrum_archive:
    USER: bsc32xxx
  transfer_node:
    USER: bsc32xxx
  transfer_node_bscearth000:
    USER: dbeltran
  bscearth000:
    USER: dbeltran
  nord3:
    USER: bsc32xxx
  ecmwf-xc40:
    USER: c3d
```

Note: The user configuration is not distributed, it is a local file that must be edited by the user.

```
# Create and run the experiment, since it contains all the info!
autosubmit create a04b  # if $expid/proj doesn't exists
autosubmit refresh a04b
autosubmit run a04b
```

The following figure shows the flow of the execution.



3.10: Advanced configuration example

Sharing an advanced configuration

The expid command can copy another user's existing expid to work correctly.

Note: This only copies the `$expid/conf/{*.yml,*yaml}` experiment configuration files.

```
#Create a new experiment.
autosubmit expid  --copy a002 -H "LOCAL" -d "Advanced configuration. --copy of a002"
# Get the expid from the output. Ex. expid=a004
cd $autosubmit_experiment_folder/a004
ls conf
minimal.yml
autosubmit create a004
```

Warning: All users must share the same experiment autosubmit.db for this to work. More info at shared-db

Sharing an experiment configuration across filesystems is possible only by including the same *DEFAULT.CUSTOM_CONFIG* and *GIT.PROJECT_ORIGIN*, *GIT.PROJECT_BRANCH* and *GIT.PROJECT_TAG* inside the expdef.yml file.

3.10 Variables reference

Autosubmit uses a variable substitution system to facilitate the development of the templates. These variables can be used on templates with the syntax `%VARIABLE_NAME%`.

All configuration variables that are not related to the current job or platform are available by accessing first their parents, e.g. `%PROJECT.PROJECT_TYPE% or %DEFAULT.EXPID%`.

You can review all variables at any given time by using the *report* command, as illustrated below.

3.5: Example usage of `autosubmit report`

```
$ autosubmit report $expid -all
```

The command will save the list of variables available to a file in the experiment area. The groups of variables of Autosubmit are detailed in the next sections on this page.

Note: All the variable tables are displayed in alphabetical order.

Note: Custom configuration files (e.g. `my-file.yml`) may contain configuration like this example:

```
MYAPP:
  MYPARAMETER: 42
  ANOTHER_PARAMETER: 1984
```

If you configure Autosubmit to include this file with the rest of your configuration, then those variables will be available to each job as `%MYAPP.MYPARAMETER%` and `%MYAPP.ANOTHER_PARAMETER%`.

3.10.1 Job variables

These variables are relatives to the current job. These variables appear in the output of the *report* command with the pattern `JOBS.${JOB_ID}.${JOB_VARIABLE}=${VALUE}`. They can be used in templates with `%JOB_VARIABLE%`.

Variable	Description
CHECKPOINT	Generates a checkpoint step for this job based on job.type.
CHUNK	Current chunk.
CPUS_PER_TASK	Number of threads that the job will use.
CURRENT_QUEUE	Returns the queue to be used by the job. Chooses between serial and parallel platforms.
CUSTOM_DIRECTIVES	List of custom directives.
DELAY	Current delay.
DELAY_RETRIALS	TODO
DEPENDENCIES	Current job dependencies.
EXPORT	TODO.
FAIL_COUNT	Number of failed attempts to run this job.
FREQUENCY	TODO.
HYPERTHREADING	Detects if hyperthreading is enabled or not.
JOBNAME	Current job full name.

continues on next page

3.1 – continued from previous page

Variable	Description
MEMBER	Current member.
MEMORY	Memory requested for the job.
MEMORY_PER_TASK	Memory requested per task.
NODES	Number of nodes that the job will use.
NUMMEMBERS	Number of members of the experiment.
NUMPROC	Number of processors that the job will use.
NUMTASK	Number of tasks that the job will use.
NUMTHREADS	Number of threads that the job will use.
PACKED	TODO
PROCESSORS	Number of processors that the job will use.
PROCESSORS_PER_NODE	Number of processors per node that the job can use.
PROJDIR	Project folder path.
RETRIALS	Max amount of retrials to run this job.
ROOTDIR	Experiment folder path.
SCRATCH_FREE_SPACE	Percentage of free space required on the `scratch`.
SDATE	Current start date.
SPLIT	Current split.
SPLITS	Max number of splits.
SYNCHRONIZE	TODO.
TASKS	Number of tasks that the job will use.
TASKS_PER_NODE	Number of tasks that the job will use.
TASKTYPE	Type of the job, as given on job configuration file.
THREADS	Number of threads that the job will use.
WALLCLOCK	Duration for which nodes used by job will remain allocated.

The following variables are present only in jobs that contain a date (e.g. `RUNNING=date`).

Variable	Description
CHUNK_END_DATE	Chunk end date.
CHUNK_END_DAY	Chunk end day.
CHUNK_END_HOUR	Chunk end hour.
CHUNK_END_IN_DAYS	Days passed from the start of the simulation until the end of the chunk.
CHUNK_END_MONTH	Chunk end month.
CHUNK_END_YEAR	Chunk end year.
CHUNK_FIRST	True if the current chunk is the first, false otherwise.
CHUNK_LAST	True if the current chunk is the last, false otherwise.
CHUNK_SECOND_TO_LAST_DATE	Chunk second to last date.
CHUNK_SECOND_TO_LAST_DAY	Chunk second to last day.
CHUNK_SECOND_TO_LAST_HOUR	Chunk second to last hour.
CHUNK_SECOND_TO_LAST_MONTH	Chunk second to last month.
CHUNK_SECOND_TO_LAST_YEAR	Chunk second to last year.
CHUNK_START_DATE	Chunk start date.
CHUNK_START_DAY	Chunk start day.
CHUNK_START_HOUR	Chunk start hour.
CHUNK_START_MONTH	Chunk start month.
CHUNK_START_YEAR	Chunk start year.
DAY_BEFORE	Day before the start date.
NOTIFY_ON	Determine the job statuses you want to be notified.
PREV	Days since start date at the chunk's start.
RUN_DAYS	Chunk length in days.

Custom directives

There are job variables that Autosubmit automatically converts into directives for your batch server. For example, `NUMTHREADS` will be set in a Slurm platform as `--SBATCH --cpus-per-task=$NUMTHREADS`.

However, the variables in Autosubmit do not contain all the directives available in each platform like Slurm. For values that do not have a direct variable, you can use `CUSTOM_DIRECTIVES` to define them in your target platform. For instance, to set the number of GPU's in a Slurm job, you can use `CUSTOM_DIRECTIVES=--gpus-per-node=10`.

3.10.2 Platform variables

These variables are relative to the platforms defined in each job configuration. The table below shows the complete set of variables available in the current platform. These variables appear in the output of the *report* command with the pattern `JOBS.${JOB_ID}.${PLATFORM_VARIABLE}=${VALUE}`. They can be used in templates with `%PLATFORM_VARIABLE%`.

A series of variables is also available in each platform, and appear in the output of the *report* command with the pattern `JOBS.${JOB_ID}.PLATFORMS.${PLATFORM_ID}.${PLATFORM_VARIABLE}=${VALUE}`. They can be used in templates with `PLATFORMS.%PLATFORM_ID%.%PLATFORM_VARIABLE%`.

Variable	Description
CURRENT_ARCH	Platform name.
CURRENT_BUDG	Platform budget.
CURRENT_EXCLUSIVITY	True if you want to request exclusivity nodes.
CURRENT_HOST	Platform url.
CURRENT_HYPERTHREADING	TODO
CURRENT_LOGDIR	The platform's LOG directory.
CURRENT_PARTITION	Partition to use for jobs.
CURRENT_PROJ	Platform project.
CURRENT_PROJ_DIR	Platform's project folder path.
CURRENT_RESERVATION	You can configure your reservation id for the given platform.
CURRENT_ROOTDIR	Platform's experiment folder path.
CURRENT_SCRATCH_DIR	Platform's scratch folder path.
CURRENT_TYPE	Platform scheduler type.
CURRENT_USER	Platform user.

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

Certain variables (e.g. `_RESERVATION`, `_EXCLUSIVITY`) are only available for certain platforms (e.g. MareNostrum).

A set of variables for the experiment's default platform are also available.

Variable	Description
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.

3.10.3 Other variables

Variable	Description
CONFIG.AUTOSUBMIT_VERSION	Current version of Autosubmit.
CONFIG.MAXWAITINGJOBS	Maximum number of jobs permitted in the waiting status.
CONFIG.TOTALJOBS	Total number of jobs in the workflow.

Variable	Description
DEFAULT.CUSTOM_CONFIG	Custom configuration location.
DEFAULT.EXPID	Job experiment ID.
DEFAULT.HPCARCH	Default HPC platform name.

Variable	Description
EXPERIMENT.CALENDAR	Calendar used for the experiment. Can be standard or noleap.
EXPERIMENT.CHUNKSIZE	Size of each chunk.
EXPERIMENT.CHUNKSIZEUNIT	Unit of the chunk size. Can be hour, day, month, or year.
EXPERIMENT.DATELIST	List of start dates
EXPERIMENT.MEMBERS	List of members.
EXPERIMENT.NUMCHUNKS	Number of chunks of the experiment.

Variable	Description
PROJECT.PROJECT_DESTINATION	Destination of the project sources.
PROJECT.PROJECT_TYPE	Type of the project.

Note: Depending on your project type other variables may be available. For example, if you choose Git, then you should have `%PROJECT_ORIGIN%`. If you choose Subversion, then you will have `%PROJECT_URL%`.

3.10.4 Performance Metrics variables

These variables apply only to the *report* subcommand.

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

3.11 Experiment ID's

Autosubmit 4 uses a SQLite database to automatically generate unique experiment ID's. The ID's of Autosubmit experiments start from `a000` and have at least four alpha-numerical characters, using digits from `0` to `9` and the `26` letters from the English alphabet, from `a` to `z`.

Internally, experiment ID's are case insensitive, but for Autosubmit commands this may not always be true, i.e. `autosubmit monitor a000` works for the experiment `a000`, but not `autosubmit monitor A000`, even though internally both `a000` and `A000` would be stored the same way in the SQLite database.

That is because experiment ID's are treated as Base-36 strings, being first decoded into integers with Base-36 (where `a` and `A` are both equal to `10`, `b` and `B` equal to `11`, etc. — `int('a', 36)` in Python).

Autosubmit provides functions that could be used by external code to produce a new experiment, or to simply calculate the next experiment ID, given the last available experiment ID. The code below shows an example of the latter:

```
from autosubmit.experiment.experiment_common import next_experiment_id

expid = next_experiment_id('a000')
print(expid)  # prints 'a001'

expid = next_experiment_id('jedi')
print(expid)  # prints 'jedj'

expid = next_experiment_id('zzzz')
print(expid)  # prints '10000'
```

To generate the next experiment ID, the decoded integer value is incremented by 1, and then re-encoded as a Base-36 string. For example, `a000` is decoded as `466560`, so the next ID is calculated as `466560 + 1 = 466561`. Finally, `466561` is re-encoded as Base-36, resulting in `a001`.

After the default initial experiment ID `a000`, the next generated experiment ID is `a001`, and it keeps being increased automatically by Autosubmit from `a001`, to `a002`, `a003`, ..., `azzz` and then the next experiment ID `b000`.

And the process repeats every time users ask Autosubmit to create a new experiment.

Users can create "test experiments" which experiment ID's start at `t001`, and "operational experiments" which experiment ID's start at `o001`. This is done via flags passed to the `autosubmit expid` in the command-line.

There is no other way for users to modify the automatic generation of experiment ID's in Autosubmit (other than manually editing the SQLite database).

The total number of available unique experiment ID's in Autosubmit with four characters is `1213055` experiment ID's (the difference between `a000` and `zzzz` Base-36 decoded as integers). After the experiment `zzzz`, the next experiment ID generated by Autosubmit would be `10000`, followed by `10001`, `10002`, and so on successfully.

3.12 Provenance

Autosubmit manages experiments following the FAIR data principles, findability, accessibility, interoperability, and reusability. It supports and uses open standards such as YAML, RO-Crate, as well as other standards such as ISO-8601.

Each Autosubmit experiment is assigned a *unique experiment ID* (also called expid). It also provides a central database and utilities that permit experiments to be referenced.

Every Autosubmit command issued by a user generates a timestamped log file in `<EXPID>/tmp/ASLOGS/`. For example, when the user runs `autosubmit create <EXPID>` and `autosubmit run <EXPID>`, these commands should create files like `<EXPID>/tmp/ASLOGS/20230808_092350_create.log` and `<EXPID>/tmp/ASLOGS/20230808_092400_run.log`, with the same content that was displayed in the console output to the user running it.

Users can *archive Autosubmit experiments*. These archives contain the complete logs and other files in the experiment directory, and can be later unarchived and executed again. Supported archival formats are ZIP and **RO-Crate**.

3.12.1 RO-Crate

RO-Crate is a community standard adopted by other workflow managers to package research data with their metadata. It is extensible, and contains profiles to package computational workflows. From the RO-Crate website, "What is RO-Crate?":

> RO-Crate is a community effort to establish a lightweight approach to packaging research data with their metadata. It is based on schema.org annotations in JSON-LD, and aims to make best-practice in formal metadata description accessible and practical for use in a wider variety of situations, from an individual researcher working with a folder of data, to large data-intensive computational research environments.

Autosubmit conforms to the following RO-Crate profiles:

- Process Run Crate
- Workflow Run Crate
- Workflow RO-Crate

Experiments archived as RO-Crate can also be uploaded to Zenodo and to WorkflowHub. The Autosubmit team worked with the WorkflowHub team to add Autosubmit as a supported language for workflows. Both Zenodo and WorkflowHub are issuers of DOI's (digital object identifiers), which can be used as persistent identifiers to resolve Autosubmit experiments referenced in papers and other documents.

3.13 Command list

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

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

3.13.1 Tutorials (How to)

- *Create an Experiment*
- *Configure Experiments*
- *How to prepare an experiment to run in two independent job_list. (Priority jobs, Two-step-run) (OLD METHOD)*
- *How to restart the experiment*

TODO add `workflow_validation`.

- *How to monitor job statistics*
- *How to archive an experiment*
- *Advanced Configuration*

CHAPTER

FOUR

DATABASES

4.1 Introduction

Autosubmit stores information about its experiments and workflows in SQLite databases and as serialized Python objects (pickle files). These are distributed through the local filesystem, where Autosubmit is installed and runs.

There is one central database that supports the core functionality of experiments in Autosubmit. There are other auxiliary databases consumed by Autosubmit and the Autosubmit API, that store finer-grained experiment information.

The name and location of the central database are defined in the `.autosubmitrc` configuration file while the other auxiliary databases have a predefined name. There are also log files with important information about experiment execution and some other relevant information such as experiment job statuses, timestamps, error messages among other things inside these files.

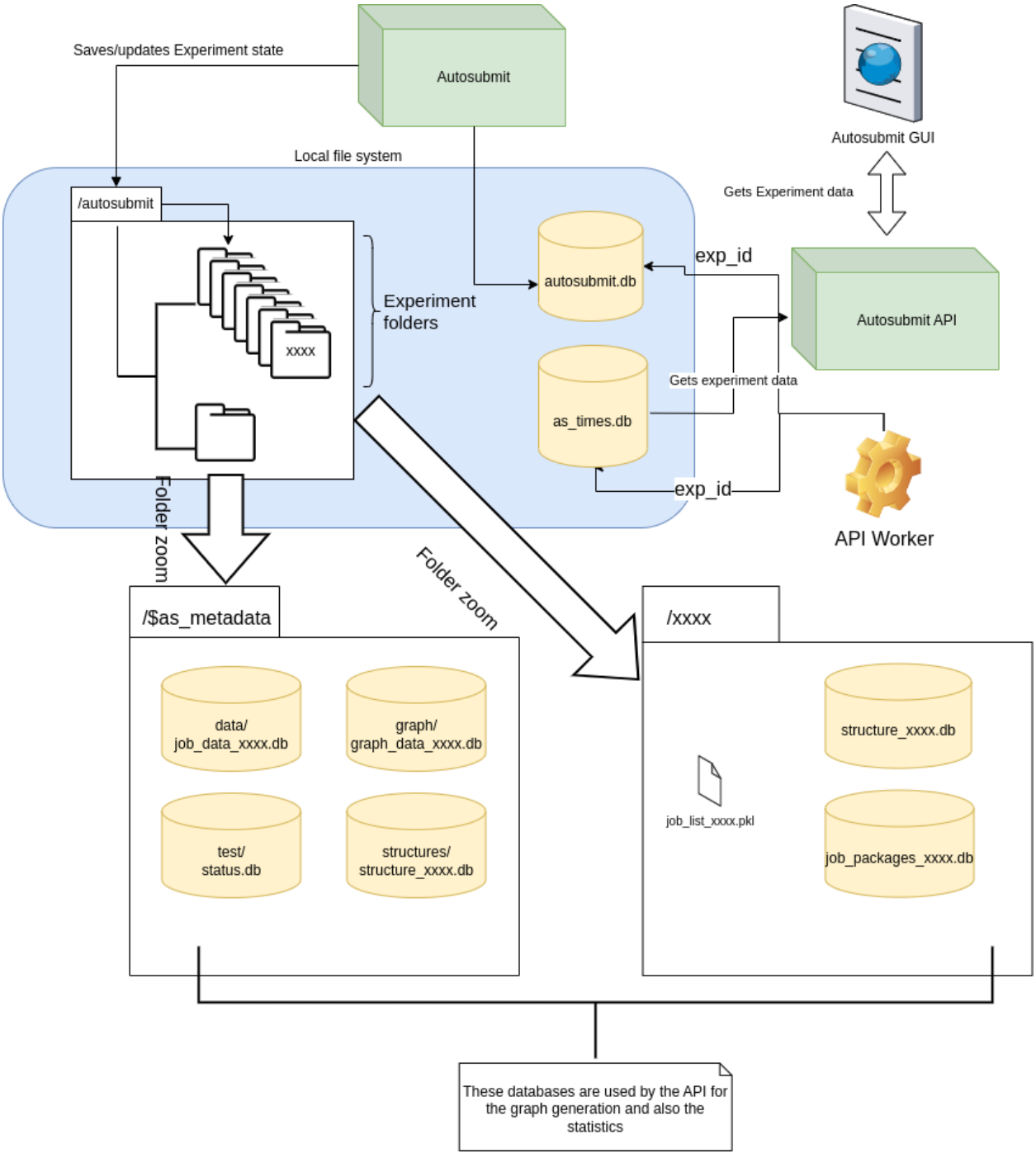
Note: The <EXPID> is an experiment ID. The location of the databases of other files can be customized in the `.autosubmitrc` configuration file.

4.2 Core databases

Database	Default location	Description
`autosubmit.db`	`$HOME/autosubmit/autosubmit.db`	The main database of Autosubmit. The location can be customized in the `autosubmitrc` file.
`as_times.db`	`$HOME/autosubmit/as_times.db`	Deprecated API. Used by Autosubmit API with Autosubmit `3.x`. Kept for backward compatibility for now.

4.3 Auxiliary databases

These databases complement the databases previously described for different purposes. Some of them are centralized in the `$AS_METADATA` directory (defined in the `.autosubmitrc` config file) while others are present inside each experiment folder.


Saves/updates Experiment state
Autosubmit
Autosubmit GUI
Local file system
Gets Experiment data
/autosubmit
exp_id
autosubmit.db
Autosubmit API
Experiment folders
xxxx
Gets experiment data
as_times.db
exp_id
API Worker
Folder zoom
Folder zoom
/$as_metadata
/xxxx
data/ job_data_xxxx.db
graph/ graph_data_xxxx.db
structure_xxxx.db
job_list_xxxx.pkl
test/ status.db
structures/ structure_xxxx.db
job_packages_xxxx.db
These databases are used by the API for the graph generation and also the statistics

4.3.1 Databases in the $AS_METADATA directory

Database	Default location	Description
graph_data_<EXPID>.db	$HOME/autosubmit/metadata/graph/graph_data_<EXPID>.db	Used by the GUI to improve the graph visualization. Populated by an API worker.
structure_<EXPID>.db	$HOME/autosubmit/metadata/structures/structure_<EXPID>.db	Used by the GUI to display edge lists. Populated by an API worker.
status.db	$HOME/autosubmit/metadata/test/status.db	Stores the status of the partition where Autosubmit databases and experiment files are stored. Populated by an API worker.
job_data_<EXPID>.db	$HOME/autosubmit/metadata/data/job_data_<EXPID>.db	Stores experiment metrics and historical information. Populated by Autosubmit.

4.3.2 Databases in each experiment directory

Database	Default location	Description
job_packages_<EXPID>.db	$HOME/autosubmit/<EXPID>/pkl/job_packages_<EXPID>.db	Stores information about the wrappers configured in the experiment. Empty if no wrappers configured.
structure_<EXPID>.db	$HOME/autosubmit/<EXPID>/pkl/structure_<EXPID>.db	Deprecated. Used in Autosubmit 3.x, now replaced by the database used by the Autosubmit API (described above).

4.4 Other files

Autosubmit stores Pickle files (e.g. $HOME/autosubmit/<EXPID>/pkl/job_list_<EXPID>.pkl) with the job list of experiments. In the event of a crash, or if the user stops the experiment, that Pickle file is used in order to be able to restore the experiment to its latest status.

There are also update list files, used to change the status of experiment jobs without stopping Autosubmit. These files are plain text files, and also present in the experiment directory.

CHAPTER

FIVE

ERROR CODES AND SOLUTIONS

Note: Increasing the logging level gives you more detailed information about your error, e.g. `autosubmit -lc DEBUG -lf DEBUG <CMD>`, where <CMD> could be `create`, `run`, etc.

Every error in Autosubmit contains a numeric error code, to help users and developers to identify the category of the error. These errors are organized as follows:

Level	Starts from
EVERYTHING	0
STATUS_FAILED	500
STATUS	1000
DEBUG	2000
WARNING	3000
INFO	4000
RESULT	5000
ERROR	6000
CRITICAL	7000
NO_LOG	8000

Levels such as `DEBUG`, `WARNING`, `INFO`, and `RESULT` are commonly used when writing log messages. You may find it in the output of commands in case there is a minor issue with your configuration such as a deprecated call.

The two levels that normally appear with traceback and important log messages are either `ERROR` or `CRITICAL`.

Autosubmit has two error types. `AutosubmitError` uses the `ERRORR` level, and is raised for minor errors where the program execution may be able to recover. `AutosubmitCritical` uses the `CRITICAL` level and is for errors that abort the program execution.

The detailed error codes along with details and possible workarounds are listed below.

5.1 Minor errors - Error codes [6000+]

Code	Details	Solution
6001	Failed to retrieve log files	Automatically, if there are no major issues
6002	Failed to reconnect	Automatically, if there are no major issues
6003	Failed connection, wrong configuration	Check your platform configuration
6004	Input output issues	Automatically, if there are no major issues
6005	Unable to execute the command	Automatically, if there are no major issues
6006	Failed command	Check err output for more info, command worked but some issue was detected
6007	Broken sFTP connection	Automatically, if there are no major issues
6008	Inconsistent/unexpected, job status	Automatically, if there are no major issues
6009	Failed job checker	Automatically, if there are no major issues
6010	Corrupted `job_list` using backup	Automatically, if it fails try `mv <EXPID>/pkl/job_list_backup.pkl <EXPID>/pkl/job_list.pkl`
6011	Incorrect mail notifier configuration	Double check your mail configuration on your job configuration (job status) and the experiment configuration (email)
6012	Migrate, archive/unarchive I/O issues	Check the migrate configuration
6013	Configuration issues	Check log output for more info
6014	Git can not clone repository submodule	Check submodule url, perform a `refresh`
6015	Submission failed	Automatically, if there are no major issues
6016	Temporary connection issues	Automatically, if there are no major issues

5.2 Experiment Locked - Critical Error 7000

Code	Details	Solution
7000	Experiment is locked due another instance of Autosubmit using it	Halt other experiment instances, then `rm <EXPID>/tmp/autosubmit.lock`

5.3 Database Issues - Critical Error codes [7001-7009]

These issues occur due to server side issues. Check your site settings, and report an issue to the Autosubmit team in Git if the issue persists.

Code	Details	Solution
7001	Connection to the db could not be established	Check if database exists
7002	Wrong version	Check system sqlite version
7003	Database doesn't exist	Check if database exists
7004	Can't create a new database	Check your user permissions
7005	AS database is corrupted or locked	Report the issue to the Autosubmit team in Git
7006	Experiment database not found	Ask the site administrator to run `autosubmit install`
7007	Experiment database permissions	Invalid permissions, ask your administrator to add `R/W`

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

These issues are caused by the user input. Check the logs and also the existing issues in git for possible workarounds. Report an issue to the Autosubmit team in Git if the issue persists.

Code	Details	Solution
7010	Experiment has been halted manually	
7011	Wrong arguments for a specific command	Check the command section for more information
7012	Insufficient permissions for an specific experiment	Check if you have enough permissions, and that the experiment exists
7013	Pending commits	You must commit pending changes in the experiment `proj` folder
7014	Wrong configuration	Check your experiment configuration files, and at the `<EXPID>/tmp/ASLOG/<CMD>.log` output
7015	Job list is empty	Check your experiment configuration files

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

The Autosubmit logs should contain more detailed information about the error. Check your platform configuration and general status (connectivity, permissions, etc.).

Code	Details	Solution
7040	Invalid experiment `pkl` or `db` files	Should be recovered automatically, if not check if there is a backup file and do it manually
7041	Unexpected job status	Try to run `autosubmit recovery <EXPID>`, report the issue to the Autosubmit team if it persists
7050	Connection can not be established	Check your experiment platform configuration
7051	Invalid SSH configuration	Check `.ssh/config` file. Additionally, check if you can perform a password-less connection to that platform
7052	Scheduler is not installed or not correctly configured	Check if there is a scheduler installed in the remote machine

5.6 Uncatalogued codes - Critical Error codes [7060+]

The Autosubmit logs should contain more detailed information about the error. If you believe you found a bug, feel free to report an issue to the Autosubmit team in Git.

Code	Details	Solution
7060	Display issues during monitoring	Use a different output or use plain text (`txt`)
7061	Stat command failed	Check the command output in `ASLOGS` for a possible bug, report it to the Autosubmit team in Git
7062	Svn issues	Check if URL was configured in the experiment configuration
7063	cp/rsync issues	Check if destination path exists
7064	Git issues	Check `GIT:` experiment configuration. If issue persists, check if `proj` folder is a valid Git repository
7065	Wrong git configuration	Invalid Git url. Check `GIT:` experiment configuration. If issue persists, check if `proj` folder is a valid Git repository
7066	Pre-submission feature issues	New feature, this message should not be issued, Please report it in Git
7067	Historical Database not found	Configure `historicdb: PATH:<file_path>`
7068	Monitor output can't be loaded	Try another output method, check if the experiment exists and is readable
7069	Monitor output format invalid	Try another output method
7070	Bug in the code	Please submit an issue to the Autosubmit team in Git
7071	AS can't run in this host	If you think that this is an error, check the `.autosubmitrc` and modify the allowed and forbidden directives
7072	Basic configuration not found	Administrator: run `autosubmit configure --advanced` or create a common file in `/etc/autosubmitrc`. User: run `autosubmit configure` or create a `$HOME/.autosubmitrc` (consult the installation documentation)
7073	Private key is encrypted	Add your key to your ssh agent, e.g. `ssh-add $HOME/.ssh/id_rsa`, then try running Autosubmit again. You can also use a non-encrypted key (make sure nobody else has access to the file)
7074	Profiling process failed	You can find more detailed information in the logs, as well as hints to solve the problem

Note: Please submit an issue to the Autosubmit team if you have not found your error code listed here.

CHAPTER

SIX

CHANGELOG

This page shows the main changes from AS3 to AS4.

Mayor mentions:

- Python version has changed to 3.7.3 instead of 2.7.
- Configuration language has changed to YAML.
- All parameters are now unified into a single dictionary.
- All sections are now uppercase.
- All parameters, except for job related ones, have now an hierarchy.
- An special key, FOR:, has been added. This key allows to create multiple jobs with almost the same configuration.
- The configuration of autosubmit is now more flexible.
- New command added, upgrade. This command will update all the scripts and autosubmit configuration.
- Wrapper definition has changed.
- Tasks dependencies system has changed.
- Added the parameter DELETE_WHEN_EDGELESS (boolean) to the section JOBS. This parameter allows to delete a job when it has no edges. (default TRUE)

Warning: The configuration language has changed. Please, check the new configuration file format.

Warning: The wrapper definition has changed. Please, check the new wrapper definition.

Warning: The tasks dependencies system has changed. Please, check the new tasks dependencies system.

Warning: Edgeless jobs are now deleted by default. Please, check the new parameter DELETE_WHEN_EDGELESS.

Warning: upgrade may not translate all the scripts, we recommend to revise your scripts before run AS.

6.1 Configuration changes

Now autosubmit is composed by two kind of YAML configurations, the default ones, which are the same as always, and the custom ones.

The custom ones, allows to define custom configurations that will override the default ones, in order to do this, you only have to put the key in the custom configuration file. These custom ones, can be anywhere and have any name, by default they're inside *<expid>/conf* but you can change this path in the expdef.yml file. *DEFAULT.CUSTOM_CONFIG*

Additionally, you must be aware of the following changes:

- All sections **keys** are normalized to **UPPERCASE**, while values remain as the user put. Beware of the scripts that relies on %CURRENT_HPCARCH% and variables that refer to a platform because they will be always in UPPERCASE. Normalize the script.
- To define a job, you must put them under the key *jobs* in any custom configuration file.
- To define a platform, you must put them under the key *platforms* in any custom configuration file.
- To define a loop, you must put the key "FOR" as the first key of the section.
- You can put any %placeholder% in the proj.yml and custom files, and also you can put %ROOTDIR% in the expdef.yml.
- All configuration is now based in an hierarchical structure, so to export a var, you must use the following syntax: *%KEY.SUBKEY.SUBSUBKEY%*. The same goes for override them.
- YAML has into account the type.

6.2 Examples

List of example with the new configuration and the structure as follows

```
$/autosubmit/a00q/conf$ ls
autosubmit_a00q.yml  custom_conf  expdef_a00q.yml  jobs_a00q.yml  platforms_a00q.yml
$/autosubmit/a00q/conf/custom_conf ls
more_jobs.yml
```

6.3 Configuration

autosubmit_expid.yml

```yaml
config:
  AUTOSUBMIT_VERSION: 4.0.0b
  MAXWAITINGJOBS: '3000'
  TOTALJOBS: '3000'
  SAFETYSLEEPTIME: 0
  RETRIALS: '10'
mail:
  NOTIFICATIONS: 'False'
  TO: daniel.beltran@bsc.es
```

expdef_expid.yml

```
DEFAULT:
  EXPID: a02u
  HPCARCH: local
  CUSTOM_CONFIG: "%ROOTDIR%/conf/custom_conf"
experiment:
  DATELIST: '20210811'
  MEMBERS: CompilationEfficiency HardwareBenchmarks WeakScaling StrongScaling
  CHUNKSIZEUNIT: hour
  CHUNKSIZE: '6'
  NUMCHUNKS: '2'
  CALENDAR: standard
rerun:
  RERUN: 'FALSE'
  CHUNKLIST: ''
project:
  PROJECT_TYPE: local
  PROJECT_DESTINATION: r_test
git:
  PROJECT_ORIGIN: https://earth.bsc.es/gitlab/ces/automatic_performance_
↪profiling.git
  PROJECT_BRANCH: autosubmit-makefile1
  PROJECT_COMMIT: ''
svn:
  PROJECT_URL: ''
  PROJECT_REVISION: ''
local:
  PROJECT_PATH: /home/dbeltran/r_test
project_files:
  FILE_PROJECT_CONF: ''
  FILE_JOBS_CONF: ''
```

jobs_expid.yml

```
JOBS:
  LOCAL_SETUP:
    FILE: LOCAL_SETUP.sh
    PLATFORM: LOCAL
    RUNNING: "once"
  REMOTE_SETUP:
    FILE: REMOTE_SETUP.sh
    DEPENDENCIES: LOCAL_SETUP
    WALLCLOCK: '00:05'
    RUNNING: once
    NOTIFY_ON: READY SUBMITTED QUEUING COMPLETED
  INI:
    FILE: INI.sh
    DEPENDENCIES: REMOTE_SETUP
    RUNNING: member
    WALLCLOCK: '00:05'
    NOTIFY_ON: READY SUBMITTED QUEUING COMPLETED

  SIM:
    FOR:
```

(continues on next page)

(continued from previous page)

```
      NAME: [20,40,80]
      PROCESSORS: [2,4,8]
      THREADS: [1,1,1]
      DEPENDENCIES: [INI SIM_20-1 CLEAN-2, INI SIM_40-1 CLEAN-2, INI SIM_80-1␣
↪CLEAN-2]
      NOTIFY_ON: READY SUBMITTED QUEUING COMPLETED

    FILE: SIM.sh
    DEPENDENCIES: INI SIM_20-1 CLEAN-2
    RUNNING: chunk
    WALLCLOCK: '00:05'
    TASKS: '1'
    NOTIFY_ON: READY SUBMITTED QUEUING COMPLETED

  POST:
    FOR:
      NAME: [ 20,40,80 ]
      PROCESSORS: [ 20,40,80 ]
      THREADS: [ 1,1,1 ]
      DEPENDENCIES: [ SIM_20 POST_20-1,SIM_40 POST_40-1,SIM_80 POST_80-1 ]
    FILE: POST.sh
    RUNNING: chunk
    WALLCLOCK: '00:05'
  CLEAN:
    FILE: CLEAN.sh
    DEPENDENCIES: POST_20 POST_40 POST_80
    RUNNING: chunk
    WALLCLOCK: '00:05'
  TRANSFER:
    FILE: TRANSFER.sh
    PLATFORM: LOCAL
    DEPENDENCIES: CLEAN
    RUNNING: member
```

platforms_expid.yml

```
Platforms:
  MaReNoStRuM4:
    TYPE: slurm
    HOST: bsc
    PROJECT: bsc32
    USER: bsc32070
    QUEUE: debug
    SCRATCH_DIR: /gpfs/scratch
    ADD_PROJECT_TO_HOST: False
    MAX_WALLCLOCK: '48:00'
    USER_TO: pr1enx13
    TEMP_DIR: ''
    SAME_USER: False
    PROJECT_TO: pr1enx00
    HOST_TO: bscprace
  marenostrum_archive:
```

(continues on next page)

(continued from previous page)

```
    TYPE: ps
    HOST: dt02.bsc.es
    PROJECT: bsc32
    USER: bsc32070
    SCRATCH_DIR: /gpfs/scratch
    ADD_PROJECT_TO_HOST: 'False'
    TEST_SUITE: 'False'
    USER_TO: pr1enx13
    TEMP_DIR: /gpfs/scratch/bsc32/bsc32070/test_migrate
    SAME_USER: false
    PROJECT_TO: pr1enx00
    HOST_TO: transferprace
  transfer_node:
    TYPE: ps
    HOST: dt01.bsc.es
    PROJECT: bsc32
    USER: bsc32070
    ADD_PROJECT_TO_HOST: false
    SCRATCH_DIR: /gpfs/scratch
    USER_TO: pr1enx13
    TEMP_DIR: /gpfs/scratch/bsc32/bsc32070/test_migrate
    SAME_USER: false
    PROJECT_TO: pr1enx00
    HOST_TO: transferprace
  transfer_node_bscearth000:
    TYPE: ps
    HOST: bscearth000
    USER: dbeltran
    PROJECT: Earth
    ADD_PROJECT_TO_HOST: false
    QUEUE: serial
    SCRATCH_DIR: /esarchive/scratch
    USER_TO: dbeltran
    TEMP_DIR: ''
    SAME_USER: true
    PROJECT_TO: Earth
    HOST_TO: bscpraceearth000
  bscearth000:
    TYPE: ps
    HOST: bscearth000
    USER: dbeltran
    PROJECT: Earth
    ADD_PROJECT_TO_HOST: false
    QUEUE: serial
    SCRATCH_DIR: /esarchive/scratch
  nord3:
    TYPE: SLURM
    HOST: nord1.bsc.es
    PROJECT: bsc32
    USER: bsc32070
    QUEUE: debug
    SCRATCH_DIR: /gpfs/scratch
```

(continues on next page)

(continued from previous page)

```
    MAX_WALLCLOCK: '48:00'
    USER_TO: pr1enx13
    TEMP_DIR: ''
    SAME_USER: true
    PROJECT_TO: pr1enx00
  ecmwf-xc40:
    TYPE: ecaccess
    VERSION: pbs
    HOST: cca
    USER: c3d
    PROJECT: spesiccf
    ADD_PROJECT_TO_HOST: false
    SCRATCH_DIR: /scratch/ms
    QUEUE: np
    SERIAL_QUEUE: ns
    MAX_WALLCLOCK: '48:00'
```

custom_conf/more_jobs.yml

```
jobs:
  Additional_job_1:
    FILE: extrajob.sh
    DEPENDENCIES: POST_20
    RUNNING: once
  additional_job_2:
    FILE: extrajob.sh
    RUNNING: once
```

6.4 Wrappers definition

To define a the wrappers:

```
wrappers:
  wrapper_sim20:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM_20"
  wrapper_sim40:
    TYPE: "vertical"
    JOBS_IN_WRAPPER: "SIM_40"
```

6.5 Loops definition

To define a loop, you need to use the FOR key and also the NAME key.

In order to generate the following jobs:

```
experiment:
  DATELIST: 19600101
  MEMBERS: "00"
```

(continues on next page)

(continued from previous page)

```
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '2'
  CALENDAR: standard
JOBS:
  POST_20:

    DEPENDENCIES:
      POST_20:
      SIM_20:
    FILE: POST.sh
    PROCESSORS: '20'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  POST_40:

    DEPENDENCIES:
      POST_40:
      SIM_40:
    FILE: POST.sh
    PROCESSORS: '40'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  POST_80:

    DEPENDENCIES:
      POST_80:
      SIM_80:
    FILE: POST.sh
    PROCESSORS: '80'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  SIM_20:

    DEPENDENCIES:
      SIM_20-1:
    FILE: POST.sh
    PROCESSORS: '20'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
  SIM_40:

    DEPENDENCIES:
      SIM_40-1:
    FILE: POST.sh
    PROCESSORS: '40'
    RUNNING: chunk
    THREADS: '1'
```

(continues on next page)

(continued from previous page)

```
    WALLCLOCK: 00:05
  SIM_80:

    DEPENDENCIES:
      SIM_80-1:
    FILE: POST.sh
    PROCESSORS: '80'
    RUNNING: chunk
    THREADS: '1'
    WALLCLOCK: 00:05
```

One can use now the following configuration:

```
experiment:
  DATELIST: 19600101
  MEMBERS: "00"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '2'
  CALENDAR: standard
JOBS:
  SIM:
    FOR:
      NAME: [ 20,40,80 ]
      PROCESSORS: [ 20,40,80 ]
      THREADS: [ 1,1,1 ]
      DEPENDENCIES: [ SIM_20-1,SIM_40-1,SIM_80-1 ]
    FILE: POST.sh
    RUNNING: chunk
    WALLCLOCK: '00:05'
  POST:
      FOR:
        NAME: [ 20,40,80 ]
        PROCESSORS: [ 20,40,80 ]
        THREADS: [ 1,1,1 ]
        DEPENDENCIES: [ SIM_20 POST_20,SIM_40 POST_40,SIM_80 POST_80 ]
      FILE: POST.sh
      RUNNING: chunk
      WALLCLOCK: '00:05'
```

Warning: Only the parameters that changes must be included inside the *FOR* key.

6.6 Dependencies rework

The DEPENDENCIES key is used to define the dependencies of a job. It can be used in the following ways:

- Basic: The dependencies are a list of jobs, separated by " ", that runs before the current task is submitted.
- New: The dependencies is a list of YAML sections, separated by "n", that runs before the current job is submitted.
  - For each dependency section, you can designate the following keywords to control the current job-affected tasks:
    - DATES_FROM: Selects the job dates that you want to alter.
    - MEMBERS_FROM: Selects the job members that you want to alter.
    - CHUNKS_FROM: Selects the job chunks that you want to alter.
  - For each dependency section and *_FROM keyword, you can designate the following keywords to control the destination of the dependency:
    - DATES_TO: Links current selected tasks to the dependency tasks of the dates specified.
    - MEMBERS_TO: Links current selected tasks to the dependency tasks of the members specified.
    - CHUNKS_TO: Links current selected tasks to the dependency tasks of the chunks specified.
  - Important keywords for [DATES|MEMBERS|CHUNKS]_TO:
    - "natural": Will keep the default linkage. Will link if it would be normally. Example, SIM_FC00_CHUNK_1 -> DA_FC00_CHUNK_1.
    - "all": Will link all selected tasks of the dependency with current selected tasks. Example, SIM_FC00_CHUNK_1 -> DA_FC00_CHUNK_1, DA_FC00_CHUNK_2, DA_FC00_CHUNK_3…
    - "none": Will unlink selected tasks of the dependency with current selected tasks.

For the new format, consider that the priority is hierarchy and goes like this DATES_FROM -(includes)-> MEMBERS_FROM -(includes)-> CHUNKS_FROM.

- You can define a DATES_FROM inside the DEPENDENCY.
- You can define a MEMBERS_FROM inside the DEPENDENCY and DEPENDENCY.DATES_FROM.
- You can define a CHUNKS_FROM inside the DEPENDENCY, DEPENDENCY.DATES_FROM, DEPENDENCY.MEMBERS_FROM, DEPENDENCY.DATES_FROM.MEMBERS_FROM

For the examples, we will consider that our experiment has the following configuration:

```
EXPERIMENT:
    DATELIST: 20220101
    MEMBERS: FC1 FC2
    NUMCHUNKS: 4
```

6.7 Basic

```
JOBS:
  JOB_1:
      FILE: job1.sh
      RUNNING: chunk
  JOB_2:
      FILE: job2.sh
      DEPENDENCIES: JOB_1
      RUNNING: chunk
  JOB_3:
      FILE: job3.sh
      DEPENDENCIES: JOB_2
      RUNNING: chunk
  SIM:
      FILE: sim.sh
      DEPENDENCIES: JOB_3 SIM-1
      RUNNING: chunk
  POST:
      FILE: post.sh
      DEPENDENCIES: SIM
      RUNNING: chunk
  TEST:
      FILE: test.sh
      DEPENDENCIES: POST
      RUNNING: chunk
```

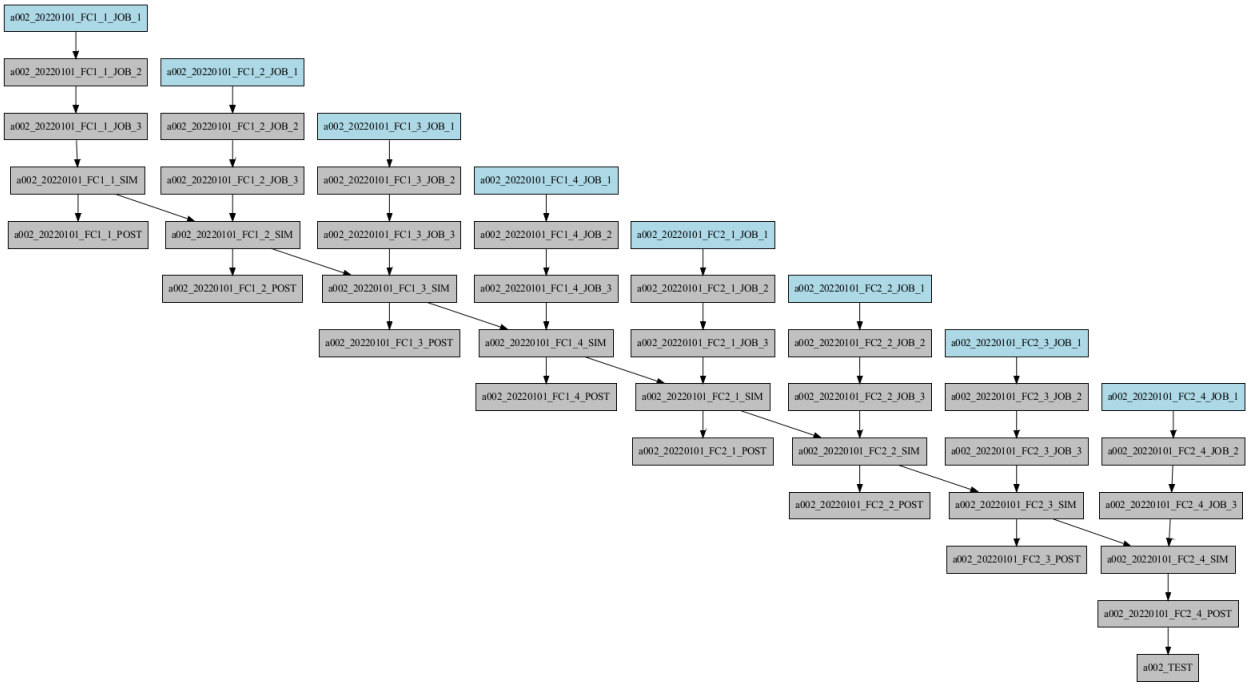
6.8 New format

```
JOBS:
  JOB_1:
      FILE: job1.sh
      RUNNING: chunk
  JOB_2:
      FILE: job2.sh
      DEPENDENCIES:
          JOB_1:
              dates_to: "natural"
              members_to: "natural"
              chunks_to: "natural"
      RUNNING: chunk
  JOB_3:
      FILE: job3.sh
      DEPENDENCIES:
          JOB_2:
              dates_to: "natural"
              members_to: "natural"
              chunks_to: "natural"
      RUNNING: chunk
  SIM:
```

(continues on next page)

(continued from previous page)

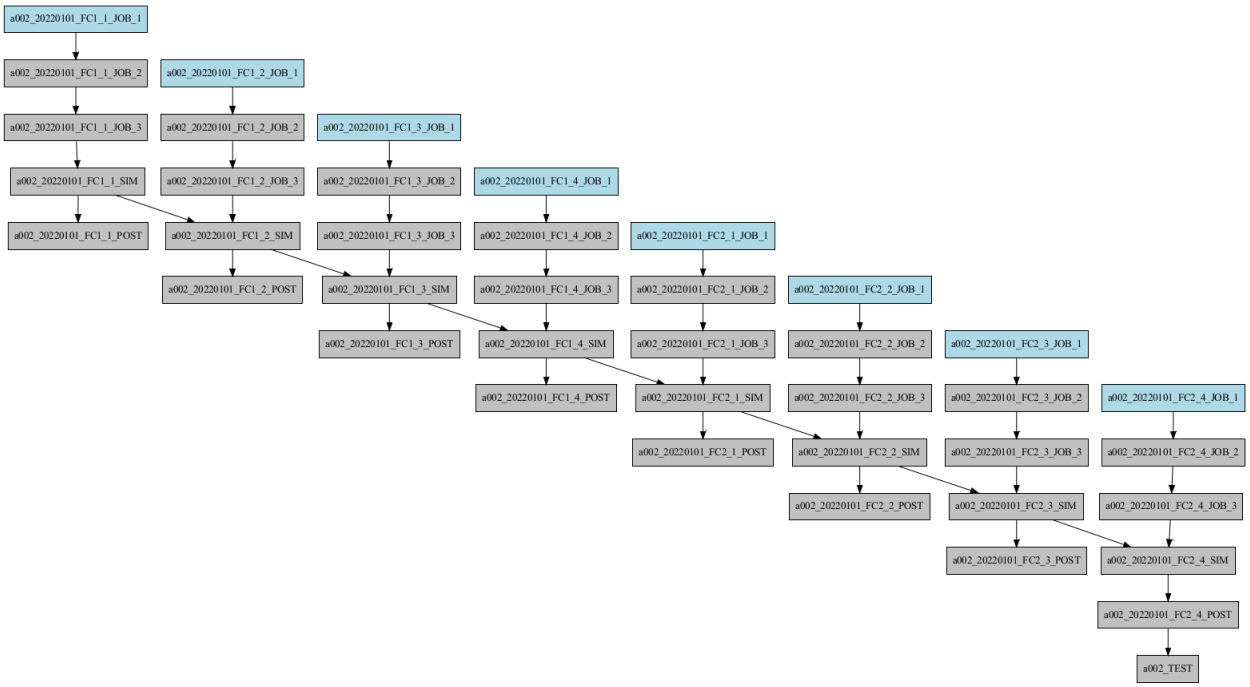
```
    FILE: sim.sh
    DEPENDENCIES:
        JOB_3:
            dates_to: "natural"
            members_to: "natural"
            chunks_to: "natural"
        SIM-1:
            dates_to: "natural"
            members_to: "natural"
            chunks_to: "natural"
    RUNNING: chunk
POST:
    FILE: post.sh
    DEPENDENCIES:
        SIM:
            dates_to: "natural"
            members_to: "natural"
            chunks_to: "natural"
    RUNNING: chunk
TEST:
    FILE: test.sh
    DEPENDENCIES:
        POST:
            dates_to: "natural"
            members_to: "natural"
            chunks_to: "natural"
    RUNNING: chunk
```



6.8.1 Example 1: New format with specific dependencies

In the following example, we want to launch the next member SIM after the last SIM chunk of the previous member is finished.

```
JOBS:
    JOB_1:
        FILE: job1.sh
        RUNNING: chunk
    JOB_2:
        FILE: job2.sh
        DEPENDENCIES:
            JOB_1:
        RUNNING: chunk
    JOB_3:
        FILE: job3.sh
        DEPENDENCIES:
            JOB_2:
        RUNNING: chunk
    SIM:
        FILE: sim.sh
        DEPENDENCIES:
            JOB_3:
            SIM-1:
            SIM:
                MEMBERS_FROM:
                  FC2:
                    CHUNKS_FROM:
                     1:
                      dates_to: "all"
                      members_to: "FC1"
                      chunks_to: "4"
        RUNNING: chunk
    POST:
        FILE: post.sh
        DEPENDENCIES:
            SIM:
        RUNNING: chunk
    TEST:
        FILE: test.sh
        DEPENDENCIES:
            POST:
              members_to: "FC2"
              chunks_to: 4
        RUNNING: once
```



6.8.2 Example 2: Crossdate wrappers using the the new dependencies

```
experiment:
  DATELIST: 20120101 20120201
  MEMBERS: "000 001"
  CHUNKSIZEUNIT: day
  CHUNKSIZE: '1'
  NUMCHUNKS: '3'
wrappers:
    wrapper_simda:
        TYPE: "horizontal-vertical"
        JOBS_IN_WRAPPER: "SIM DA"

JOBS:
  LOCAL_SETUP:
    FILE: templates/local_setup.sh
    PLATFORM: marenostrum_archive
    RUNNING: once
    NOTIFY_ON: COMPLETED
  LOCAL_SEND_SOURCE:
    FILE: templates/01_local_send_source.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
    NOTIFY_ON: FAILED
  LOCAL_SEND_STATIC:
    FILE: templates/01b_local_send_static.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP
    RUNNING: once
```

(continues on next page)

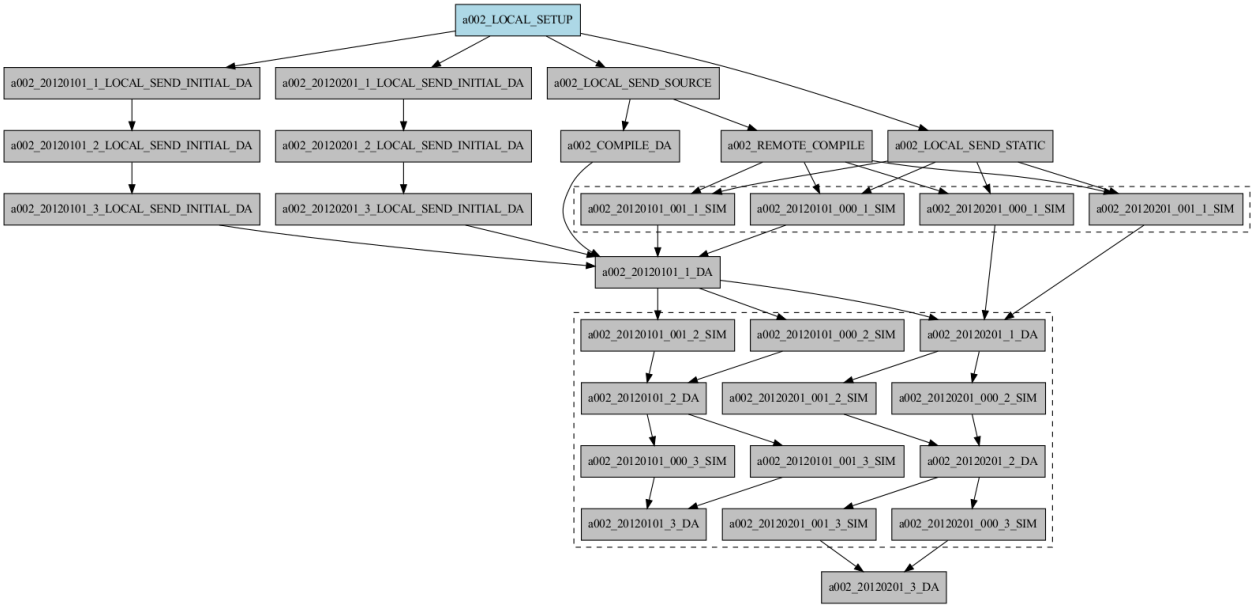
(continued from previous page)

```
    NOTIFY_ON: FAILED
  REMOTE_COMPILE:
    FILE: templates/02_compile.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    PROCESSORS: '4'
    WALLCLOCK: 00:50
    NOTIFY_ON: COMPLETED
  SIM:
    FILE: templates/05b_sim.sh
    DEPENDENCIES:
      LOCAL_SEND_STATIC:
      REMOTE_COMPILE:
      SIM-1:
      DA-1:
    RUNNING: chunk
    PROCESSORS: '68'
    WALLCLOCK: 00:12
    NOTIFY_ON: FAILED
  LOCAL_SEND_INITIAL_DA:
    FILE: templates/00b_local_send_initial_DA.sh
    PLATFORM: marenostrum_archive
    DEPENDENCIES: LOCAL_SETUP LOCAL_SEND_INITIAL_DA-1
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
  COMPILE_DA:
    FILE: templates/02b_compile_da.sh
    DEPENDENCIES: LOCAL_SEND_SOURCE
    RUNNING: once
    WALLCLOCK: 00:20
    NOTIFY_ON: FAILED
  DA:
    FILE: templates/05c_da.sh
    DEPENDENCIES:
      SIM:
      LOCAL_SEND_INITIAL_DA:
        CHUNKS_TO: "all"
        DATES_TO: "all"
        MEMBERS_TO: "all"
      COMPILE_DA:
      DA:
        DATES_FROM:
          "20120201":
            CHUNKS_FROM:
              1:
                DATES_TO: "20120101"
                CHUNKS_TO: "1"
    RUNNING: chunk
    SYNCHRONIZE: member
    DELAY: '0'
    WALLCLOCK: 00:12
```

(continues on next page)

(continued from previous page)

```
    PROCESSORS: '256'
    NOTIFY_ON: FAILED
```



CHAPTER

SEVEN

TROUBLESHOOTING

7.1 How to change the job status stopping autosubmit

Review *How to change the job status*.

7.2 How to change the job status without stopping autosubmit

Review *How to change the job status without stopping autosubmit*.

7.3 My project parameters are not being substituted in the templates

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

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

7.4 Unable to recover remote logs files.

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

7.5 Error on create caused by a configuration parsing error

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

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

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

7.6 Other possible errors

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

Explanation: The latest version of autosubmit uses a database to efficiently track changes in the jobs of your experiment. It could have happened that this small database got corrupted.

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

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

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

7.7 Error codes

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

7.8 Changelog

review *Changelog*.

CHAPTER

EIGHT

API

8.1 autosubmit

class autosubmit.autosubmit.**Autosubmit**

Bases: object

Interface class for autosubmit.

static archive(*expid*, *noclean=True*, *uncompress=True*, *rocrate=False*)

Archives an experiment: call clean (if experiment is of version 3 or later), compress folder to tar.gz and moves to year's folder

Parameters

- **expid** (*str*) – experiment identifier
- **noclean** (*bool*) – flag telling it whether to clean the experiment or not.
- **uncompress** (*bool*) – flag telling it whether to decompress or not.
- **rocrate** (*bool*) – flag to enable RO-Crate

Returns

True if the experiment has been successfully archived. False otherwise.

Return type

bool

static as_conf_default_values(*exp_id*, *hpc='local'*, *minimal_configuration=False*, *git_repo=''*, *git_branch='main'*, *git_as_conf=''*)

Replace default values in as_conf files :param exp_id: experiment id :param hpc: platform :param minimal_configuration: minimal configuration :param git_repo: path to project git repository :param git_branch: main branch :param git_as_conf: path to as_conf file in git repository :return: None

static cat_log(*exp_or_job_id: str*, *file: Union[None, str]*, *mode: Union[None, str]*, *inspect: bool = False*) → bool

The cat-log command allows users to view Autosubmit logs using the command-line.

It is possible to use autosubmit cat-log for Workflow and for Job logs. It decides whether to show Workflow or Job logs based on the ID given. Shorter ID's, such as a000` are considered Workflow ID's, so it will display logs for that workflow. For longer ID's, such as ``a000_20220401_fc0_1_GSV, the command will display logs for that specific job.

Users can choose the log file using the FILE parameter, to display an error or output log file, for instance.

Finally, the MODE parameter allows users to choose whether to display the complete file contents (similar to the cat command) or to start tailing its output (akin to tail -f).

Args:

exp_or_job_id: A workflow or job ID. file: the type of the file to be printed (not the file path!). mode: the mode to print the file (e.g. cat, tail). inspect: when True it will use job files in tmp/ instead of tmp/LOG_a000/.

`static change_status`(*final*, *final_status*, *job*, *save*)

Set job status to final

Parameters

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

`static check`(*experiment_id*, *notransitive=False*)

Checks experiment configuration and warns about any detected error or inconsistency.

Parameters

- **notransitive** –
- **experiment_id** (*str*) – experiment identifier:

`static check_wrapper_stored_status`(*as_conf*, *job_list*)

Check if the wrapper job has been submitted and the inner jobs are in the queue. :param as_conf: a BasicConfig object :param job_list: a JobList object :return: JobList object updated

`static check_wrappers`(*as_conf*, *job_list*, *platforms_to_test*, *expid*)

Check wrappers and inner jobs status also order the non-wrapped jobs to be submitted by active platforms :param as_conf: a AutosubmitConfig object :param job_list: a JobList object :param platforms_to_test: a list of Platform :param expid: a string with the experiment id :return: non-wrapped jobs to check and a dictionary with the changes in the jobs status

`static clean`(*expid*, *project*, *plot*, *stats*)

Clean experiment's directory to save storage space. It removes project directory and outdated plots or stats.

Parameters

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

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

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

Parameters

- **advanced** –
- **database_path** (*str*) – path to autosubmit database
- **database_filename** (*str*) – database filename
- **local_root_path** (*str*) – path to autosubmit's experiments' directory

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

static configure_dialog()

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

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

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

Parameters

- **detail** –
- **check_wrappers** –
- **notransitive** –
- **expand_status** –
- **expand** –
- **group_by** –
- **expid** (*str*) – experiment identifier
- **noplot** (*bool*) – if True, method omits final plotting of the jobs list. Only needed on large experiments when plotting time can be much larger than creation time.
- **hide** (*bool*) – hides plot window
- **hide** – hides plot window
- **output** (*str*) – plot's file format. It can be pdf, png, ps or svg

Returns

True if successful, False if not

Return type

bool

static database_fix(*expid*)

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

Parameters

expid (*str*) – experiment identifier

Returns

Return type

static delete(*expid*, *force*)

Deletes and experiment from database and experiment's folder

Parameters

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

Returns
True if successful, False if not

Return type
bool

static describe(*input_experiment_list='*'*, *get_from_user=''*)

Show details for specified experiment

Parameters

- **experiments_id** (*str*) – experiments identifier:
- **get_from_user** (*str*) – user to get the experiments from

Returns
str,str,str,str

static environ_init()

Initialise AS environment.

property experiment_data

Get the current voltage.

static expid(*description*, *hpc=''*, *copy_id=''*, *dummy=False*, *minimal_configuration=False*, *git_repo=''*, *git_branch=''*, *git_as_conf=''*, *operational=False*, *testcase=False*, *use_local_minimal=False*)

Creates a new experiment for given HPC description: description of the experiment hpc: HPC where the experiment will be executed copy_id: if specified, experiment id to copy dummy: if true, creates a dummy experiment minimal_configuration: if true, creates a minimal configuration git_repo: git repository to clone git_branch: git branch to clone git_as_conf: path to as_conf file in git repository operational: if true, creates an operational experiment local: Gets local minimal instead of git minimal

static generate_as_config(*exp_id: str*, *dummy: bool = False*, *minimal_configuration: bool = False*, *local: bool = False*, *parameters: Optional[Dict[str, Union[Dict, List, str]]] = None*) → None

Retrieve the configuration from autosubmitconfigparser package.

Parameters

- **exp_id** – Experiment ID
- **dummy** – Whether the experiment is a dummy one or not.
- **minimal_configuration** – Whether the experiment is configured with minimal configuration or not.
- **local** – Whether the experiment project type is local or not.
- **parameters** – Optional list of parameters to be used when processing the configuration files.

Returns
None

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

Parameters

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

Returns
Nothing

Return type

static get_historical_database(*expid*, *job_list*, *as_conf*)

Get the historical database for the experiment :param expid: a string with the experiment id :param job_list: a JobList object :param as_conf: a AutosubmitConfig object :return: an experiment history object

static get_iteration_info(*as_conf*, *job_list*)

Prints the current iteration information :param as_conf: autosubmit configuration object :param job_list: job list object :return: common parameters for the iteration

static inspect(*expid*, *lst*, *filter_chunks*, *filter_status*, *filter_section*, *notransitive=False*, *force=False*, *check_wrapper=False*, *quick=False*)

Generates cmd files experiment.

Parameters

- **check_wrapper** –
- **force** –
- **notransitive** –
- **filter_section** –
- **filter_status** –
- **filter_chunks** –
- **lst** –
- **expid** (*str*) – identifier of experiment to be run

Returns
True if run to the end, False otherwise

Return type
bool

static install()

Creates a new database instance for autosubmit at the configured path

static migrate(*experiment_id*, *offer*, *pickup*, *only_remote*)

Migrates experiment files from current to other user. It takes mapping information for new user from config files.

Parameters

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

static monitor(*expid*, *file_format*, *lst*, *filter_chunks*, *filter_status*, *filter_section*, *hide*, *txt_only=False*, *group_by=None*, *expand=''*, *expand_status=[]*, *hide_groups=False*, *notransitive=False*, *check_wrapper=False*, *txt_logfiles=False*, *profile=False*, *detail=False*)

Plots workflow graph for a given experiment with status of each job coded by node color. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

- **txt_logfiles** –
- **expid** (*str*) – identifier of the experiment to plot
- **file_format** (*str*) – plot's file format. It can be pdf, png, ps or svg
- **lst** (*str*) – list of jobs to change status
- **filter_chunks** (*str*) – chunks to change status
- **filter_status** (*str*) – current status of the jobs to change status
- **filter_section** (*str*) – sections to change status
- **hide** (*bool*) – hides plot window
- **txt_only** (*bool*) – workflow will only be written as text
- **group_by** (*bool*) – workflow will only be written as text
- **expand** (*str*) – Filtering of jobs for its visualization
- **expand_status** (*str*) – Filtering of jobs for its visualization
- **hide_groups** (*bool*) – Simplified workflow illustration by encapsulating the jobs.
- **notransitive** (*bool*) – workflow will only be written as text
- **check_wrapper** (*bool*) – Shows a preview of how the wrappers will look
- **notransitive** – Some dependencies will be omitted
- **detail** (*bool*) – better text format representation but more expensive

static parse_args()

Parse arguments given to an executable and start execution of command given

static pkl_fix(*expid*)

Tries to find a backup of the pkl file and restores it. Verifies that autosubmit is not running on this experiment.

Parameters

expid (*str*) – experiment identifier

Returns

Return type

static prepare_run(*expid*, *notransitive=False*, *start_time=None*, *start_after=None*, *run_only_members=None*, *recover=False*)

Prepare the run of the experiment. :param expid: a string with the experiment id. :param notransitive: a boolean to indicate for the experiment to not use transitive dependencies. :param start_time: a string with the starting time of the experiment. :param start_after: a string with the experiment id to start after. :param run_only_members: a string with the members to run. :param recover: a boolean to indicate if the experiment is recovering from a failure. :return: a tuple

static process_historical_data_iteration(*job_list*, *job_changes_tracker*, *expid*)

Process the historical data for the current iteration. :param job_list: a JobList object. :param job_changes_tracker: a dictionary with the changes in the job status. :param expid: a string with the experiment id. :return: an ExperimentHistory object.

static recovery(*expid*, *noplot*, *save*, *all_jobs*, *hide*, *group_by=None*, *expand=[]*, *expand_status=[]*, *notransitive=False*, *no_recover_logs=False*, *detail=False*, *force=False*)

Method to check all active jobs. If COMPLETED file is found, job status will be changed to COMPLETED, otherwise it will be set to WAITING. It will also update the jobs list.

Parameters

- **detail** –
- **no_recover_logs** –
- **notransitive** –
- **expand_status** –
- **expand** –
- **group_by** –
- **noplot** –
- **expid** (*str*) – identifier of the experiment to recover
- **save** (*bool*) – If true, recovery saves changes to the jobs list
- **all_jobs** (*bool*) – if True, it tries to get completed files for all jobs, not only active.
- **hide** (*bool*) – hides plot window
- **force** (*bool*) – Allows to restore the workflow even if there are running jobs

static refresh(*expid*, *model_conf*, *jobs_conf*)

Refresh project folder for given experiment

Parameters

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

static report(*expid*, *template_file_path=''*, *show_all_parameters=False*, *folder_path=''*, *placeholders=False*)

Show report for specified experiment :param expid: experiment identifier :type expid: str :param template_file_path: path to template file :type template_file_path: str :param show_all_parameters: show all parameters :type show_all_parameters: bool :param folder_path: path to folder :type folder_path: str :param placeholders: show placeholders :type placeholders: bool

static rerun_recovery(*expid*, *job_list*, *rerun_list*, *as_conf*)

Method to check all active jobs. If COMPLETED file is found, job status will be changed to COMPLETED, otherwise it will be set to WAITING. It will also update the jobs list.

Parameters

- **expid** (*str*) – identifier of the experiment to recover
- **job_list** (JobList) – job list to update
- **rerun_list** (*list*) – list of jobs to rerun
- **as_conf** (AutosubmitConfig) – AutosubmitConfig object

Returns

static rocrate(*expid*, *path: Path*)

Produces an RO-Crate archive for an Autosubmit experiment.

Parameters

- **expid** (*str*) – experiment ID
- **path** (*Path*) – path to save the RO-Crate in

Returns
`True` if successful, `False` otherwise

Return type
bool

static run_experiment(*expid*, *notransitive=False*, *start_time=None*, *start_after=None*, *run_only_members=None*, *profile=False*)

Runs and experiment (submitting all the jobs properly and repeating its execution in case of failure). :param expid: the experiment id :param notransitive: if True, the transitive closure of the graph is not computed :param start_time: the time at which the experiment should start :param start_after: the expid after which the experiment should start :param run_only_members: the members to run :param profile: if True, the function will be profiled :return: None

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

Set status of jobs :param expid: experiment id :param noplot: do not plot :param save: save :param final: final status :param filter_list: list of jobs :param filter_chunks: filter chunks :param filter_status: filter status :param filter_section: filter section :param filter_type_chunk: filter type chunk :param filter_chunk_split: filter chunk split :param hide: hide :param group_by: group by :param expand: expand :param expand_status: expand status :param notransitive: notransitive :param check_wrapper: check wrapper :param detail: detail :return:

static statistics(*expid*, *filter_type*, *filter_period*, *file_format*, *hide*, *notransitive=False*)

Plots statistics graph for a given experiment. Plot is created in experiment's plot folder with name <expid>_<date>_<time>.<file_format>

Parameters

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

- **hide** (*bool*) – hides plot window
- **notransitive** – Reduces workflow linkage complexity

static submit_ready_jobs(*as_conf: AutosubmitConfig*, *job_list: JobList*, *platforms_to_test: Set[Platform]*, *packages_persistence: JobPackagePersistence*, *inspect: bool = False*, *only_wrappers: bool = False*, *hold: bool = False*) → bool

Gets READY jobs and send them to the platforms if there is available space on the queues

Parameters

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

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

Return type
Boolean

static test(*expid*, *chunks*, *member=None*, *start_date=None*, *hpc=None*, *branch=None*)

Method to conduct a test for a given experiment. It creates a new experiment for a given experiment with a given number of chunks with a random start date and a random member to be run on a random HPC.

Parameters

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

Returns
True if test was successful, False otherwise

Return type
bool

static testcase(*description*, *chunks=None*, *member=None*, *start_date=None*, *hpc=None*, *copy_id=None*, *minimal_configuration=False*, *git_repo=None*, *git_branch=None*, *git_as_conf=None*, *use_local_minimal=False*)

Method to conduct a test for a given experiment. It creates a new experiment for a given experiment with a given number of chunks with a random start date and a random member to be run on a random HPC. :param description: description of the experiment :type description: str :param chunks: number of chunks to be run by the experiment :type chunks: int :param member: member to be used by the test. If None, a random member will be chosen :type member: str :param start_date: start date of the experiment. If None, a random start date will be chosen :type start_date: str :param hpc: HPC to be used by the test. If None, a random HPC will be chosen :type hpc: str :param copy_id: copy id to be used by the test. If None, a random copy id will be chosen :type copy_id: str :param minimal_configuration: if True, the experiment will be run with a minimal configuration :type minimal_configuration: bool :param git_repo: git repository to be used by the test. If None, a random git repository will be chosen :type git_repo: str :param git_branch: git branch to be used by the test. If None, a random git branch will be chosen :type git_branch: str :param git_as_conf: git autosubmit configuration to be used by the test. If None, a random git autosubmit configuration will be chosen :type git_as_conf: str :param use_local_minimal: if True, the experiment will be run with a local minimal configuration :type use_local_minimal: bool :return: experiment identifier :rtype: str

static unarchive(*experiment_id*, *uncompressed=True*, *rocrate=False*)

Unarchives an experiment: uncompress folder from tar.gz and moves to experiment root folder

Parameters

- **experiment_id** (*str*) – experiment identifier
- **uncompressed** (*bool*) – if True, the tar file is uncompressed
- **rocrate** (*bool*) – flag to enable RO-Crate

static update_version(*expid*)

Refresh experiment version with the current autosubmit version :param expid: experiment identifier :type expid: str

class autosubmit.autosubmit.**MyParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*, *exit_on_error=True*)

Bases: ArgumentParser

add_argument(*dest*, ..., *name=value*, ...)

add_argument(*option_string*, *option_string*, ..., *name=value*, ...) → None

error(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

autosubmit.autosubmit.**signal_handler**(*signal_received*, *frame*)

Used to handle interrupt signals, allowing autosubmit to clean before exit

Parameters

- **signal_received** –
- **frame** –

autosubmit.autosubmit.**signal_handler_create**(*signal_received*, *frame*)

Used to handle KeyboardInterrupt signals while the create method is being executed

Parameters

- **signal_received** –

- **frame** –

8.2 autosubmit.config

8.2.1 autosubmitconfigparser.config.basicConfig

class autosubmitconfigparser.config.basicconfig.**BasicConfig**

Bases: object

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.

8.2.2 autosubmitconfigparser.config.config_common

class autosubmitconfigparser.config.configcommon.**AutosubmitConfig**(*expid*, *basic_config=<class 'autosubmitconfigparser.config.basicconfig.BasicConfig'>*, *parser_factory=<autosubmitconfigparser.config.object>*)

Bases: object

Class to handle experiment configuration coming from file or database

Parameters
expid (*str*) – experiment identifier

check_autosubmit_conf(*no_log=False*)

Checks experiment's autosubmit configuration file. :param refresh: True if the function is called during the refresh of the program :type refresh: bool :param no_log: True if the function is called during describe :type no_log: bool :return: True if everything is correct, False if it founds any error :rtype: bool

check_conf_files(*running_time=False*, *force_load=True*, *no_log=False*)

Checks configuration files (autosubmit, experiment jobs and platforms), looking for invalid values, missing required options. Print results in log :param running_time: True if the function is called during the execution of the program :type running_time: bool :param force_load: True if the function is called during the first load of the program :type force_load: bool :param refresh: True if the function is called during the refresh of the program :type refresh: bool :param no_log: True if the function is called during describe :type no_log: bool :return: True if everything is correct, False if it finds any error :rtype: bool

check_dict_keys_type(*parameters*)

Check if keys are plain into 1 dimension, checks for 33% of dict to ensure it. :param parameters: experiment parameters :return:

check_expdef_conf(*no_log=False*)

Checks experiment's experiment configuration file. :param refresh: if True, it doesn't check the mandatory parameters :type refresh: bool :param no_log: if True, it doesn't print any log message :type no_log: bool :return: True if everything is correct, False if it founds any error :rtype: bool

check_jobs_conf(*no_log=False*)

Checks experiment's jobs configuration file. :param no_log: if True, it doesn't print any log message :type no_log: bool :return: True if everything is correct, False if it founds any error :rtype: bool

check_platforms_conf(*no_log=False*)

Checks experiment's platforms configuration file.

check_wrapper_conf(*wrappers={}*, *no_log=False*)

Checks wrapper config file

Parameters

- **wrappers** –
- **no_log** –

Returns

clean_dynamic_variables(*pattern*, *in_the_end=False*)

Clean dynamic variables :param pattern: :param in_the_end: :return:

convert_list_to_string(*data*)

Convert a list to a string

deep_add_missing_starter_conf(*experiment_data*, *starter_conf*)

Add the missing keys from starter_conf to experiment_data :param experiment_data: :param starter_conf: :return:

deep_normalize(*data*)

normalize a nested dictionary or similar mapping to uppercase. Modify `source` in place.

deep_parameters_export(*data*)

Export all variables of this experiment. Resultant format will be Section.{subsections1...subsectionN} = Value. In other words, it plain the dictionary into one level

deep_read_loops(*data*, *for_keys=[]*, *long_key=''*)

Update a nested dictionary or similar mapping. Modify `source` in place.

deep_update(*unified_config*, *new_dict*)

Update a nested dictionary or similar mapping. Modify `source` in place.

detailed_deep_diff(*current_data*, *last_run_data*, *level=0*)

Returns a dictionary with for each key, the difference between the current configuration and the last_run_data :param current_data: dictionary with the current data :param last_run_data: dictionary with the last_run_data data :return: differences: dictionary

file_modified(*file*, *prev_mod_time*)

Function to check if a file has been modified. :param file: path :return: bool,new_time

get_chunk_ini(*default=1*)

Returns the first chunk from where the experiment will start

Parameters
default –

Returns
initial chunk

Return type
int

get_chunk_size(*default=1*)

Chunk Size as defined in the expdef file.

Returns
Chunksize, 1 as default.

Return type
int

get_chunk_size_unit()

Unit for the chunk length

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

Return type
str

get_communications_library()

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

Returns
communications library

Return type
str

get_copy_remote_logs()

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

Returns
if logs local copy

Return type
str

get_current_host(*section*)

Returns the user to be changed from platform config file.

Returns
migrate user to

Return type
str

get_current_project(*section*)

Returns the project to be changed from platform config file.

Returns
migrate user to

Return type
str

get_current_user(*section*)

Returns the user to be changed from platform config file.

Returns
migrate user to

Return type
str

`get_custom_directives`(*section*)

Gets custom directives needed for the given job type :param section: job type :type section: str :return: custom directives needed :rtype: str

`get_date_list`()

Returns startdates list from experiment's config file

Returns
experiment's startdates

Return type
list

`get_default_job_type`()

Returns the default job type from experiment's config file

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

Return type
str

`get_delay_retry_time`()

Returns delay time from autosubmit's config file

Returns
safety sleep time

Return type
int

`get_dependencies`(*section='None'*)

Returns dependencies list from jobs config file

Returns
experiment's members

Return type
list

`get_disable_recovery_threads`(*section*)

Returns FALSE/TRUE :return: recovery_threads_option :rtype: str

`get_export`(*section*)

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

`get_extensible_wallclock`(*wrapper={}*)

Gets extend_wallclock for the given wrapper

Parameters
wrapper (*dict*) – wrapper

Returns
extend_wallclock

Return type
int

`get_fetch_single_branch()`

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

Return config as json object

`get_git_project_branch()`

Returns git branch from experiment's config file

Returns
git branch

Return type
str

`get_git_project_commit()`

Returns git commit from experiment's config file

Returns
git commit

Return type
str

`get_git_project_origin()`

Returns git origin from experiment config file

Returns
git origin

Return type
str

`get_git_remote_project_root()`

Returns remote machine ROOT PATH

Returns
git commit

Return type
str

`get_local_project_path()`

Gets path to origin for local project

Returns
path to local project

Return type
str

`get_mails_to()`

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

Returns
mail address

Return type
[str]

`get_max_processors()`

Returns max processors from autosubmit's config file

Return type
str

`get_max_waiting_jobs()`

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

Returns
main platforms

Return type
int

`get_max_wallclock()`

Returns max wallclock

Return type
str

`get_max_wrapped_jobs(wrapper={})`

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

Returns
maximum number of jobs (or total jobs)

Return type
int

`get_max_wrapped_jobs_horizontal(wrapper={})`

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

Returns
maximum number of jobs (or total jobs)

Return type
int

`get_max_wrapped_jobs_vertical(wrapper={})`

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

Returns
maximum number of jobs (or total jobs)

Return type
int

`get_member_list`(*run_only=False*)

Returns members list from experiment's config file

Returns
experiment's members

Return type
list

`get_memory`(*section*)

Gets memory needed for the given job type :param section: job type :type section: str :return: memory needed :rtype: str

`get_memory_per_task`(*section*)

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

`get_migrate_duplicate`(*section*)

Returns the user to change to from platform config file.

Returns
migrate user to

Return type
str

`get_migrate_host_to`(*section*)

Returns the host to change to from platform config file.

Returns
host_to

Return type
str

`get_migrate_project_to`(*section*)

Returns the project to change to from platform config file.

Returns
migrate project to

Return type
str

`get_migrate_user_to`(*section*)

Returns the user to change to from platform config file.

Returns
migrate user to

Return type
str

get_min_wrapped_jobs(*wrapper={}*)

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

Returns
minim number of jobs (or total jobs)

Return type
int

get_min_wrapped_jobs_horizontal(*wrapper={}*)

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

Returns
maximum number of jobs (or total jobs)

Return type
int

get_min_wrapped_jobs_vertical(*wrapper={}*)

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

Returns
maximum number of jobs (or total jobs)

Return type
int

get_notifications()

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

Returns
if notifications

Return type
string

get_notifications_crash()

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

Returns
if notifications

Return type
string

get_num_chunks()

Returns number of chunks to run for each member

Returns
number of chunks

Return type
int

get_output_type()

Returns default output type, pdf if none

Returns
output type

Return type
string

`get_parse_two_step_start()`

Returns two-step start jobs

Returns
jobs_list

Return type
str

`static get_parser(parser_factory, file_path)`

Gets parser for given file

Parameters

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

Returns
parser

Return type
YAMLParser

`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_submodules_depth()

Returns the max depth of submodule at the moment of cloning Default is -1 (no limit) :return: depth :rtype: list

get_project_type()

Returns project type from experiment config file

Returns
project type

Return type
str

get_remote_dependencies()

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

Returns
if remote dependencies

Return type
string

get_rerun()

Returns startdates list from experiment's config file

Returns
rerun value

Return type
bool

get_rerun_jobs()

Returns rerun jobs

Returns
jobs_list

Return type
str

get_retrials()

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

Returns
safety sleep time

Return type
int

get_safetysleeptime()

Returns safety sleep time from autosubmit's config file

Returns
safety sleep time

Return type
int

get_scratch_free_space(*section*)

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

get_section(*section*, *d_value=''*, *must_exists=False*)

Gets any section if it exists within the dictionary, else returns None or error if must exist. :param section: section to get :type section: list :param d_value: default value to return if section does not exist :type d_value: str :param must_exists: if true, error is raised if section does not exist :type must_exists: bool :return: section value :rtype: str

get_storage_type()

Returns the storage system from autosubmit's config file. Pkl by default.

Returns
communications library

Return type
str

get_submodules_list() → Union[List[str], bool]

Returns submodules list from experiment's config file. Default is –recursive. Can be disabled by setting the configuration key to `False`. :return: submodules to load :rtype: Union[List[str], bool]

get_svn_project_revision()

Get revision for subversion project

Returns
revision for subversion project

Return type
str

get_svn_project_url()

Gets subversion project url

Returns
subversion project url

Return type
str

get_synchronize(*section*)

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

get_tasks(*section*)

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

get_threads(*section*)

Gets threads needed for the given job type :param section: job type :type section: str :return: threads needed :rtype: str

get_total_jobs()

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

Returns
max number of running jobs

Return type
int

get_version()

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

Returns
version

Return type
str

get_wchunkinc(*section*)

Gets the chunk increase to wallclock :param section: job type :type section: str :return: wallclock increase per chunk :rtype: str

get_wrapper_check_time()

Returns time to check the status of jobs in the wrapper

Returns
wrapper check time

Return type
int

get_wrapper_export(*wrapper={}*)

Returns modules variable from wrapper

Returns
string

Return type
string

get_wrapper_jobs(*wrapper=None*)

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

Returns
expression (or none)

Return type
string

get_wrapper_machinefiles(*wrapper={}*)

Returns the strategy for creating the machinefiles in wrapper jobs

Returns
machinefiles function to use

Return type
string

get_wrapper_method(*wrapper={}*)

Returns the method of make the wrapper

Returns
method

Return type
string

get_wrapper_partition(*wrapper={}*)

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

Returns
expression (or none)

Return type
string

`get_wrapper_policy`(*wrapper={}*)

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

Returns
wrapper type (or none)

Return type
string

`get_wrapper_queue`(*wrapper={}*)

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

Returns
expression (or none)

Return type
string

`get_wrapper_retrials`(*wrapper={}*)

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

Returns
safety sleep time

Return type
int

`get_wrapper_type`(*wrapper={}*)

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

Returns
wrapper type (or none)

Return type
string

`get_wrappers`()

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

Returns
expression

Return type
dict

`get_x11`(*section*)

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

`get_x11_jobs`()

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

Returns
expression (or none)

Return type
string

`get_yaml_filenames_to_load`(*yaml_folder*, *ignore_minimal=False*)

Get all yaml files in a folder and return a list with the filenames :param yaml_folder: folder to search for yaml files :param ignore_minimal: ignore minimal files :return: list of filenames

`load_common_parameters`(*parameters*)

Loads common parameters not specific to a job neither a platform :param parameters: :return:

`load_config_file`(*current_folder_data*, *yaml_file*)

Load a config file and parse it :param current_folder_data: current folder data :param yaml_file: yaml file to load :return: unified config file

`load_config_folder`(*current_data*, *yaml_folder*, *ignore_minimal=False*)

Load a config folder and return pre and post config :param current_data: current data to be updated :param yaml_folder: folder to load config :param ignore_minimal: ignore minimal config files :return: pre and post config

`load_custom_config`(*current_data*, *filenames_to_load*)

Loads custom config files :param current_data: dict with current data :param filenames_to_load: list of filenames to load :return: current_data_pre,current_data_post with unified data

`load_custom_config_section`(*current_data*, *filenames_to_load*)

Loads a section (PRE or POST), simple str are also PRE data of the custom config files :param current_data: data until now :param filenames_to_load: files to load in this section :return:

`load_parameters`()

Load all experiment data :return: a dictionary containing tuples [parameter_name, parameter_value] :rtype: dict

`load_platform_parameters`()

Load parameters from platform config files.

Returns
a dictionary containing tuples [parameter_name, parameter_value]

Return type
dict

`load_section_parameters`(*job_list*, *as_conf*, *submitter*)

Load parameters from job config files.

Returns
a dictionary containing tuples [parameter_name, parameter_value]

Return type
dict

`normalize_parameters_keys`(*parameters*, *default_parameters={}*)

Normalize the parameters keys to be exportable in the templates case-insensitive. :param parameters: dictionary containing the parameters :param default_parameters: dictionary containing the default parameters, they must remain in lower-case :return: upper-case parameters

`normalize_variables`(*data*)

Apply some memory internal variables to normalize it format. (right now only dependencies)

parse_data_loops(*experiment_data*, *data_loops*)

This function, looks for the FOR keyword, to generates N amount of subsections of the same section. Looks for the "NAME" keyword, inside this FOR keyword to determine the name of the new sections Experiment_data is the dictionary that contains all the sections, a subsection could be located at the root but also in a nested section :param experiment_data: dictionary with all the sections :param data_loops: list of lists with the path to the section that contains the FOR keyword :return: Original experiment_data with the sections in the data_loops updated changing the FOR by multiple new sections

parse_githooks()

Parse githooks section in configuration file

Returns
dictionary with githooks configuration

Return type
dict

static parse_placeholders(*content*, *parameters*)

Parse placeholders in content

Parameters

- **content** (*str*) – content to be parsed
- **parameters** (*dict*) – parameters to be used in parsing

Returns
parsed content

Return type
str

quick_deep_diff(*current_data*, *last_run_data*, *changed=False*)

Returns if there is any difference between the current configuration and the stored one :param last_run_data: dictionary with the stored data :return: changed: boolean, True if the configuration has changed

reload(*force_load=False*, *only_experiment_data=False*, *save=False*)

Reloads the configuration files :param force_load: If True, reloads all the files, if False, reloads only the modified files

save()

Saves the experiment data into the experiment_folder/conf/metadata folder as a yaml file :return: True if the data has changed, False otherwise

set_expid(*exp_id*)

Set experiment identifier in autosubmit and experiment config files

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

set_git_project_commit(*as_conf*)

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

set_new_host(*section*, *new_host*)

Sets new host for given platform :param new_host: :param section: platform name :type: str

set_new_project(*section*, *new_project*)

Sets new project for given platform :param new_project: :param section: platform name :type: str

set_new_user(*section*, *new_user*)

Sets new user for given platform :param new_user: :param section: platform name :type: str

set_platform(*hpc*)

Sets main platforms in experiment's config file

Parameters
hpc – main platforms

Type
str

set_safetysleeptime(*sleep_time*)

Sets autosubmit's version in autosubmit's config file

Parameters
sleep_time (*int*) – value to set

set_version(*autosubmit_version*)

Sets autosubmit's version in autosubmit's config file

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

substitute_dynamic_variables(*parameters=None*, *max_deep=25*, *dict_keys_type=None*, *not_in_data=''*, *in_the_end=False*)

Substitute dynamic variables in the experiment data :parameter :return:

unify_conf(*current_data*, *new_data*)

Unifies all configuration files into a single dictionary. :param current_data: dict with current configuration :param new_data: dict with new configuration :return: dict with new configuration taking priority over current configuration

8.3 autosubmit.database

Module containing functions to manage autosubmit's database.

exception autosubmit.database.db_common.**DbException**(*message*)

Exception class for database errors

autosubmit.database.db_common.**check_db**()

Checks if database file exist

Returns
None if exists, terminates program if not

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

Checks if exist an experiment with the given name. Anti-lock version.

Parameters

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

Returns
If experiment exists returns true, if not returns false

Return type
bool

autosubmit.database.db_common.**close_conn**(*conn*, *cursor*)

Commits changes and close connection to database

Parameters

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

autosubmit.database.db_common.**create_db**(*qry*)

Creates a new database for autosubmit

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

autosubmit.database.db_common.**delete_experiment**(*experiment_id*)

Removes experiment from database. Anti-lock version.

Parameters
experiment_id (*str*) – experiment identifier

Returns
True if delete is successful

Return type
bool

autosubmit.database.db_common.**get_autosubmit_version**(*expid*)

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

Parameters
expid (*str*) – Experiment name

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

Return type
str

autosubmit.database.db_common.**last_name_used**(*test=False*, *operational=False*)

Gets last experiment identifier used. Anti-lock version.

Parameters

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

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

Return type
str

autosubmit.database.db_common.**open_conn**(*check_version=True*)

Opens a connection to database

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

Returns
connection object, cursor object

Return type
sqlite3.Connection, sqlite3.Cursor

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

Stores experiment in database. Anti-lock version.

Parameters

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

autosubmit.database.db_common.**update_experiment_descrip_version**(*name*, *description=None*, *version=None*)

Updates the experiment's description and/or version. Anti-lock version.

Parameters

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

Rtype name
str

Rtype description
str

Rtype version
str

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

Return type
bool

8.4 autosubmit.git

class autosubmit.git.autosubmit_git.**AutosubmitGit**(*expid*)

Class to handle experiment git repository

Parameters
expid (*str*) – experiment identifier

static check_commit(*as_conf*)

Function to check uncommitted changes

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

static clean_git(*as_conf*)

Function to clean space on BasicConfig.LOCAL_ROOT_DIR/git directory.

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

static clone_repository(*as_conf*, *force*, *hpcarch*)

Clones a specified git repository on the project folder

Parameters

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

Returns
True if clone was successful, False otherwise

8.5 autosubmit.job

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

class autosubmit.job.job.**Job**(*name*, *job_id*, *status*, *priority*)

Class to handle all the tasks with Jobs at HPC.

A job is created by default with a name, a jobid, a status and a type. It can have children and parents. The inheritance reflects the dependency between jobs. If Job2 must wait until Job1 is completed then Job2 is a child of Job1. Inversely Job1 is a parent of Job2

add_children(*children*)

Add children for the job. It also adds current job as a parent for all the new children

Parameters
children (*list of Job objects*) – job's children to add

add_edge_info(*parent*, *special_conditions*)

Adds edge information to the job

Parameters

- **parent** (Job) – parent job
- **special_conditions** (*dict*) – special variables

add_parent(**parents*)

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

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

calendar_chunk(*parameters*)

Calendar for chunks

Parameters
parameters –

Returns

calendar_split(*as_conf*, *parameters*)

Calendar for splits :param parameters: :return:

check_completion(*default_status=-1*, *over_wallclock=False*)

Check the presence of *COMPLETED* file. Change status to COMPLETED if *COMPLETED* file exists and to FAILED otherwise. :param over_wallclock: :param default_status: status to set if job is not completed. By default, is FAILED :type default_status: Status

check_end_time(*fail_count=-1*)

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_running_after(*date_limit*)

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

check_script(*as_conf*, *parameters*, *show_logs='false'*)

Checks if script is well-formed

Parameters

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

Returns
true if not problem has been detected, false otherwise

Return type
bool

check_start_time(*fail_count=-1*)

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

`property checkpoint`

Generates a checkpoint step for this job based on job.type.

`property children`

Returns a list containing all children of the job

Returns
child jobs

Return type
set

`property children_names_str`

Comma separated list of children's names

`property chunk`

Current chunk.

`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

`property custom_directives`

List of custom directives.

`property delay`

Current delay.

`property delay_retrials`

TODO

`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

`property dependencies`

Current job dependencies.

property export

TODO.

property fail_count

Number of failed attempts to run this job.

property frequency

TODO.

get_checkpoint_files()

Check if there is a file on the remote host that contains the checkpoint

get_last_retrials() → List[Union[datetime, str]]

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

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

Return type
list of list

get_new_remotelog_name(*count=-1*)

Checks if remote log file exists on remote host if it exists, remote_log variable is updated :param

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

property hyperthreading

Detects if hyperthreading is enabled or not.

inc_fail_count()

Increments fail count

static is_a_completed_retrial(*fields*)

Returns true only if there are 4 fields: submit start finish status, and status equals COMPLETED.

is_ancestor(*job*)

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

is_over_wallclock(*start_time*, *wallclock*)

Check if the job is over the wallclock time, it is an alternative method to avoid platform issues :param start_time: :param wallclock: :return:

is_parent(*job*)

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

property long_name

Job's long name. If not set, returns name

Returns
long name

Return type
str

property member

Current member.

property memory

Memory requested for the job.

property memory_per_task

Memory requested per task.

property name

Current job full name.

property nodes

Number of nodes that the job will use.

property packed

TODO

property parents

Returns parent jobs list

Returns
parent jobs

Return type
set

property partition

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

property platform

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

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

process_scheduler_parameters(*as_conf*, *parameters*, *job_platform*, *chunk*)

Parsers yaml data stored in the dictionary and calculates the components of the heterogeneous job if any :return:

property processors

Number of processors that the job will use.

property processors_per_node

Number of processors per node that the job can use.

property 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

read_header_tailer_script(*script_path: str*, *as_conf:* AutosubmitConfig, *is_header: bool*)

Opens and reads a script. If it is not a BASH script it will fail :(

Will strip away the line with the hash bang (#!)

Parameters

- **script_path** – relative to the experiment directory path to the script
- **as_conf** – Autosubmit configuration file
- **is_header** – boolean indicating if it is header extended script

remove_redundant_parents()

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

property retrials

Max amount of retrials to run this job.

retrieve_logfiles(*platform*, *raise_error=False*)

Retrieves log files from remote host meant to be used inside a process. :param platform: platform that is calling the function, already connected. :param raise_error: boolean to raise an error if the logs are not retrieved :return:

property scratch_free_space

Percentage of free space required on the `scratch`.

property sdate

Current start date.

property section

Type of the job, as given on job configuration file.

property split

Current split.

property splits

Max number of splits.

property status_str

String representation of the current status

property synchronize

TODO.

property tasks

Number of tasks that the job will use.

property threads

Number of threads that the job will use.

property total_processors

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

update_content(*as_conf*)

Create the script content to be run for the job

Parameters
as_conf (*config*) – config

Returns
script code

Return type
str

update_job_variables_final_values(*parameters*)

Jobs variables final values based on parameters dict instead of as_conf This function is called to handle %CURRENT_% placeholders as they are filled up dynamically for each job

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

Refresh parameters value

Parameters

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

update_status(*as_conf*, *failed_file=False*)

Updates job status, checking COMPLETED file if needed

Parameters

- **as_conf** –
- **failed_file** – boolean, if True, checks if the job failed

Returns

property wallclock

Duration for which nodes used by job will remain allocated.

write_end_time(*completed*, *enable_vertical_write=False*, *count=-1*)

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

write_start_time(*enable_vertical_write=False*, *from_stat_file=False*, *count=-1*)

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

write_submit_time()

Writes submit date and time to TOTAL_STATS file. It doesn't write if hold is True.

write_total_stat_by_retries(*total_stats*, *first_retrial=False*)

Writes all data to TOTAL_STATS file :param total_stats: data gathered by the wrapper :type total_stats: dict :param first_retrial: True if this is the first retry, False otherwise :type first_retrial: bool

class autosubmit.job.job.**WrapperJob**(*name*, *job_id*, *status*, *priority*, *job_list*, *total_wallclock*, *num_processors*, *platform*, *as_config*, *hold*)

Defines a wrapper from a package.

Calls Job constructor.

Parameters

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

class autosubmit.job.job_common.**StatisticsSnippetBash**

Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetEmpty**

Class to handle the statistics snippet of a job. It contains header and footer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetPython**(*version='3'*)

Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**StatisticsSnippetR**

Class to handle the statistics snippet of a job. It contains header and tailer for local and remote jobs

class autosubmit.job.job_common.**Status**

Class to handle the status of a job

class autosubmit.job.job_common.**Type**

Class to handle the status of a job

autosubmit.job.job_common.**increase_wallclock_by_chunk**(*current*, *increase*, *chunk*)

Receives the wallclock times an increases it according to a quantity times the number of the current chunk. The result cannot be larger than 48:00. If Chunk = 0 then no increment.

Parameters

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

Returns

HH:MM wallclock

Return type

str

autosubmit.job.job_common.**parse_output_number**(*string_number*)

Parses number in format 1.0K 1.0M 1.0G

Parameters

string_number (*str*) – String representation of number

Returns

number in float format

Return type
float

class autosubmit.job.job_list.**JobList**(*expid*, *config*, *parser_factory*, *job_list_persistence*, *as_conf*)

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

add_logs(*logs*)

add logs to the current job_list :return: logs :rtype: dict(tuple)

add_special_conditions(*job*, *special_conditions*, *filters_to_apply*, *parent*)

Add special conditions to the job edge :param job: Job :param special_conditions: dict :param filters_to_apply: dict :param parent: parent job :return:

backup_save()

Persists the job list

check_checkpoint(*job*, *parent*)

Check if a checkpoint step exists for this edge

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

check_special_status()

Check if all parents of a job have the correct status for checkpointing :return: jobs that fullfill the special conditions

property expid

Returns the experiment identifier

Returns
experiment's identifier

Return type
str

find_and_delete_redundant_relations(*problematic_jobs*)

Jobs with intrisic rules than can't be safelty not added without messing other workflows. The graph will have the least amount of edges added as much as safely possible before this function. Structure: problematic_jobs structure is {section: {child_name: [parent_names]}}

Returns

generate(*as_conf*, *date_list*, *member_list*, *num_chunks*, *chunk_ini*, *parameters*, *date_format*, *default_retrials*, *default_job_type*, *wrapper_jobs={}*, *new=True*, *run_only_members=[]*, *show_log=True*, *monitor=False*, *force=False*, *create=False*)

Creates all jobs needed for the current workflow. :param as_conf: AutosubmitConfig object :type as_conf: AutosubmitConfig :param date_list: list of dates :type date_list: list :param member_list: list of members :type member_list: list :param num_chunks: number of chunks :type num_chunks: int :param chunk_ini: initial chunk :type chunk_ini: int :param parameters: parameters :type parameters: dict :param date_format: date format (D/M/Y) :type date_format: str :param default_retrials: default number of retrials :type default_retrials: int :param default_job_type: default job type :type default_job_type: str :param wrapper_jobs: wrapper jobs :type wrapper_jobs: dict :param new: new :type new: bool :param run_only_members: run only members :type run_only_members: list :param show_log: show log :type show_log: bool :param monitor: monitor :type monitor: bool

get_active(*platform=None*, *wrapper=False*)

Returns a list of active jobs (In platforms queue + Ready)

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
active jobs

Return type
list

get_all(*platform=None*, *wrapper=False*)

Returns a list of all jobs

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
all jobs

Return type
list

get_chunk_list()

Get inner chunk list

Returns
chunk list

Return type
list

get_completed(*platform=None*, *wrapper=False*)

Returns a list of completed jobs

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
completed jobs

Return type
list

get_completed_without_logs(*platform=None*)

Returns a list of completed jobs without updated logs

Parameters
platform (*HPCPlatform*) – job platform

Returns
completed jobs

Return type
list

get_date_list()

Get inner date list

Returns
date list

Return type
list

get_delayed(*platform=None*)

Returns a list of delayed jobs

Parameters
platform (*HPCPlatform*) – job platform

Returns
delayed jobs

Return type
list

get_failed(*platform=None*, *wrapper=False*)

Returns a list of failed jobs

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
failed jobs

Return type
list

get_finished(*platform=None*, *wrapper=False*)

Returns a list of jobs finished (Completed, Failed)

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
finished jobs

Return type
list

get_held_jobs(*platform=None*)

Returns a list of jobs in the platforms (Held)

Parameters
platform (*HPCPlatform*) – job platform

Returns
jobs in platforms

Return type
list

get_in_queue(*platform=None*, *wrapper=False*)

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

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
jobs in platforms

Return type
list

get_job_by_name(*name*)

Returns the job that its name matches parameter name

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

Returns
found job

Return type
job

get_job_list()

Get inner job list

Returns
job list

Return type
list

get_job_names(*lower_case=False*)

Returns a list of all job names :param: lower_case: if true, returns lower case job names :type: lower_case: bool

Returns
all job names

Return type
list

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

Parameters

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

Returns
jobs_list names

Return type
list

get_jobs_by_section(*section_list*)

Returns the job that its name matches parameter section :parameter section_list: list of sections to look for :type section_list: list :return: found job :rtype: job

get_logs()

Returns a dict of logs by jobs_name jobs

Returns
logs

Return type
dict(tuple)

get_member_list()

Get inner member list

Returns
member list

Return type
list

get_not_in_queue(*platform=None*, *wrapper=False*)

Returns a list of jobs NOT in the platforms (Ready, Waiting)

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
jobs not in platforms

Return type
list

get_ordered_jobs_by_date_member(*section*)

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

Returns
jobs ordered divided by date and member

Return type
dict

get_prepared(*platform=None*)

Returns a list of prepared jobs

Parameters
platform (*HPCPlatform*) – job platform

Returns
prepared jobs

Return type
list

get_queuing(*platform=None*, *wrapper=False*)

Returns a list of jobs queuing

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
queuedjobs

Return type
list

get_ready(*platform=None*, *hold=False*, *wrapper=False*)

Returns a list of ready jobs

Parameters

- **wrapper** –
- **hold** –
- **platform** (*HPCPlatform*) – job platform

Returns
ready jobs

Return type
list

get_running(*platform=None*, *wrapper=False*)

Returns a list of jobs running

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
running jobs

Return type
list

get_skipped(*platform=None*)

Returns a list of skipped jobs

Parameters
platform (*HPCPlatform*) – job platform

Returns
skipped jobs

Return type
list

get_submitted(*platform=None*, *hold=False*, *wrapper=False*)

Returns a list of submitted jobs

Parameters

- **wrapper** –
- **hold** –
- **platform** (*HPCPlatform*) – job platform

Returns
submitted jobs

Return type
list

`get_suspended`(*platform=None*, *wrapper=False*)

Returns a list of jobs on unknown state

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
unknown state jobs

Return type
list

`get_uncompleted`(*platform=None*, *wrapper=False*)

Returns a list of completed jobs

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
completed jobs

Return type
list

`get_uncompleted_and_not_waiting`(*platform=None*, *wrapper=False*)

Returns a list of completed jobs and waiting

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
completed jobs

Return type
list

`get_unknown`(*platform=None*, *wrapper=False*)

Returns a list of jobs on unknown state

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
unknown state jobs

Return type
list

get_unsubmitted(*platform=None*, *wrapper=False*)

Returns a list of unsubmitted jobs

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
all jobs

Return type
list

get_waiting(*platform=None*, *wrapper=False*)

Returns a list of jobs waiting

Parameters

- **wrapper** –
- **platform** (*HPCPlatform*) – job platform

Returns
waiting jobs

Return type
list

get_waiting_remote_dependencies(*platform_type='slurm'*)

Returns a list of jobs waiting on slurm scheduler :param platform_type: platform type :type platform_type: str :return: waiting jobs :rtype: list

load(*create=False*, *backup=False*)

Recreates a stored job list from the persistence

Returns
loaded job list object

Return type
JobList

static load_file(*filename*)

Recreates a stored joblist from the pickle file

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

Returns
loaded joblist object

Return type
JobList

property parameters

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

print_with_status(*statusChange=None*, *nocolor=False*, *existingList=None*)

Returns the string representation of the dependency tree of the Job List

Parameters

- **statusChange** (*List of strings*) – List of changes in the list, supplied in set status

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

Returns
String representation

Return type
String

remove_rerun_only_jobs(*notransitive=False*)

Removes all jobs to be run only in reruns

rerun(*job_list_unparsed*, *as_conf*, *monitor=False*)

Updates job list to rerun the jobs specified by a job list :param job_list_unparsed: list of jobs to rerun :type job_list_unparsed: str :param as_conf: experiment configuration :type as_conf: AutosubmitConfig :param monitor: if True, the job list will be monitored :type monitor: bool

static retrieve_packages(*BasicConfig*, *expid*, *current_jobs=None*)

Retrieves dictionaries that map the collection of packages in the experiment

Parameters

- **BasicConfig** (*Configuration Object*) – Basic configuration
- **expid** (*String*) – Experiment ID
- **current_jobs** (*list*) – list of names of current jobs

Returns
job to package, package to job, package to package_id, package to symbol

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

static retrieve_times(*status_code*, *name*, *tmp_path*, *make_exception=False*, *job_times=None*, *seconds=False*, *job_data_collection=None*)

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

save()

Persists the job list

sort_by_id()

Returns a list of jobs sorted by id

Returns
jobs sorted by ID

Return type
list

sort_by_name()

Returns a list of jobs sorted by name

Returns
jobs sorted by name

Return type
list

`sort_by_status()`

Returns a list of jobs sorted by status

Returns
job sorted by status

Return type
list

`sort_by_type()`

Returns a list of jobs sorted by type

Returns
job sorted by type

Return type
list

`update_from_file(store_change=True)`

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

`update_genealogy()`

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

`update_list(as_conf: AutosubmitConfig, store_change: bool = True, fromSetStatus: bool = False, submitter: Optional[object] = None, first_time: bool = False) → bool`

Updates job list, resetting failed jobs and changing to READY all WAITING jobs with all parents COMPLETED

Parameters

- **first_time** –
- **submitter** –
- **fromSetStatus** –
- **store_change** –
- **as_conf** (AutosubmitConfig) – autosubmit config object

Returns
True if job status were modified, False otherwise

Return type
bool

`update_log_status(job, as_conf)`

Updates the log err and log out.

8.6 autosubmit.monitor

class autosubmit.monitor.monitor.**Monitor**

Class to handle monitoring of Jobs at HPC.

static clean_plot(*expid*)

Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all plots except last two.

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

static clean_stats(*expid*)

Function to clean space on BasicConfig.LOCAL_ROOT_DIR/plot directory. Removes all stats' plots except last two.

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

static color_status(*status*)

Return color associated to given status

Parameters
status (Status) – status

Returns
color

Return type
str

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

Create graph from joblist

Parameters

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

Returns
created graph

Return type
pydotplus.Dot

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

Plots graph for joblist and stores it in a file

Parameters

- **hide_groups** –
- **groups** –
- **packages** –

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

generate_output_stats(*expid: str*, *joblist: List[Job]*, *output_format: str = 'pdf'*, *period_ini: Optional[datetime] = None*, *period_fi: Optional[datetime] = None*, *show: bool = False*, *queue_time_fixes: Optional[Dict[str, int]] = None*) → None

Plots stats for joblist and stores it in a file

Parameters

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

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

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

static get_general_stats(*expid: str*) → List[str]

Returns all the options in the sections of the %expid%_GENERAL_STATS. Options with values larger than GENERAL_STATS_OPTION_MAX_LENGTH characters are not added.

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

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

Return type
list

8.7 autosubmit.platform

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

Bases: ParamikoPlatform

Class to manage queues with ecaccess

Parameters

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

check_Alljobs(*job_list*, *as_conf*, *retries=5*)

Checks jobs running status :param job_list: list of jobs :type job_list: list :param job_list_cmd: list of jobs in the queue system :type job_list_cmd: str :param remote_logs: remote logs :type remote_logs: str :param retries: retries :type default_status: bool :return: current job status :rtype: autosubmit.job.job_common.Status

connect(*as_conf*, *reconnect=False*)

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

Returns
True

Return type
bool

delete_file(*filename*)

Deletes a file from this platform

Parameters
filename (*str*) – file name

Returns
True if successful or file does not exist

Return type
bool

get_checkAlljobs_cmd(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_checkjob_cmd(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_file(*filename*, *must_exist=True*, *relative_path=''*, *ignore_log=False*, *wrapper_failed=False*)

Copies a file from the current platform to experiment's tmp folder

Parameters

- **wrapper_failed** –

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

Returns
True if file is copied successfully, false otherwise

Return type
bool

get_mkdir_cmd()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

get_ssh_output()

Gets output from last command executed

Returns
output from last command

Return type
str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

Returns
command to submit job to platforms

Return type
str

get_submitted_job_id(*output*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

jobs_in_queue()

Returns empty list because ecacces does not support this command

Returns
empty list

Return type
list

move_file(*src*, *dest*, *must_exist=False*)

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

parse_Alljobs_output(*output*, *job_id*)

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

parse_job_output(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

restore_connection(*as_conf*)

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

Returns
True

Return type
bool

send_command(*command*, *ignore_log=False*, *x11=False*)

Sends given command to HPC

Parameters

- **x11** –
- **ignore_log** –
- **command** (*str*) – command to send

Returns
True if executed, False if failed

Return type
bool

send_file(*filename*, *check=True*)

Sends a local file to the platform :param check: :param filename: name of the file to send :type filename: str

submit_Script(*hold=False*)

Sends a Submitfile Script, exec in platform and retrieve the Jobs_ID. :param hold: send job hold :type hold: boolean :return: job id for the submitted job :rtype: int

test_connection(*as_conf*)

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

Returns
True

Return type
bool

update_cmds()

Updates commands for platforms

class autosubmit.platforms.lsfplatform.LsfPlatform(*expid*, *name*, *config*)

Bases: ParamikoPlatform

Class to manage jobs to host using LSF scheduler

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

check_Alljobs(*job_list*, *as_conf*, *retries=5*)

Checks jobs running status :param job_list: list of jobs :type job_list: list :param job_list_cmd: list of jobs in the queue system :type job_list_cmd: str :param remote_logs: remote logs :type remote_logs: str :param retries: retries :type default_status: bool :return: current job status :rtype: autosubmit.job.job_common.Status

get_checkAlljobs_cmd(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_checkjob_cmd(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_mkdir_cmd()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

`get_submit_cmd`(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

Returns
command to submit job to platforms

Return type
str

`get_submitted_job_id`(*output*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

`parse_Alljobs_output`(*output*, *job_id*)

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

`parse_job_output`(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

`submit_Script`(*hold=False*)

Sends a Submitfile Script, exec in platform and retrieve the Jobs_ID. :param hold: send job hold :type hold: boolean :return: job id for the submitted job :rtype: int

`update_cmds`()

Updates commands for platforms

class autosubmit.platforms.pbsplatform.**PBSPlatform**(*expid*, *name*, *config*, *version*)

Bases: ParamikoPlatform

Class to manage jobs to host using PBS scheduler

Parameters

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

- **version** (*str*) – scheduler version

check_Alljobs(*job_list*, *as_conf*, *retries=5*)

Checks jobs running status :param job_list: list of jobs :type job_list: list :param job_list_cmd: list of jobs in the queue system :type job_list_cmd: str :param remote_logs: remote logs :type remote_logs: str :param retries: retries :type default_status: bool :return: current job status :rtype: autosubmit.job.job_common.Status

get_checkAlljobs_cmd(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_checkjob_cmd(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_mkdir_cmd()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

Returns
command to submit job to platforms

Return type
str

get_submitted_job_id(*output*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

parse_Alljobs_output(*output*, *job_id*)

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

parse_job_output(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

submit_Script(*hold=False*)

Sends a Submitfile Script, exec in platform and retrieve the Jobs_ID. :param hold: send job hold :type hold: boolean :return: job id for the submitted job :rtype: int

update_cmds()

Updates commands for platforms

class autosubmit.platforms.sgeplatform.**SgePlatform**(*expid*, *name*, *config*)

Bases: ParamikoPlatform

Class to manage jobs to host using SGE scheduler

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

check_Alljobs(*job_list*, *as_conf*, *retries=5*)

Checks jobs running status :param job_list: list of jobs :type job_list: list :param job_list_cmd: list of jobs in the queue system :type job_list_cmd: str :param remote_logs: remote logs :type remote_logs: str :param retries: retries :type default_status: bool :return: current job status :rtype: autosubmit.job.job_common.Status

connect(*as_conf*, *reconnect=False*)

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

Returns
True

Return type
bool

get_checkAlljobs_cmd(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_checkjob_cmd(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

get_mkdir_cmd()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

get_submit_cmd(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

Returns
command to submit job to platforms

Return type
str

get_submitted_job_id(*output*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

parse_Alljobs_output(*output*, *job_id*)

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

parse_job_output(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

restore_connection(*as_conf*)

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

Returns
True

Return type
bool

submit_Script(*hold=False*)

Sends a Submitfile Script, exec in platform and retrieve the Jobs_ID. :param hold: send job hold :type hold: boolean :return: job id for the submitted job :rtype: int

test_connection(*as_conf*)

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

Returns
True

Return type
bool

update_cmds()

Updates commands for platforms

class autosubmit.platforms.slurmplatform.**SlurmPlatform**(*expid*, *name*, *config*, *auth_password=None*)

Bases: ParamikoPlatform

Class to manage jobs to host using SLURM scheduler

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

check_remote_log_dir()

Creates log dir on remote host

get_checkAlljobs_cmd(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

`get_checkjob_cmd`(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

`get_estimated_queue_time_cmd`(*job_id*)

Returns command to get estimated queue time on remote platforms

Parameters

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

Returns
command to get estimated queue time

`get_jobid_by_jobname_cmd`(*job_name*)

Returns command to get job id by job name on remote platforms :param job_name: :return: str

`get_mkdir_cmd`()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

`get_queue_status`(*in_queue_jobs*, *list_queue_jobid*, *as_conf*)

Get queue status for a list of jobs.

The job statuses are normally found via a command sent to the remote platform.

Each `job` in `in_queue_jobs` must be updated. Implementations may check for the reason for queueing cancellation, or if the job is held, and update the `job` status appropriately.

`get_queue_status_cmd`(*job_id*)

Returns command to get queue status on remote platforms :return: str

`get_submit_cmd`(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

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

Returns
command to submit job to platforms

Return type
str

get_submitted_job_id(*outputlines*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

open_submit_script()

Opens Submit script file

parse_Alljobs_output(*output*, *job_id*)

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

parse_job_finish_data(*output*, *packed*)

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

When a wrapper has finished, capture finish time.

Parameters

- **output** (*str*) – The sacct output
- **packed** (*bool*) – true if job belongs to package

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

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

parse_job_output(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

parse_queue_reason(*output*, *job_id*)

Parses the queue reason from the output of the command :param output: output of the command :param job_id: job id :return: queue reason :rtype: str

process_batch_ready_jobs(*valid_packages_to_submit*, *failed_packages*, *error_message=''*, *hold=False*)

Retrieve multiple jobs identifiers. :param valid_packages_to_submit: :param failed_packages: :param error_message: :param hold: :return:

submit_Script(*hold: bool = False*) → Union[List[str], str]

Sends a Submit file Script, execute it in the platform and retrieves the Jobs_ID of all jobs at once.

Parameters
hold (*bool*) – if True, the job will be held

Returns
job id for submitted jobs

Return type
list(str)

submit_job(*job*, *script_name*, *hold=False*, *export='none'*)

Submit a job from a given job object.

Parameters

- **export** –
- **job** (autosubmit.job.job.Job) – job object
- **script_name** – job script's name
- **hold** (*boolean*) – send job hold

Rtype scriptname
str

Returns
job id for the submitted job

Return type
int

update_cmds()

Updates commands for platforms

class autosubmit.platforms.locplatform.**LocalPlatform**(*expid*, *name*, *config*, *auth_password=None*)

Bases: ParamikoPlatform

Class to manage jobs to localhost

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

check_Alljobs(*job_list*, *as_conf*, *retries=5*)

Checks jobs running status :param job_list: list of jobs :type job_list: list :param job_list_cmd: list of jobs in the queue system :type job_list_cmd: str :param remote_logs: remote logs :type remote_logs: str :param retries: retries :type default_status: bool :return: current job status :rtype: autosubmit.job.job_common.Status

check_file_exists(*src*, *wrapper_failed=False*, *sleeptime=5*, *max_retries=3*, *first=True*)

Moves a file on the platform :param src: source name :type src: str :param: wrapper_failed: if True, the wrapper failed. :type wrapper_failed: bool

connect(*as_conf*, *reconnect=False*)

Creates ssh connection to host

Returns
True if connection is created, False otherwise

Return type
bool

`delete_file`(*filename*, *del_cmd=False*)

Deletes a file from this platform

Parameters
filename (*str*) – file name

Returns
True if successful or file does not exist

Return type
bool

`get_checkAlljobs_cmd`(*jobs_id*)

Returns command to check jobs status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

`get_checkjob_cmd`(*job_id*)

Returns command to check job status on remote platforms

Parameters

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

Returns
command to check job status

Return type
str

`get_file`(*filename*, *must_exist=True*, *relative_path=''*, *ignore_log=False*, *wrapper_failed=False*)

Copies a file from the current platform to experiment's tmp folder

Parameters

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

Returns
True if file is copied successfully, false otherwise

Return type
bool

`get_logs_files`(*exp_id*, *remote_logs*)

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

Parameters

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

`get_mkdir_cmd`()

Gets command to create directories on HPC

Returns
command to create directories on HPC

Return type
str

`get_ssh_output`()

Gets output from last command executed

Returns
output from last command

Return type
str

`get_submit_cmd`(*job_script*, *job*, *hold=False*, *export=''*)

Get command to add job to scheduler

Parameters

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

Returns
command to submit job to platforms

Return type
str

`get_submitted_job_id`(*output*, *x11=False*)

Parses submit command output to extract job id :param x11: :param output: output to parse :type output: str :return: job id :rtype: str

`move_file`(*src*, *dest*, *must_exist=False*)

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

parse_Alljobs_output(*output*, *job_id*)

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

parse_job_output(*output*)

Parses check job command output, so it can be interpreted by autosubmit

Parameters
output (*str*) – output to parse

Returns
job status

Return type
str

send_command(*command*, *ignore_log=False*, *x11=False*)

Sends given command to HPC

Parameters

- **x11** –
- **ignore_log** –
- **command** (*str*) – command to send

Returns
True if executed, False if failed

Return type
bool

send_file(*filename*, *check=True*)

Sends a local file to the platform :param check: :param filename: name of the file to send :type filename: str

submit_Script(*hold=False*)

Sends a Submitfile Script, exec in platform and retrieve the Jobs_ID. :param hold: send job hold :type hold: boolean :return: job id for the submitted job :rtype: int

test_connection(*as_conf*)

Test if the connection is still alive, reconnect if not.

update_cmds()

Updates commands for platforms

Autosubmit is a lightweight workflow manager designed to meet climate research necessities. Unlike other workflow solutions in the domain, it integrates the capabilities of an experiment manager, workflow orchestrator and monitor in a self-contained application.

It is a Python package available at PyPI. The source code in Git contains a Dockerfile used in cloud environments with Kubernetes, and there are examples of how to install Autosubmit with Conda.

CHAPTER

NINE

CONTACT US

GitLab	https://earth.bsc.es/gitlab/es/autosubmit/
Email	support-autosubmit@bsc.es

PYTHON MODULE INDEX

a

autosubmit.autosubmit, 135
autosubmit.database.db_common, 160
autosubmit.git.autosubmit_git, 162
autosubmit.job.job, 163
autosubmit.job.job_common, 170
autosubmit.job.job_list, 171
autosubmit.monitor.monitor, 181
autosubmit.platforms.ecplatform, 182
autosubmit.platforms.locplatform, 194
autosubmit.platforms.lsfplatform, 186
autosubmit.platforms.pbsplatform, 187
autosubmit.platforms.sgeplatform, 189
autosubmit.platforms.slurmplatform, 191
autosubmitconfigparser.config.basicconfig, 145
autosubmitconfigparser.config.configcommon, 145

INDEX

A

add_argument() (*autosubmit.autosubmit.MyParser method*), 144
add_children() (*autosubmit.job.job.Job method*), 163
add_edge_info() (*autosubmit.job.job.Job method*), 163
add_logs() (*autosubmit.job.job_list.JobList method*), 171
add_parent() (*autosubmit.job.job.Job method*), 163
add_special_conditions() (*autosubmit.job.job_list.JobList method*), 171
archive() (*autosubmit.autosubmit.Autosubmit static method*), 135
as_conf_default_values() (*autosubmit.autosubmit.Autosubmit static method*), 135
Autosubmit (*class in autosubmit.autosubmit*), 135
autosubmit.autosubmit
    module, 135
autosubmit.database.db_common
    module, 160
autosubmit.git.autosubmit_git
    module, 162
autosubmit.job.job
    module, 163
autosubmit.job.job_common
    module, 170
autosubmit.job.job_list
    module, 171
autosubmit.monitor.monitor
    module, 181
autosubmit.platforms.ecplatform
    module, 182
autosubmit.platforms.locplatform
    module, 194
autosubmit.platforms.lsfplatform
    module, 186
autosubmit.platforms.pbsplatform
    module, 187
autosubmit.platforms.sgeplatform
    module, 189
autosubmit.platforms.slurmplatform
    module, 191
AutosubmitConfig (*class in autosubmitconfigparser.config.configcommon*), 145
autosubmitconfigparser.config.basicconfig
    module, 145
autosubmitconfigparser.config.configcommon
    module, 145
AutosubmitGit (*class in autosubmit.git.autosubmit_git*), 162

B

backup_save() (*autosubmit.job.job_list.JobList method*), 171
BasicConfig (*class in autosubmitconfigparser.config.basicconfig*), 145

C

calendar_chunk() (*autosubmit.job.job.Job method*), 163
calendar_split() (*autosubmit.job.job.Job method*), 163
cat_log() (*autosubmit.autosubmit.Autosubmit static method*), 135
change_status() (*autosubmit.autosubmit.Autosubmit static method*), 136
check() (*autosubmit.autosubmit.Autosubmit static method*), 136
check_Alljobs() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
check_Alljobs() (*autosubmit.platforms.locplatform.LocalPlatform method*), 194
check_Alljobs() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 186
check_Alljobs() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 188
check_Alljobs() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 189

check_autosubmit_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 145
check_checkpoint() (*autosubmit.job.job_list.JobList method*), 171
check_commit() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 162
check_completion() (*autosubmit.job.job.Job method*), 164
check_conf_files() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 145
check_db() (*in module autosubmit.database.db_common*), 160
check_dict_keys_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 145
check_end_time() (*autosubmit.job.job.Job method*), 164
check_expdef_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 145
check_experiment_exists() (*in module autosubmit.database.db_common*), 160
check_file_exists() (*autosubmit.platforms.locplatform.LocalPlatform method*), 194
check_jobs_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 145
check_platforms_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
check_remote_log_dir() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 191
check_retrials_end_time() (*autosubmit.job.job.Job method*), 164
check_retrials_start_time() (*autosubmit.job.job.Job method*), 164
check_running_after() (*autosubmit.job.job.Job method*), 164
check_script() (*autosubmit.job.job.Job method*), 164
check_scripts() (*autosubmit.job.job_list.JobList method*), 171
check_special_status() (*autosubmit.job.job_list.JobList method*), 171
check_start_time() (*autosubmit.job.job.Job method*), 164
check_started_after() (*autosubmit.job.job.Job method*), 165
check_wrapper_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
check_wrapper_stored_status() (*autosubmit.autosubmit.Autosubmit static method*), 136
check_wrappers() (*autosubmit.autosubmit.Autosubmit static method*), 136
checkpoint (*autosubmit.job.job.Job property*), 165
children (*autosubmit.job.job.Job property*), 165
children_names_str (*autosubmit.job.job.Job property*), 165
chunk (*autosubmit.job.job.Job property*), 165
clean() (*autosubmit.autosubmit.Autosubmit static method*), 136
clean_dynamic_variables() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
clean_git() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 162
clean_plot() (*autosubmit.monitor.monitor.Monitor static method*), 181
clean_stats() (*autosubmit.monitor.monitor.Monitor static method*), 181
clone_repository() (*autosubmit.git.autosubmit_git.AutosubmitGit static method*), 163
close_conn() (*in module autosubmit.database.db_common*), 161
color_status() (*autosubmit.monitor.monitor.Monitor static method*), 181
configure() (*autosubmit.autosubmit.Autosubmit static method*), 136
configure_dialog() (*autosubmit.autosubmit.Autosubmit static method*), 137
connect() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
connect() (*autosubmit.platforms.locplatform.LocalPlatform method*), 194
connect() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 189
convert_list_to_string() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
create() (*autosubmit.autosubmit.Autosubmit static method*), 137
create_db() (*in module autosubmit.database.db_common*), 161
create_script() (*autosubmit.job.job.Job method*), 165
create_tree_list() (*autosubmit.monitor.monitor.Monitor method*), 181
custom_directives (*autosubmit.job.job.Job property*), 165

D

database_fix() (*autosubmit.autosubmit.Autosubmit static method*), 137
DbException, 160
deep_add_missing_starter_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
deep_normalize() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
deep_parameters_export() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
deep_read_loops() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
deep_update() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
delay (*autosubmit.job.job.Job property*), 165
delay_retrials (*autosubmit.job.job.Job property*), 165
delete() (*autosubmit.autosubmit.Autosubmit static method*), 137
delete_child() (*autosubmit.job.job.Job method*), 165
delete_experiment() (*in module autosubmit.database.db_common*), 161
delete_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
delete_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 195
delete_parent() (*autosubmit.job.job.Job method*), 165
dependencies (*autosubmit.job.job.Job property*), 165
describe() (*autosubmit.autosubmit.Autosubmit static method*), 138
detailed_deep_diff() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146

E

EcPlatform (*class in autosubmit.platforms.ecplatform*), 182
environ_init() (*autosubmit.autosubmit.Autosubmit static method*), 138
error() (*autosubmit.autosubmit.MyParser method*), 144
experiment_data (*autosubmit.autosubmit.Autosubmit property*), 138
expid (*autosubmit.job.job_list.JobList property*), 171
expid() (*autosubmit.autosubmit.Autosubmit static method*), 138
export (*autosubmit.job.job.Job property*), 165

F

fail_count (*autosubmit.job.job.Job property*), 166
file_modified() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
find_and_delete_redundant_relations() (*autosubmit.job.job_list.JobList method*), 171
frequency (*autosubmit.job.job.Job property*), 166

G

generate() (*autosubmit.job.job_list.JobList method*), 171
generate_as_config() (*autosubmit.autosubmit.Autosubmit static method*), 138
generate_output() (*autosubmit.monitor.monitor.Monitor method*), 181
generate_output_stats() (*autosubmit.monitor.monitor.Monitor method*), 182
generate_output_txt() (*autosubmit.monitor.monitor.Monitor method*), 182
generate_scripts_andor_wrappers() (*autosubmit.autosubmit.Autosubmit static method*), 138
get_active() (*autosubmit.job.job_list.JobList method*), 171
get_all() (*autosubmit.job.job_list.JobList method*), 172
get_autosubmit_version() (*in module autosubmit.database.db_common*), 161
get_checkAlljobs_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
get_checkAlljobs_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 195
get_checkAlljobs_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 186
get_checkAlljobs_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 188
get_checkAlljobs_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 189
get_checkAlljobs_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 191
get_checkjob_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
get_checkjob_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 195

get_checkjob_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 186
get_checkjob_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 188
get_checkjob_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 190
get_checkjob_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_checkpoint_files() (*autosubmit.job.job.Job method*), 166
get_chunk_ini() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
get_chunk_list() (*autosubmit.job.job_list.JobList method*), 172
get_chunk_size() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 146
get_chunk_size_unit() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_communications_library() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_completed() (*autosubmit.job.job_list.JobList method*), 172
get_completed_without_logs() (*autosubmit.job.job_list.JobList method*), 172
get_copy_remote_logs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_current_host() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_current_project() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_current_user() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_custom_directives() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 147
get_date_list() (*autosubmit.job.job_list.JobList method*), 172
get_date_list() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_default_job_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_delay_retry_time() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_delayed() (*autosubmit.job.job_list.JobList method*), 173
get_dependencies() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_disable_recovery_threads() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_estimated_queue_time_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_export() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_extensible_wallclock() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_failed() (*autosubmit.job.job_list.JobList method*), 173
get_fetch_single_branch() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 148
get_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 183
get_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 195
get_file_jobs_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_file_project_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_finished() (*autosubmit.job.job_list.JobList method*), 173
get_full_config_as_json() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_general_stats() (*autosubmit.monitor.monitor.Monitor static method*), 182
get_git_project_branch() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_git_project_commit() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_git_project_origin() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149

get_git_remote_project_root() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_held_jobs() (*autosubmit.job.job_list.JobList method*), 173
get_historical_database() (*autosubmit.autosubmit.Autosubmit static method*), 139
get_in_queue() (*autosubmit.job.job_list.JobList method*), 173
get_iteration_info() (*autosubmit.autosubmit.Autosubmit static method*), 139
get_job_by_name() (*autosubmit.job.job_list.JobList method*), 174
get_job_list() (*autosubmit.job.job_list.JobList method*), 174
get_job_names() (*autosubmit.job.job_list.JobList method*), 174
get_job_related() (*autosubmit.job.job_list.JobList method*), 174
get_jobid_by_jobname_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_jobs_by_section() (*autosubmit.job.job_list.JobList method*), 174
get_last_retrials() (*autosubmit.job.job.Job method*), 166
get_local_project_path() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 149
get_logs() (*autosubmit.job.job_list.JobList method*), 175
get_logs_files() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
get_mails_to() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_processors() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_waiting_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_wallclock() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_wrapped_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_wrapped_jobs_horizontal() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_max_wrapped_jobs_vertical() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 150
get_member_list() (*autosubmit.job.job_list.JobList method*), 175
get_member_list() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_memory() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_memory_per_task() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_migrate_duplicate() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_migrate_host_to() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_migrate_project_to() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_migrate_user_to() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_min_wrapped_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 151
get_min_wrapped_jobs_horizontal() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_min_wrapped_jobs_vertical() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_mkdir_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 184
get_mkdir_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
get_mkdir_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 186
get_mkdir_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 188
get_mkdir_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 190

get_mkdir_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_new_remotelog_name() (*autosubmit.job.job.Job method*), 166
get_not_in_queue() (*autosubmit.job.job_list.JobList method*), 175
get_notifications() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_notifications_crash() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_num_chunks() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_ordered_jobs_by_date_member() (*autosubmit.job.job_list.JobList method*), 175
get_output_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 152
get_parse_two_step_start() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_parser() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig static method*), 153
get_platform() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_prepared() (*autosubmit.job.job_list.JobList method*), 175
get_processors() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_project_destination() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_project_dir() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_project_submodules_depth() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 153
get_project_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_queue_status() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_queue_status_cmd() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 192
get_queuing() (*autosubmit.job.job_list.JobList method*), 175
get_ready() (*autosubmit.job.job_list.JobList method*), 176
get_remote_dependencies() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_rerun() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_rerun_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_retrials() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_running() (*autosubmit.job.job_list.JobList method*), 176
get_safetysleeptime() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_scratch_free_space() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 154
get_section() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_skipped() (*autosubmit.job.job_list.JobList method*), 176
get_ssh_output() (*autosubmit.platforms.ecplatform.EcPlatform method*), 184
get_ssh_output() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
get_storage_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_submit_cmd() (*autosubmit.platforms.ecplatform.EcPlatform method*), 184
get_submit_cmd() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
get_submit_cmd() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
get_submit_cmd() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 188
get_submit_cmd() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 190
get_submit_cmd() (*autosub-*

mit.platforms.slurmplatform.SlurmPlatform method), 192
get_submitted() (*autosubmit.job.job_list.JobList method*), 176
get_submitted_job_id() (*autosubmit.platforms.ecplatform.EcPlatform method*), 184
get_submitted_job_id() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
get_submitted_job_id() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
get_submitted_job_id() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 189
get_submitted_job_id() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 190
get_submitted_job_id() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
get_submodules_list() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_suspended() (*autosubmit.job.job_list.JobList method*), 177
get_svn_project_revision() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_svn_project_url() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_synchronize() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_tasks() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_threads() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_total_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_uncompleted() (*autosubmit.job.job_list.JobList method*), 177
get_uncompleted_and_not_waiting() (*autosubmit.job.job_list.JobList method*), 177
get_unknown() (*autosubmit.job.job_list.JobList method*), 177
get_unsubmitted() (*autosubmit.job.job_list.JobList method*), 177
get_version() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 155
get_waiting() (*autosubmit.job.job_list.JobList method*), 178
get_waiting_remote_dependencies() (*autosubmit.job.job_list.JobList method*), 178
get_wchunkinc() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_check_time() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_export() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_machinefiles() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_method() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_partition() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 156
get_wrapper_policy() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_wrapper_queue() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_wrapper_retrials() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_wrapper_type() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_wrappers() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_x11() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_x11_jobs() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 157
get_yaml_filenames_to_load() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158

H

has_children() (*autosubmit.job.job.Job method*), 166

has_parents() (*autosubmit.job.job.Job method*), 166
hyperthreading (*autosubmit.job.job.Job property*), 166

I

inc_fail_count() (*autosubmit.job.job.Job method*), 166
increase_wallclock_by_chunk() (*in module autosubmit.job.job_common*), 170
inspect() (*autosubmit.autosubmit.Autosubmit static method*), 139
install() (*autosubmit.autosubmit.Autosubmit static method*), 139
is_a_completed_retrial() (*autosubmit.job.job.Job static method*), 166
is_ancestor() (*autosubmit.job.job.Job method*), 166
is_over_wallclock() (*autosubmit.job.job.Job method*), 166
is_parent() (*autosubmit.job.job.Job method*), 166

J

Job (*class in autosubmit.job.job*), 163
JobList (*class in autosubmit.job.job_list*), 171
jobs_in_queue() (*autosubmit.platforms.ecplatform.EcPlatform method*), 184

L

last_name_used() (*in module autosubmit.database.db_common*), 161
load() (*autosubmit.job.job_list.JobList method*), 178
load_common_parameters() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_config_file() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_config_folder() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_custom_config() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_custom_config_section() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_file() (*autosubmit.job.job_list.JobList static method*), 178
load_parameters() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_platform_parameters() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
load_section_parameters() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
LocalPlatform (*class in autosubmit.platforms.locplatform*), 194
long_name (*autosubmit.job.job.Job property*), 167
LsfPlatform (*class in autosubmit.platforms.lsfplatform*), 186

M

member (*autosubmit.job.job.Job property*), 167
memory (*autosubmit.job.job.Job property*), 167
memory_per_task (*autosubmit.job.job.Job property*), 167
migrate() (*autosubmit.autosubmit.Autosubmit static method*), 139
module
    autosubmit.autosubmit, 135
    autosubmit.database.db_common, 160
    autosubmit.git.autosubmit_git, 162
    autosubmit.job.job, 163
    autosubmit.job.job_common, 170
    autosubmit.job.job_list, 171
    autosubmit.monitor.monitor, 181
    autosubmit.platforms.ecplatform, 182
    autosubmit.platforms.locplatform, 194
    autosubmit.platforms.lsfplatform, 186
    autosubmit.platforms.pbsplatform, 187
    autosubmit.platforms.sgeplatform, 189
    autosubmit.platforms.slurmplatform, 191
    autosubmitconfigparser.config.basicconfig, 145
    autosubmitconfigparser.config.configcommon, 145
Monitor (*class in autosubmit.monitor.monitor*), 181
monitor() (*autosubmit.autosubmit.Autosubmit static method*), 140
move_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
move_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
MyParser (*class in autosubmit.autosubmit*), 144

N

name (*autosubmit.job.job.Job property*), 167
nodes (*autosubmit.job.job.Job property*), 167
normalize_parameters_keys() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
normalize_variables() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158

O

open_conn() (*in module autosubmit.database.db_common*), 161
open_submit_script() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193

P

packed (*autosubmit.job.job.Job property*), 167
parameters (*autosubmit.job.job_list.JobList property*), 178
parents (*autosubmit.job.job.Job property*), 167
parse_Alljobs_output() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
parse_Alljobs_output() (*autosubmit.platforms.locplatform.LocalPlatform method*), 196
parse_Alljobs_output() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
parse_Alljobs_output() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 189
parse_Alljobs_output() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 190
parse_Alljobs_output() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
parse_args() (*autosubmit.autosubmit.Autosubmit static method*), 140
parse_data_loops() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 158
parse_githooks() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
parse_job_finish_data() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
parse_job_output() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
parse_job_output() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
parse_job_output() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
parse_job_output() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 189
parse_job_output() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 191
parse_job_output() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
parse_output_number() (*in module autosubmit.job.job_common*), 170
parse_placeholders() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig static method*), 159
parse_queue_reason() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
partition (*autosubmit.job.job.Job property*), 167
PBSPlatform (*class in autosubmit.platforms.pbsplatform*), 187
pkl_fix() (*autosubmit.autosubmit.Autosubmit static method*), 140
platform (*autosubmit.job.job.Job property*), 167
prepare_run() (*autosubmit.autosubmit.Autosubmit static method*), 141
print_with_status() (*autosubmit.job.job_list.JobList method*), 178
process_batch_ready_jobs() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 193
process_historical_data_iteration() (*autosubmit.autosubmit.Autosubmit static method*), 141
process_scheduler_parameters() (*autosubmit.job.job.Job method*), 167
processors (*autosubmit.job.job.Job property*), 167
processors_per_node (*autosubmit.job.job.Job property*), 167

Q

queue (*autosubmit.job.job.Job property*), 167
quick_deep_diff() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159

R

read() (*autosubmitconfigparser.config.basicconfig.BasicConfig static method*), 145
read_header_tailer_script() (*autosubmit.job.job.Job method*), 168
recovery() (*autosubmit.autosubmit.Autosubmit static method*), 141
refresh() (*autosubmit.autosubmit.Autosubmit static method*), 141
reload() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159

remove_redundant_parents() (*autosubmit.job.job.Job method*), 168
remove_rerun_only_jobs() (*autosubmit.job.job_list.JobList method*), 179
report() (*autosubmit.autosubmit.Autosubmit static method*), 141
rerun() (*autosubmit.job.job_list.JobList method*), 179
rerun_recovery() (*autosubmit.autosubmit.Autosubmit static method*), 141
restore_connection() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
restore_connection() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 191
retrials (*autosubmit.job.job.Job property*), 168
retrieve_logfiles() (*autosubmit.job.job.Job method*), 168
retrieve_packages() (*autosubmit.job.job_list.JobList static method*), 179
retrieve_times() (*autosubmit.job.job_list.JobList static method*), 179
rocrate() (*autosubmit.autosubmit.Autosubmit static method*), 142
run_experiment() (*autosubmit.autosubmit.Autosubmit static method*), 142

S

save() (*autosubmit.job.job_list.JobList method*), 179
save() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
save_experiment() (*in module autosubmit.database.db_common*), 162
scratch_free_space (*autosubmit.job.job.Job property*), 168
sdate (*autosubmit.job.job.Job property*), 168
section (*autosubmit.job.job.Job property*), 168
send_command() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
send_command() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
send_file() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
send_file() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
set_expid() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
set_git_project_commit() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
set_new_host() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
set_new_project() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
set_new_user() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 159
set_platform() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 160
set_safetysleeptime() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 160
set_status() (*autosubmit.autosubmit.Autosubmit static method*), 142
set_version() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 160
SgePlatform (*class in autosubmit.platforms.sgeplatform*), 189
signal_handler() (*in module autosubmit.autosubmit*), 144
signal_handler_create() (*in module autosubmit.autosubmit*), 144
SlurmPlatform (*class in autosubmit.platforms.slurmplatform*), 191
sort_by_id() (*autosubmit.job.job_list.JobList method*), 179
sort_by_name() (*autosubmit.job.job_list.JobList method*), 179
sort_by_status() (*autosubmit.job.job_list.JobList method*), 180
sort_by_type() (*autosubmit.job.job_list.JobList method*), 180
split (*autosubmit.job.job.Job property*), 168
splits (*autosubmit.job.job.Job property*), 168
statistics() (*autosubmit.autosubmit.Autosubmit static method*), 142
StatisticsSnippetBash (*class in autosubmit.job.job_common*), 170
StatisticsSnippetEmpty (*class in autosubmit.job.job_common*), 170
StatisticsSnippetPython (*class in autosubmit.job.job_common*), 170
StatisticsSnippetR (*class in autosubmit.job.job_common*), 170
Status (*class in autosubmit.job.job_common*), 170
status_str (*autosubmit.job.job.Job property*), 168
submit_job() (*autosub-*

mit.platforms.slurmplatform.SlurmPlatform method), 194
submit_ready_jobs() (*autosubmit.autosubmit.Autosubmit static method*), 143
submit_Script() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
submit_Script() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
submit_Script() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
submit_Script() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 189
submit_Script() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 191
submit_Script() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 194
substitute_dynamic_variables() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 160
synchronize (*autosubmit.job.job.Job property*), 168

T

tasks (*autosubmit.job.job.Job property*), 168
test() (*autosubmit.autosubmit.Autosubmit static method*), 143
test_connection() (*autosubmit.platforms.ecplatform.EcPlatform method*), 185
test_connection() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
test_connection() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 191
testcase() (*autosubmit.autosubmit.Autosubmit static method*), 143
threads (*autosubmit.job.job.Job property*), 168
total_processors (*autosubmit.job.job.Job property*), 168
Type (*class in autosubmit.job.job_common*), 170

U

unarchive() (*autosubmit.autosubmit.Autosubmit static method*), 144
unify_conf() (*autosubmitconfigparser.config.configcommon.AutosubmitConfig method*), 160
update_cmds() (*autosubmit.platforms.ecplatform.EcPlatform method*), 186
update_cmds() (*autosubmit.platforms.locplatform.LocalPlatform method*), 197
update_cmds() (*autosubmit.platforms.lsfplatform.LsfPlatform method*), 187
update_cmds() (*autosubmit.platforms.pbsplatform.PBSPlatform method*), 189
update_cmds() (*autosubmit.platforms.sgeplatform.SgePlatform method*), 191
update_cmds() (*autosubmit.platforms.slurmplatform.SlurmPlatform method*), 194
update_content() (*autosubmit.job.job.Job method*), 168
update_experiment_descrip_version() (*in module autosubmit.database.db_common*), 162
update_from_file() (*autosubmit.job.job_list.JobList method*), 180
update_genealogy() (*autosubmit.job.job_list.JobList method*), 180
update_job_variables_final_values() (*autosubmit.job.job.Job method*), 169
update_list() (*autosubmit.job.job_list.JobList method*), 180
update_log_status() (*autosubmit.job.job_list.JobList method*), 180
update_parameters() (*autosubmit.job.job.Job method*), 169
update_status() (*autosubmit.job.job.Job method*), 169
update_version() (*autosubmit.autosubmit.Autosubmit static method*), 144

W

wallclock (*autosubmit.job.job.Job property*), 169
WrapperJob (*class in autosubmit.job.job*), 169
write_end_time() (*autosubmit.job.job.Job method*), 169
write_start_time() (*autosubmit.job.job.Job method*), 169
write_submit_time() (*autosubmit.job.job.Job method*), 169
write_total_stat_by_retries() (*autosubmit.job.job.Job method*), 169