

---

# **PCVS Documentation**

***Release 0.7.0***

**Julien Adam**

**Jun 09, 2023**



# BASICS

<b>1 Feature Overview</b>	<b>3</b>
1.1 Run the validation from anywhere . . . . .	3
1.2 Scale a test-suite depending on the target machine . . . . .	3
1.3 Automatic Test-suite builder . . . . .	3
1.4 Definition - Execution - Reporting in one place . . . . .	4
<b>2 Getting Started</b>	<b>5</b>
2.1 Installation . . . . .	5
2.2 PCVS-formatted test-suite . . . . .	5
2.3 Execute the test-suite . . . . .	6
2.4 Access the results . . . . .	7
<b>3 Installation Guide</b>	<b>9</b>
3.1 System Prerequisites . . . . .	9
3.2 Installation from PyPI . . . . .	10
3.3 Installation from sources . . . . .	10
3.4 Managing Dependencies . . . . .	10
<b>4 Basic usage</b>	<b>13</b>
4.1 Create a profile . . . . .	13
4.2 Implement job descriptions . . . . .	14
4.3 Run a test-suite . . . . .	14
<b>5 Known issues</b>	<b>15</b>
5.1 Error installing Pygit2 . . . . .	15
<b>6 Common workflow</b>	<b>17</b>
6.1 Per-project test infrastructure . . . . .	17
6.2 Dedicated benchmark repository . . . . .	17
<b>7 Running test-suites</b>	<b>19</b>
<b>8 Visualizing results</b>	<b>21</b>
8.1 Real-time progress reports . . . . .	21
8.2 Post-mortem analysis . . . . .	21
<b>9 Validation setup</b>	<b>23</b>
9.1 Generalities . . . . .	23
9.2 Structure . . . . .	23
<b>10 Configuration basic blocks</b>	<b>27</b>

10.1	Generalities . . . . .	27
10.2	Scope . . . . .	27
10.3	Blocks description . . . . .	27
<b>11</b>	<b>Profiles</b>	<b>31</b>
11.1	Generalitites . . . . .	31
11.2	Scope . . . . .	31
11.3	Building a new Profile . . . . .	31
11.4	Managing Profiles . . . . .	32
<b>12</b>	<b>Bank management</b>	<b>33</b>
12.1	About . . . . .	33
12.2	Create/manage/delete a bank . . . . .	33
12.3	Feed a bank with results . . . . .	33
<b>13</b>	<b>Sessions</b>	<b>35</b>
13.1	About . . . . .	35
13.2	Start a new session . . . . .	35
13.3	List sessions . . . . .	35
13.4	Cleanup . . . . .	35
<b>14</b>	<b>Utilities</b>	<b>37</b>
14.1	Run a single test . . . . .	37
14.2	Input correctness . . . . .	37
14.3	Discover tests & generate proper configurations . . . . .	37
14.4	Build & artifacts cleanup . . . . .	38
<b>15</b>	<b>Plugins</b>	<b>39</b>
15.1	About . . . . .	39
15.2	Create a new plugin . . . . .	39
15.3	Debug a plugin . . . . .	39
15.4	A special case: runtime Plugins . . . . .	39
<b>16</b>	<b>Input Examples</b>	<b>41</b>
16.1	TE nodes: The complete list . . . . .	41
16.2	Profile: The complete list . . . . .	43
<b>17</b>	<b>PCVS Orchestration</b>	<b>45</b>
17.1	Run Context initialization . . . . .	45
17.2	Test Load Generation . . . . .	45
17.3	Job scheduling . . . . .	45
17.4	Post-mortem & live Reporting . . . . .	45
<b>18</b>	<b>Contribution Guide</b>	<b>47</b>
<b>19</b>	<b>Package Documentation</b>	<b>49</b>
19.1	Subpackages . . . . .	49
19.2	Submodules . . . . .	118
19.3	Module contents . . . . .	119
<b>Python Module Index</b>		<b>121</b>
<b>Index</b>		<b>123</b>

**Parallel Computing Validation System** (shorten to **PCVS**) is a Validation Orchestrator designed by and for software at scale. Its primary target is HPC applications & runtimes but can flawlessly address smaller use cases. PCVS can help users to create their test scenarios and reuse them among multiples implementations, an high value when it comes to validating Programmation standards (like APIs & ABIs). No matter the number of programs, benchmarks, languages, or tech non-regression bases use, PCVS gathers in a single execution, and, with a focus on interacting with HPC batch managers efficiently, run jobs concurrently to reduce the time-to-result overall. Through basic YAML-based configuration files, PCVS handles more than hundreds of thousands of tests and helps developers to ensure code production is moving forward.

While PCVS is a validation engine, not providing any benchmarks on its own, it provides configurations to the most widely used MPI/OpenMP test applications (benchmarks & proxy apps), constituting a 300,000+ test base, offering a new way to compare implementations standard compliance.

PCVS acts as a bridge putting at scale regular test bases, by making the most of the parallelism brought by HPC architectures, able to run thousands of tests at the same time. This decrease in the time to result boosts productivity of development, leading to a better software. By separating tests (=code to execute) from application to validate, the effort of writing tests and setting validation platform is strongly reduced, for instance:

1. The dev team writes tests and describes scenario.
2. Cluster admins sets up the architecture for testing (compilers, tools...).
3. Q/A dept. executes applications on given resource.

Splitting the effort will enforce reusability and flexibility. In many cases, multiple test bases share same compilers, runtimes, tools or even machine partition, it is then convenient (for administration purposes) to gather at a higher level this type of responsibility.\*

Using PCVS may have a slight cost as test logic must be converted to a new semantics (not the tests themselves but the testing system behind it), without being destructive for the original tests (PCVS can go together with other test frameworks).

---

**Note:** PCVS does not act as a test framework on its own. It does not have all the features from a test reporting interface either. Its purpose is to provide a new approach when building test-bases to dissociate testers from tested, to increase flexibility, reusability, ease maintenance and speed up test scheduling to minimize time to result. It is compatible with multiple build systems and test frameworks and can be integrated in a non-destructive way inside already-existing projects.

---



---

CHAPTER  
ONE

---

## FEATURE OVERVIEW

Here is a quick overview of features that make PCVS a robust solution different than other validation engines.

### 1.1 Run the validation from anywhere

After installation, PCVS can be configured in a minute for the current directory:

```
$ pcvsc scan .
$ pcvsc run
```

### 1.2 Scale a test-suite depending on the target machine

From PCVS approach, a benchmarks is a collection of programs. coupled with a compiler and a launcher, sets of tests can be build to dynamically adapt the machine to test. With a single edition, a test-suite can be ported from a validating an MPI implementation on a simple workstation (no tests requiring more than one node & 4-8 MPI processes) to largest supercomputers (thousands of nodes). PCVS allow this thanks to **criterions**, a variadic component to apply to a program to build tests. It may be populated as follows in a profile:

```
criterion:
  iterators:
    number_of_mpi_processes:
      values: [1, 2, 4, 8, 16, 32]
```

### 1.3 Automatic Test-suite builder

PCVS relies on a specific test description syntax in order to build an efficient test-suite. To help with that process, PCVS can pre-generate templates:

```
$ pcvsc scan /dir
```

## **1.4 Definition - Execution - Reporting in one place**

One main advantage of PCVS is the capability to gather all validation modules in one single place, easy to install as a single user. Among others:

- Highly customizable test generation framework
- Orchestrator designed to run tests at scale
- Autonomous reporting web platform.
- Store results persistently as a Git repository for easy imports/exports & validation progression.

## GETTING STARTED

### 2.1 Installation

The simplest way to install PCVS is through **PyPI** repositories. It will fetch the lastest release but a specific version can be specified (detailed documentation in *Installation Guide*):

```
$ pip3 install pcvs
# OR
$ pip3 install pcvs<=0.5.0
$ pcvs
Usage: pcvs [OPTIONS] COMMAND [ARGS] ...

PCVS main program.
...
```

Full completion (options & arguments) is provide an can be activated with:

```
# ZSH support
$ eval "$(_PCVS_COMPLETE=zsh_source pcvs)"
# BASH support
$ eval "$(_PCVS_COMPLETE=bash_source pcvs)"
```

### 2.2 PCVS-formatted test-suite

#### 2.2.1 Test-suite layout

While PCVS is highly customizable, it comes with templates to locally test it without any prior knowledge. Before using PCVS, let's consider a provided test-suite as any `tests/` directory ( `all-reduce.c` & `wave.c` provided for convenience):

```
$ tree tests
tests
├── coll
│   └── all-reduce.c
└── pt2pt
    └── wave.c
2 directories, 2 files
```

PCVS needs rules to know how to parse the test-suite above to create tests. This will be done through `pcvs.yml` specification file. Such a file can be placed anywhere in the file tree. Consider putting it directly under the `tests/` directory for this example. Here is the content of this file:

---

**Note:** A test is the combination of a program, its arguments and the environment used to execute it. from PCVS' point of view, a test file does not carry the whole test environment. the orchestrator itself manage to build it directly from specification. Thus `pcvs.yml` expects the user to describe programs to be used to build the test-suite.

---

```
# put this in test/pcvs.yml:
all_reduce_test:
    build:
        files: ["coll/all-reduce.c"]
    run:
        program: "a.out"
pt2pt_test:
    build:
        files: ["pt2pt/wave.c"]
    run:
        program: "a.out"
```

This file specifies two root nodes referred as *Test Expressions* (TE) or *Test Descriptors* (TD). It contains subnodes describing how to build programs. A `build` gives informations about how to build the program. `files` (a list or a string) contains the whole list of files required to build the program (in case of a C file for instance). With no other information, PCVS will assume the program to be built with a compiler (no invocation to a build system here). A `run` subnode instructs PCVS to execute this program. The expected program name is `a.out`. This is the simplest way to integrate tests to PCVS. For a complete list of nodes to be used in a `pcvs.yml`, please consult [TE nodes: The complete list](#)

**Warning:** Beware of tabulations, YAML indentations only supports spaces !

## 2.3 Execute the test-suite

PCVS relies on (1) test specifications and (2) execution profile to create and execute a full benchmarks. Building a valid profile may be complex at first but offer a huge flexibility to solve complex validation scenarios. Still, most scenarios share similarities, like, in that case, running MPI programs. PCVS comes with default profiles for default scenarios. Here, we select the `mpi` base profile to build our own:

```
$ pcvs profile create user.my-profile --base mpi
$ pcvs profile list
```

By specifying `user.my-profile`, it will save the profile under `~/.pcvs/` and make it available for the whole \$USER, no matter the current working directory used when running PCVS. To learn more about profile scope, please see [Scope](#).

---

**Note:** As this profile uses MPI, we need to source an MPI implementation in the environment. Please use the method suiting your needs (spack/module/source). If interested by autoloading spack-or-module-based MPI implementation, please read [Profiles](#).

---

Now, start PCVS. You must provide the profile & the directory where tests are located:

```
$ pcvs run --profile my-profile ./tests/
```

---

**Note:** the user. prefix to the profile name may be removed as there is no name ambiguity, PCVS will detect the proper scope.

---

## 2.4 Access the results

Results are stored in `$PWD/.pcvs-build/rawdata/*.json` by default. the default output directory may be changed with `pcvs run -output`. JSON files can directly processed by third-party tools. The `scheme` can be used to update the input parser with compliant output. Currently PCVS only provides specific JSON format. It is planned to support common validation format (like JUnit).

If no third-party tool is available, PCVS comes with a lightweight web server (=Flask) to serve results in a web browser:

```
# where pcvs run has been run:  
$ pcvs report  
# OR you may specify the run path  
$ pcvs report <path>
```

Then, browse `http://localhost:5000/` to browse your results.



## INSTALLATION GUIDE

### 3.1 System Prerequisites

PCVS requires **python3.5+** before being installed. We encourage users to use virtual environment before installing PCVS, especially if targeting a single-user usage. To create a virtual environment, create & activate it **before** using pip3. Please check [venv](#) (native), [virtualenv](#) (external package) or even [pyenv](#) (third-party tool) to manage them.

Here some quickstarts for each approach:

```
$ python3 -m venv ./env/  
# to use it:  
$ source env/bin/activate  
# Work in virtual environment  
$ deactivate
```

```
# install first:  
$ pip3 install virtualenv  
$ virtualenv ./env/  
# to use it:  
$ source ./env/bin/activate  
# work...  
$ deactivate
```

```
# install first:  
$ curl https://pyenv.run | bash  
$ pyenv virtualenv my_project  
# to use it:  
$ pyenv activate my_project  
# work...  
$ pyenv deactivate
```

## 3.2 Installation from PyPI

The simplest way to install PCVS is through **PyPI** repositories. It will fetch the lastest release but a specific version can be specified:

```
$ pip3 install pcvs
# OR
$ pip3 install pcvs<=0.5.0
```

## 3.3 Installation from sources

The source code is also available on Github, based on Setuptools, the manual installation is pretty simple. The latest release (and any previous archive) is also available on the [website](#). To checkout the latest release:

```
$ git clone https://github.com/cea-hpc/pcvs.git pcvs_latest
$ pip3 install ./pcvs_latest
# OR
$ python3 ./pcvs_latest/setup.py install
```

## 3.4 Managing Dependencies

Installing only dependencies come in two way: `requirements.txt` gathers production-side deps, required for PCVS to work, while `requirements-dev.txt` contains (in addition to the base) the validation toolkit (pytest, coverage, etc.):

```
$ pip3 install -r requirements.txt
$ pip3 install -r requirements-dev.txt
# allowing to use:
$ coverage run
```

### 3.4.1 Dealing with offline networks

In some scenarios, it may not be possible to access PyPI mirrors to download dependencies (or even PCVS itself). Procedures below will describe how to download dep archives locally on a machine with internet access and then make them available for installation once manually moved to the ‘offline’ network. It consists in two steps. First, download the deps and create and archive (considering the project is already cloned locally):

```
$ git clone https://github.com/cea-hpc/pcvs.git # if not already done
$ pip3 download . -d ./pcvs_deps
# OR, select a proper requirements file
$ pip3 download -r requirements-dev.txt -d ./pcvs_deps
$ tar czf pcvs_deps.tar.gz ./pcvs_deps
```

Once the archive moved to the offline network (=where one wants to install PCVS), we are still considering PCVS is cloned locally:

```
$ tar xf ./pcvs_deps.tar.gz
$ pip3 install . --find-links ./pcvs_deps --no-index
# or any installation variations (-e ...)
```

**Warning:** Please use extra caution when using this method with different architectures between source & destination. By default, pip will download source-compatible wheel/source package, which may not be suited for the target machine.

pip provides options to select a given platform/target python version, which differ from the current one. Note in that case no intermediate source package will be used, only distributed versions (compiled one). To ‘accept’ it, you must specify `--only-binary=:all:` to force downloading distribution packages (but will fail if not provided) or `--no-deps` to exclude any dependencies to be downloaded (and should be taken care manually):

```
$ pip3 download -r ... -d ... --platform x86_64 --python-version 3.5.4 [--only-binary=:all:|--no-deps]
```

### 3.4.2 Important note

- PCVS requires Click >= 8.0, latest versions changed a critical keyword (to support completion) not backward compatible. Furthermore, Flask also have a dep to Click>7.1.
- To manage dict-based configuration object, PCVS relies on [Addict](#). Not common, planned to be replaced but still required to ease configuration management process through PCVS.
- Banks are managed through Git repositories. Thus, PCVS relies on [pygit2](#). One major issue is when pygit2 deployment requires to be rebuilt, as a strong dep to libgit2 development headers is required and may not be always provided. As a workaround for now:
  - Install a more recent pip version, able to work with wheel package (>20.x). This way, the pygit2 package won’t have to be reinstalled.
  - install libgit2 headers manually

---

**Note:** A quick fix to install pygit2/libgit2 is to rely on [Spack](#). Both are available for installation: [libgit2](#) & [py-pygit2](#). Be sure to take a proper version above **1.x**.

---



---

## CHAPTER FOUR

---

## BASIC USAGE

Once PCVS is installed through the *Installation Guide*, the `pcvs` is available in PATH. This program is the only entry point to PCVS:

```
$ pcvs
Usage: pcvs [OPTIONS] COMMAND [ARGS]...
PCVS main program.
```

---

### – Options

```
| –verbose -v INTEGER RANGE Enable PCVS debug verbosity (cumulative) | | [env var: PCVS_VERBOSE]
| –color -c/-no-color Use colors to beautify the output [env var: PCVS_COLOR] | | –glyph -g/-no-glyph enable/disable Unicode glyphs [env var: PCVS_ENCODING] | | –exec-path -C DIRECTORY [env var: PCVS_EXEC_PATH] | | –version -V Display current version | | –width -w INTEGER Terminal width (autodetection if omitted | | –plugin-path -P PATH Default Plugin path prefix [env var: PCVS_PLUGIN_PATH] | | –plugin -m TEXT | | –help -h,-help Show this message and exit. |
```

---

### – Commands

```
| bank Persistent data repository management | | check Ensure future input will be conformant to standards | |
| clean Remove artifacts generated from PCVS | | config Manage Configuration blocks | | exec Running as-
| specific test | | profile Manage Profiles | | report Manage PCVS result reporting interface | | run Run a
| validation | | scan Analyze directories to build up test conf. files | | session Manage multiple validations |
```

---

## 4.1 Create a profile

A profile contains the whole PCVS configuration in a single place. While this approach allow deeply complex approaches, we'll target a simple MPI implementation for this example. To create the most basic profile able to run MPI programs, we may herit ours from pre-generated called a template:

```
$ pcvs profile create -t mpi user.myprofile
```

This profile can not be references with `user.myprofile` (or `myprofile` in short, where there are no possible conflicts). This profile will be available at user-level scope. It is also possible to set this profile as `local` (only for the current directory) or `global` (anyone using the PCVS installation). You may replace `newprofile` by a name of your choice. For a complete list of available templates, please check `pcvs profile list --all`.

A profile can be edited if necessary with `pcvs profile edit newprofile`. It will open an `$EDITOR`. When exiting, the profile is validated to ensure coherency. In case it does not fulfill a proper format, a rejection file is crated in the current directory. Once fixed, the profile can be saved as a replacement with:

```
$ pcvs profile import newprofile --force --source file.yml
```

**Warning:** The `--force` option will overwrite any profile with the same name, if it exists. Please use this option with care. In case of a rejection, the import needs to be forced in order to replace the old one.

## 4.2 Implement job descriptions

For a short example of implementing test descriptions, please refer to the *Test-suite layout* shown in the Getting-Started guide. A more detailed presentation of test description capabilities is available in its own documentation page.

The most basic `pcvs.yml` file may look like this:

```
my_program:
  build:
    files: ['main.c']
  run:
    program: ['a.out']
```

PCVS also supports building programs through Make, CMake & Autotools, each system having its own set of keys to configure:

- `build.make.target`: allow to configure a Make target to invoke.
- `build.cmake.vars`: variables to forward to cmake (to be prefixed w/ -D)
- `build.autotools.params`: configure script flags
- `build.autotools.autogen`: boolean whether to execute autogen.sh first.

Proper YAML formats can be checked before running a test-suite with:

```
$ pcvs check --directory /path/to/dir
$ pcvs check --profiles
```

## 4.3 Run a test-suite

Start a run from the local directory with our profile is as simple as:

```
$ pcvs run --profile newprofile
```

A list of directories can also be given. Once started, the validation process is logged under `$PWD/.pcvs-build` directory. If the directory already exists, it is cleaned up and reused. A lock is put in that directory to protect against concurrent PCVS execution in the same directory.

## KNOWN ISSUES

### 5.1 Error installing Pygit2

Status: **Unresolved**

**Typical Error message:** While running `pip3 install`, Such error messages might be raised:

```
-> error: git2.h: No such file or directory
-> error: #error You need a compatible libgit2 version (1.1.x)
```

#### 5.1.1 Solution / Workaround

- Manually install libgit2, version  $\geq 1.0.0$
- Use wheel package to install third-party tools (only Python3.7+)



## COMMON WORKFLOW

### 6.1 Per-project test infrastructure

### 6.2 Dedicated benchmark repository

#### 6.2.1 Benchmark description

PCVS is a program built for current HPC structures, it allows the launch of programs with a large coverage of parameters. Moreover, PCVS allows users to log in and out streams and handle sessions. In order to do that, PCVS needs an exhaustive configuration handled in files and profiles.

#### 6.2.2 Validation profile

Validation profiles are configuration files used at launch in pcvs run. A PCVS validation profile can be generated with the following command :

```
pcvs profile build tutorial_profile
```

A pcvs profile is made of blocks that can be customized, you can export a profile to a file :

```
pcvs profile export tutorial_profile tutorial_profile.yml
```

you should have a tutorial\_profile.yml file which has the following nodes :

- compiler
- criterion
- group
- machine
- runtimes

### 6.2.3 Launch tests

Tests launches are done in this case by the following command :

```
pcvs run -f -p tutorial_profile .
```

The generic command is :

```
pcvs run [option] -p [profile] [directory]
```

PCVS will scan the target directory and find any “pcvs.yml” or “pcvs.setup” file within the directory or its subdirectories, and launch the benchmark on the corresponding files.

“pcvs.setup” files must return a yaml-structured character string describing a pcvs configuration described in pcvs.yml files.

The pcvs run configuration is also structured in nodes, here is a typical example:

```
tutorial_test:  
  build:  
    files: "@SRCPATH@/tutorial_program.c"  
    sources:  
      binary: "tutorial_binary"  
  run:  
    program: "tutorial_binary"
```

With a directory like such :

```
└── pcvs.yml  
    └── tutorial_program.c
```

The run will :

- build `tutorial_binary` by compiling `tutorial.c` using `gcc` (as specified earlier)
- run the `tutorial_binary` file

Many other options are available such as tags, flags, etc, these are referenced in the documentation of PCVS.

### 6.2.4 Visualize results

PCVS owns an html report generator, it can be used with :

```
pcvs report
```

`pcvs report` must be used on a directory on which tests have been run.

---

**CHAPTER  
SEVEN**

---

**RUNNING TEST-SUITES**



---

**CHAPTER  
EIGHT**

---

**VISUALIZING RESULTS**

**8.1 Real-time progress reports**

**8.2 Post-mortem analysis**



## VALIDATION SETUP

### 9.1 Generalities

#### 9.1.1 Setup files

Like profiles, setup configurations have nodes to describe different steps of the process. These nodes are splitted into subnodes to describe the course of the run.

The validation configuration is specified using setup files. These files can be in the yml format, or be an executable files generating a yml configuration in stdout. The informations of this configuration are crossed with the profile informations to

When PCVS is launched in a directory, it browses every subdirectory to find any `pcvs.yml` or `pcvs.setup` file and launches itself with the corresponding configuration.

#### example

```
examplmtree/
└── subdir1
    └── pcv.yml
└── subdir2
    └── pcv.yml
```

Launching ``pcvs run examplmtree`` will generate tests for `subdir1/pcv.yml` **and** for `subdir2/pcv.yml`. There is no need to put a setup configuration in the root of `examplmtree`, but it is possible to add a setup here.

### 9.2 Structure

The yml input must have one node per test. Each test can describe the following configurations :

- build
- run
- validate
- group
- tag
- artifact

### 9.2.1 Build

The build node describe how a binary file should be built depending on its sources. It contains the following subnodes :

```
build:
    files: path/to/the/file/to/build
    sources:
        binary: name of the binary to be built (if necessary)
        depends_on: ["list of test names it depends on"]

    cflags: extra cflags
    ldflags: extra ldflags
    cwd: directory where the binary should be built
    variants: [list of variants (CF Configuration basic blocks -> compiler
node)]

    autotools:
        params: [list of options for autotools]
    cmake:
        params: [list of options for cmake]
    make:
        target: target for make command
```

### 9.2.2 Run

The run node describes how a binary file should be launched. It contains the following nodes :

```
run:
    cwd: path to build directory
    depends_on:
        test: [list of tests on which it depends]
        spack: [list of spack dependencies used by this test]
        module: [list of installed modules this test needs]
    program: name of the binary file
```

The run node owns the `iterate` subnode which can contain custom iterators desribed in the `criterion` node in the selected profile. Moreover, the `run.iterate` node can define custom iterators without defining them in `criterion` by writing them in the `run.iterate.program` node.

```
run:
    iterate:
        iterator_described_in_profile.runtime.criterion:
            values: [list of values for the corresponding iterator]
    program:
        custom_iterator:
            numeric: true/false
            type: "argument" or "environment"
            values: [list of values taken by the iterator]
            subtitle: string chosen to identify this iterator
```

### 9.2.3 Validate

The validate node describes the expected test behaviour, including exit, time and matching output.

```
validate:
  expect_exit: expected exit code (integer)
  time:
    mean: expected time to compute the test (seconds / float) tolerance:
    standard deviation for expected time (seconds / float)
    kill_after: maximum time after which process has to be killed
    (seconds / float)
  match:
    label:
      expr:
      expect:
  script:
    path: Path to a validating script
```

### 9.2.4 Group

Groups are described in profiles. They can contain build, run, tag, validate, and artifact subnodes. Once a group is defined in the used profile it can be called in the validation setup file.

```
group: name of the group defined in the profile
```

### 9.2.5 Tag

Tags get in the results and tests can be sorted tag-wise. A test can have multiple tags and tags do not have to be defined upstream.

```
tag:
  - tag1
  - tag2
```

### 9.2.6 Artifact

The artifact node contains anything the output should have in addition to the results of tests.

```
artifact:
  obj1: "path/to/obj1"
  obj2: "path/to/obj2"
```



## CONFIGURATION BASIC BLOCKS

### 10.1 Generalities

Configuration blocks define settings for PCVS. There are 5 configurable blocks which are :

- compiler
- criterion
- group
- machine
- runtime

The configuration block is a virtual object, it doesn't exist per se, configuration blocks are used to build profiles which can be imported/exported. It is possible however to share configuration blocks by addressing them in a scope that is large enough to reach other users.

Each configuration block contains sub-blocks in order to isolate and classify informations.

### 10.2 Scope

PCVS allows 3 scopes :

- **global** for everyone on the machine having access to the PCVS installation
- **user** accessible from everywhere for the corresponding user
- **local** accessible only from a directory

### 10.3 Blocks description

#### 10.3.1 compiler node

The compiler node describes the building sequence of tests, from the compiler command to options, arguments, tags, libraries, etc.

This node can contain the subnodes **commands** and **variants**

## commands

The compiler.commands block contains a collection of compiler commands.

```
cc: compilation command for C code
cxx: compilation command for C++ code
f77: compilation command for Fortran77 code
f90: compilation command for Fortran90 code
fc: compilation command for generic Fortran code
```

## variants

The variants block can contain any custom variant. The variant must have a **name**, and **arguments** as such :

```
example_variant:
    args: additionnal arguments for the example variant
openmp:
    args: -fopenmp
strict :
    args: -Werror -Wall -Wextra
```

In this example the variants “example\_variant”, “openmp”, and “strict” have to be specified in the validation setup where the user wants to use them.

### 10.3.2 criterion node

the criterion node contains a collection of iterators that describe the tests. PCVS can iterate over custom parameters as such :

```
iterators :
    n_[iterator] :
        **subtitle** : string used to indicate the number of [iterator] in
                      the test description

        **values** : values that [iterator] allowed to take
```

## Example

```
iterators:
    n_core:
        subtitle: C
        values:
            - 1
            - 2
```

In this case the program has to iterate on the core number and has to take the values 1 and 2. The name **n\_core** is arbitrary and has to be put in the validation setup file.

### 10.3.3 group node

The group node contains group definitions that describe tests. A group description can contain any node present in the Configuration basic blocks (CF *Validation Setup* section).

#### Example

```
GRPMPI:
  run:
    iterate:
      n_omp:
        **values**: null
```

### 10.3.4 machine node

The machine node describes the constraints of the physical machine.

#### machine :

- nodes : number of accessible nodes
- cores\_per\_node : number of accessible cores per node
- concurrent\_run : maximum number of processes that can coexist

### 10.3.5 runtime node

The runtime node specifies entries that must be passed to the launch command. It contains subnodes such as args, `iterators, etc. The iterator node contains arguments passed to the launching command. For example, if prterun takes the “-np” argument, which corresponds to the number of MPI threads, let’s say n\_mpi, we will get the following runtime profile :

args : arguments for the launch command

```
iterators:
  n_mpi:
    numeric : true

    option : "-np"

    type : argument

    aliases :
      [dictionary of aliases for the option]

plugins
```



## PROFILES

### 11.1 Generalities

A PCVS profile defines a configuration in which PCVS will launch. This configuration is divided in nodes, and it can be customized within pcv or in yaml files that can be imported/exported via the command line interface.

This configuration is separated in 5 nodes :

- compiler
- criterion
- group
- machine
- runtimes

each node is separated in subnodes and can be defined separately in multiple ways. As files, profiles are written with the yml syntax.

### 11.2 Scope

Profiles can have different scopes depending on which user should have access or which project should be affected by it. The 3 scopes are the following :

- Local : The profile is only seeable from a specific folder
- User : The profile is seeable from everywhere in an userspace.
- Global : The profile is accessible to everyone and from everywhere.

### 11.3 Building a new Profile

To create a blank profile, one can use the command :

```
pcvs profile build example_profile
```

To export this profile in a file format, use the command :

```
pcvs profile export example_profile example_profile.yml
```

This profile is fully customizable with any text editor, to import the profile back into PCVS use the command :

```
pcvs profile import example_profile example_profile.yml
```

Without arguments, the `pcvs profile build` command builds blocks as default, but a profile can be built with custom configuration blocks.

### **11.3.1 Building a profile with existing configuration blocks**

Buiding a profile based on configuration blocks can be done in two ways :

- in the CLI
- with the interactive mode

#### **In the CLI**

```
pcvs profile build example_profile -b [scope].[block-name].[block-type]
```

This command has to include either 0 or 5 -b blocks (default or complete configuration).

#### **With the interactive mode**

```
pcvs profile build -i
```

For the configuration blocks setting please refer to the Configuration blocks section.

## **11.4 Managing Profiles**

Besides being built, exported or imported, profiles can be altered or destroyed with the corresponding commands.

Use `pcvs profile list` to see every available profiles.

`pcvs profile alter [profile]` launches a text editor in order to manually change the profile. PCVS scans editors to give a choice to users.

`pcvs profile destroy [profile]` deletes a profile

TBW

Using Profiles

Profiles are used at runtime, they are specified with the `-p` option.

```
pcvs run -p example_profile
```

---

CHAPTER  
**TWELVE**

---

**BANK MANAGEMENT**

**12.1 About**

**12.2 Create/manage/delete a bank**

**12.3 Feed a bank with results**



---

CHAPTER  
THIRTEEN

---

SESSIONS

### 13.1 About

### 13.2 Start a new session

```
$ pcvs run <...> # start an interactive session
$ pcvs run <...> --detach # start a background session
```

### 13.3 List sessions

```
$ pcvs session
$ pcvs session -l
```

### 13.4 Cleanup

Only for background sessions:

```
$ pcvs session --ack <sid>
$ pcvs session --ack-all # only completed sessions
```



---

CHAPTER  
FOURTEEN

---

UTILITIES

## 14.1 Run a single test

```
$ pcvx exec <fully-qualified test-name>
$ pcvx exec --show [cmd|env|loads|all] <test-name>
```

## 14.2 Input correctness

### 14.2.1 Profiles

```
$ pcvx check --profiles
```

### 14.2.2 Configuration blocks

```
$ pcvx check --configs
```

### 14.2.3 Test-suites

```
$ pcvx check --directory <path>
```

## 14.3 Discover tests & generate proper configurations

```
$ pcvx scan <path>
```

## 14.4 Build & artifacts cleanup

```
$ pcvx clean --dry-run # list content to be deleted, recursively  
$ pcvx clean --force # actually deleting content
```

## 15.1 About

### 15.1.1 Possible passes

- START\_BEFORE
- START\_AFTER
- TFILE\_BEFORE
- TDESC\_BEFORE
- TEST\_EVAL
- TDESC\_AFTER
- TFILE\_AFTER
- SCHED\_BEFORE
- SCHED\_SET\_BEFORE
- SCHED\_SET\_EVAL
- SCHED\_SET\_AFTER
- SCHED\_PUBLISH\_BEFORE
- SCHED\_PUBLISH\_AFTER
- SCHED\_AFTER
- END\_BEFORE
- END\_AFTER

## 15.2 Create a new plugin

## 15.3 Debug a plugin

## 15.4 A special case: runtime Plugins



## INPUT EXAMPLES

### 16.1 TE nodes: The complete list

```
1  ---
2  .this_is_a_job:
3      group: "GRPSERIAL"
4      tag: ["a", "b"]
5      build:
6          # source files to include (if needed)
7          files: "@SRCPATH@/*.c"
8          autotools:
9              args: ['--disable-bootstrap']
10             envs: []
11             cmake:
12                 vars: ['CMAKE_VERBOSE_MAKEFILE=ON']
13                 args: []
14                 envs: []
15             make:
16                 target: "all"
17                 args: ['MPIICC=@COMPILER_CC@']
18                 envs: []
19             sources:
20                 # program binary (if needed)
21                 binary: "a.out"
22                 # extra cflags
23                 cflags: "extra cflags"
24                 # extra ldflags
25                 ldflags: "extra ldflags"
26                 envs: []
27             custom:
28                 program: "myscript.sh"
29                 envs: ["VAR1=value", "VAR2=value"]
30             # dependency scheme
31             depends_on: ["this_is_another_test"]
32             # directory where program should be built
33             cwd: "dir/to/build"
34             # variants describing the job.
35             # available variants are provided by compiler.$LANG
36             variants:
37                 - openmp
```

(continues on next page)

(continued from previous page)

```

38     - accel
39     - mpi
40   attributes:
41     command_wrap: true
42
43   run: &run_part
44     program: "./a.out"
45   attributes:
46     command_wrap: true
47     path_resolution: false
48   iterate:
49     # runtime iterators
50     n_mpi:
51       values: [2, 4]
52     n_omp:
53       values: [1, 2]
54   program:
55     # name will be used as part of final test-name
56   give_it_a_name:
57     numeric: true
58     type: "argument"
59     values: ["-iter 1000", "-fast"]
60     subtitle: "lol"
61   # directory where program should be built
62   cwd: "dir/to/build"
63   # dependency scheme
64   depends_on: ["this_is_another_run_test_in_the_same_file"]
65   package_manager:
66     spack:
67       - protobuf@3.1.1
68       - gcc@7.3.0
69     module:
70       - protobuf
71       - gnu/gcc/7.3.0
72   artifact:
73     # relative to $BUILDPATH
74     obj1: "./path/1"
75     obj2: "./path/2"
76
77   metrics:
78     metric1:
79       key: "regex"
80     metric2:
81       key: "regex"
82
83   # this is a copy/paste from pav2
84   validate:
85     expect_exit: 0
86     time:
87       mean: 10.0
88       tolerance: 2.0
89       kill_after: 20

```

(continues on next page)

(continued from previous page)

```
90  match:
91    label:
92      expr: '^\\d+(\\.\\d+) received$'
93      expect: true|false
94    label2: 'Total Elapsed: \\d+\\.\\d+ sec.$'
95  analysis:
96    method: "<method>"
97  script:
98    path: "/path/to/script"
99
100 #####
101
102 # depicts an inheritance mechanism.
103 real_test:
104   build:
105     make:
106       target: all
107   run:
108     <<: *run_part
```

## 16.2 Profile: The complete list



---

CHAPTER  
**SEVENTEEN**

---

## **PCVS ORCHESTRATION**

**17.1 Run Context initialization**

**17.2 Test Load Generation**

**17.3 Job scheduling**

**17.4 Post-mortem & live Reporting**



---

CHAPTER  
**EIGHTEEN**

---

**CONTRIBUTION GUIDE**



## PACKAGE DOCUMENTATION

### 19.1 Subpackages

#### 19.1.1 pcvs.backend package

##### Submodules

###### pcvs.backend.bank module

pcvs.backend.bank.BANKS: `Dict[str, str] = {}`

##### Variables

`BANKS` – list of available banks when PCVS starts up

`class pcvs.backend.bank.Bank(path=None, token="")`

Bases: Bank

Representation of a PCVS result datastore.

Stored as a Git repo, a bank hold multiple results to be scanned and used to analyse benchmarks result over time. A single bank can manipulate namespaces (referred as ‘projects’). The namespace is provided by suffixing @proj to the original name.

##### Parameters

- `root (str)` – the root bank directory
- `repo (Pygit2.Repository)` – the Pygit2 handle
- `config (MetaDict)` – when set, configuration file of the just-submitted archive
- `rootree (Pygit2.Object)` – When set, root handler to the next commit to insert
- `locked (bool)` – Serialize Bank manipulation among multiple processes
- `proj_name (str)` – extracted default-proj from initial token

`build_target_branch_name(tag=None, hash=None)`

Compute the target branch to store data.

**This is used to build the exact Git branch name based on:**

- default-proj
- unique profile hash, used to run the validation

**Parameters**

**tag** (`str`) – overridable default-proj (if different)

**Returns**

fully-qualified target branch name

**Return type**

`str`

**property default\_project: str**

Get project set as default when none are provided.

**Returns**

the project name (as a Ref branch)

**Return type**

`str`

**exists()**

Check if the bank is stored in PATH\_BANK file.

Verification is made either on name **or** path.

**Returns**

True if both the bank exist and globally registered

**Return type**

`bool`

**get\_count()**

Get the number of projects managed by this bank handle.

**Returns**

number of projects

**Return type**

`int`

**property name: str**

Get bank name.

**Returns**

the exact label (without default-project suffix)

**Return type**

`str`

**name\_exist()**

Check if the bank name is registered into PATH\_BANK file.

**Returns**

True if the name (lowered) is in the keys()

**Return type**

`bool`

**path\_exist()**

Check if the bank path is registered into PATH\_BANK file.

**Returns**

True if the path is known.

**Return type**`bool`**property prefix:** `Optional[str]`

Get path to bank directory.

**Returns**

absolute path to directory

**Return type**`str`**save\_from\_archive**(*tag*, *archivepath*, *msg=None*)

Extract results from the archive, if used to export results.

This is basically the same as `Bank.save_from_buildir()` except the archive is extracted first.

**Parameters**

- **tag** (`str`) – overridable default project (if different)
- **archivepath** (`str`) – archive path

**Return type**`None`**save\_from\_buildir**(*tag*, *buildpath*, *msg=None*)

Extract results from the given build directory & store into the bank.

**Parameters**

- **tag** (`str`) – overridable default project (if different)
- **buildpath** (`str`) – the directory where PCVS stored results

**Return type**`None`**save\_new\_run**(*target\_project*, *path*)**Return type**`None`**save\_new\_run\_from\_instance**(*target\_project*, *hdl*, *msg=None*)

Create a new node into the bank for the given project, based on the open result handler.

**Parameters**

- **target\_project** (`str`) – valid project (=branch)
- **hdl** (class:*BuildDirectoryManager*) – the result build directory handler

**Return type**`None`**save\_to\_global()**

Store the current bank into PATH\_BANK file.

**Return type**`None`

**show(stringify=False)**  
Print the bank on stdout. :rtype: `Optional[str]`

---

**Note:** This function does not use `log.IOManager`

---

`pcvs.backend.bank.add_banklink(name, path)`

Store a new bank to the global system.

**Parameters**

- `name (str)` – bank label
- `path (str)` – path to bank directory

**Return type**

`None`

`pcvs.backend.bank.flush_to_disk()`

Update the PATH\_BANK file with in-memory object.

**Raises**

`IOError` – Unable to properly manipulate the tree layout

**Return type**

`None`

`pcvs.backend.bank.init()`

Bank interface detection.

Called when program initializes. Detects defined banks in PATH\_BANK

**Return type**

`None`

`pcvs.backend.bank.list_banks()`

Accessor to bank dict (outside of this module).

**Returns**

dict of available banks.

**Return type**

`dict`

`pcvs.backend.bank.rm_banklink(name)`

Remove a bank from the global management system.

**Parameters**

`name (str)` – bank name

**Return type**

`None`

## pcvs.backend.config module

**class** `pcvs.backend.config.ConfigurationBlock(kind, name, scope=None)`

Bases: `object`

Handle the basic configuration block, smallest part of a profile.

From a user perspective, a basic block is a dict, gathering in a Python object informations relative to the configuration of a single component. In PCVS, there is 5 types of components:

- Compiler-oriented (defining compiler commands)
- Runtime-oriented (setting runtime & parametrization)
- Machine-oriented (Defining resources used for execution)
- Group-oriented (used a templates to globally describe tests)
- Criterion-oriented (range of parameters used to run a test-suite)

This class helps to manage any of these config blocks above. The distinction between them is carried over by an instance attribute `_kind`.

**Note:** This object can easily be confused with `system.Config`. While ConfigurationBlocks are from a user perspective, `system.Config` handles the internal configuration tree, on which runs rely. Nonetheless, both could be merged into a single representation in later versions.

### Parameters

- `_kind (str)` – which component this object describes
- `_name (str)` – block name
- `details (dict)` – block content
- `_scope (str)` – block scope, may be None
- `_file (str)` – absolute path for the block on disk
- `_exists (bool)` – True if the block exist on disk

**check(allow\_conversion=True)**

Validate a single configuration block according to its scheme.

### Return type

`None`

**clone(clone)**

Copy the current object to create an identical one.

Mainly used to mirror two objects from different scopes.

### Parameters

`clone (ConfigurationBlock)` – the object to mirror

### Return type

`None`

**delete()**

Delete a configuration block from disk

**Return type**

None

**display()**

Configuration block pretty printer

**Return type**

None

**dump()**

Convert the configuration Block to a regular dict.

This function first load the last version, to ensure being in sync.

**Returns**

a regular dict() representing the config blocK

**Return type**

dict

**edit()**

Open the current block for edition.

**Raises**

**Exception** – Something occured on the edited version.

**Parameters**

**e** (*str*) – the EDITOR to use instead of default.

**Return type**

None

**edit\_plugin()**

Special case to handle ‘plugin’ key for ‘runtime’ blocks.

This allows to edit a de-serialized version of the ‘plugin’ field. By default, data are stored as a base64 string. In order to let user edit the code, the string need to be decoded first.

**Parameters**

**e** (*str*) – the editor to use instead of defaults

**Return type**

None

**fill(*raw*)**

Populate the block content with parameters.

**Parameters**

**raw** (*dict*) – the data to fill.

**Return type**

None

**flush\_to\_disk()**

write the configuration block to disk

**Return type**

None

**property full\_name: str**

Return complete block label (scope + kind + name)

**Returns**

the fully-qualified name.

**Return type**

`str`

**`is_found()`**

Check if the current config block is present on fs.

**Returns**

True if it exists

**Return type**

`bool`

**`load_from_disk()`**

load the configuration file to populate the current object.

**Raises**

- `BadTokenError` – the scope/kind/name tuple does not refer to a valid file.
- `NotFoundError` – The target file does not exist

**Return type**

`None`

**`load_template(name=None)`**

load from the specific template, to create a new config block

**Return type**

`None`

**`property ref_file: str`**

Return filepath associated with current config block.

**Returns**

the filepath, may be `None`

**Return type**

`str`

**`retrieve_file()`**

Associate the actual filepath to the config block.

From the stored kind, scope, name, attempt to detect configuration block on the file system (i.e. detected during module init())

**Return type**

`None`

**`property scope: str`**

Return block scope.

**Returns**

the scope, resolved if needed.

**Return type**

`str`

**`property short_name: str`**

Return the block label only.

**Returns**

the short name (may conflict with other config block)

**Return type**

`str`

`pcvs.backend.config.check_valid_kind(s)`

Assert the parameter is a valid kind.

Kind are defined by CONFIG\_BLOCKS module attribute.

**Parameters**

`s (str)` – the kind to validate

**Raises**

- `BadTokenError` – Kind is None
- `BadTokenError` – Kind is not in allowed values.

**Return type**

`bool`

`pcvs.backend.config.init()`

Load configuration tree available on disk.

This function is called when PCVS starts to load 3-scope configuration trees.

**Return type**

`None`

`pcvs.backend.config.list_blocks(kind, scope=None)`

Get available configuration blocks, as present on disk.

**Parameters**

- `kind` (`str`, one of CONFIG\_BLOCKS values) – configBlock kind (see CONFIG\_BLOCKS for possible values)
- `scope` (`'user'`, `'global'` or `'local'`, `optional`) – where the configblocks is located, defaults to None

**Returns**

list blocks with specified kind, restricted by scope (if any)

**Return type**

`dict` of config blocks

`pcvs.backend.config.list_templates()`

List available templates to be used for boostraping config. blocks.

**Returns**

a list of valid templates.

**Return type**

`list`

## pcvs.backend.profile module

**class** `pcvs.backend.profile.Profile(name, scope=None)`

Bases: `object`

A profile represents the most complete object the user can provide.

It is built upon 5 components, called configuration blocks (or basic blocks), one of each kind (compiler, runtime, machine, criterion & group) and gathers all required information to start a validation process. A profile object is the basic representation to be manipulated by the user.

---

**Note:** A profile object can be confused with `pcvs.helpers.system.MetaConfig`. While both are carrying the whole user configuration, a Profile object is used to build/manipulate it, while a Metaconfig is the actual internal representation of a complete run config.

---

### Parameters

- `_name` (`str`) – profile name, should be unique for a given scope
- `_scope` (`str`) – profile scope, allowed values in `storage_order()`, defaults to None
- `_exists` (`bool`) – return True if the profile exists on disk.
- `_file` (`str`) – profile file absolute path

**check(allow\_legacy=True)**

Ensure profile meets scheme requirements, as a concatenation of 5 configuration block schemes.

### Raises

`FormatError` – A ‘kind’ is missing from profile OR incorrect profile.

**clone(clone)**

Duplicate a valid profile into the current one.

### Parameters

`clone` (`Profile`) – a valid profile object

**property compiler**

Access the ‘compiler’ section.

### Returns

the ‘compiler’ dict segment

### Return type

`dict`

**property criterion**

Access the ‘criterion’ section.

### Returns

the ‘criterion’ dict segment

### Return type

`dict`

**delete()**

Remove the current profile from disk.

It does not destroy the Python object, though.

**display()**

Display profile data into stdout/file.

**dump()**

Return the full profile content as a single regular dict.

This function loads the profile on disk first.

**Returns**

a regular dict for this profile

**Return type**

`dict`

**edit()**

Open the editor to manipulate profile content.

**Parameters**

`e` (`str`) – an editor program to use instead of defaults

**Raises**

`Exception` – Something happened while editing the file

**Warning:** If the edition failed (validation failed) a rejected file is created in the current directory containing the rejected profile. Once manually edited, it may be submitted again through `pcvs profile import`.

**edit\_plugin()**

Edit the ‘runtime.plugin’ section of the current profile.

**Parameters**

`e` (`str`) – an editor program to use instead of defaults

**Raises**

`Exception` – Something happened while editing the file.

**Warning:** If the edition failed (validation failed) a rejected file is created in the current directory containing the rejected profile. Once manually edited, it may be submitted again through `pcvs profile import`.

**fill(`raw`)**

Update the given profile with content stored in parameter.

**Parameters**

`raw` (`dict`) – tree of (key, values) pairs to update

**flush\_to\_disk()**

Write down profile to disk.

Also, ensure the filepath is valid and profile content is compliant with schemes.

**property full\_name**

Return fully-qualified profile name (scope + name).

**Returns**

the unique profile name.

**Return type**`str`**`get_unique_id()`**

Compute unique hash string identifying a profile.

This is required to make distinction between multiple profiles, based on its content (banks relies on such unicity).

**Returns**

an hashed version of profile content

**Return type**`str`**`property group`**

Access the ‘group’ section.

**Returns**

the ‘group’ dict segment

**Return type**`dict`**`is_found()`**

Check if the current profile exists on disk.

**Returns**

True if the file exist on disk

**Return type**`bool`**`load_from_disk()`**

Load the profile from its representation on disk.

**Raises**

- `NotFoundError` – profile does not exist
- `NotFoundError` – profile path is not valid

**`load_template(name='default')`**

Populate the profile from templates of 5 basic config. blocks.

Filepath still need to be determined via `retrieve_file()` call.

**`property machine`**

Access the ‘machine’ section.

**Returns**

the ‘machine’ dict segment

**Return type**`dict`**`property runtime`**

Access the ‘runtime’ section.

**Returns**

the ‘runtime’ dict segment

**Return type**

dict

**property scope**

Return the profile scope.

**Returns**

profile scope

**Return type**

str

**split\_into\_configs**(*prefix*, *blocklist*, *scope=None*)

Convert the given profile into a list of basic blocks.

This is the reverse operation of creating a profile (not the ‘opposite’).

**Parameters**

- **prefix** (*str*) – common prefix name used to name basic blocks.
- **blocklist** (*list*) – list of config.blocks to generate (all 5 by default but can be retrained)
- **scope** (*str, optional*) – config block scope, defaults to None

**Raises**

*AlreadyExistError* – the created configuration block name already exist

**Returns**

list of created ConfigurationBlock

**Return type**

list

**pcvs.backend.profile.init()**

Initialization callback, loading available profiles on disk.

**pcvs.backend.profile.list\_profiles**(*scope=None*)

Return a list of valid profiles found on disk.

**Parameters**

**scope** (*str, optional*) – restriction on scope, defaults to None

**Returns**

dict of 3 dicts (‘user’, ‘local’ & ‘global’) or a single dict (if ‘scope’ was set), containing, for each profile name, the filepath.

**Return type**

dict

**pcvs.backend.profile.list\_templates()**

List available templates to be used for boostraping profiles.

**Returns**

a list of valid templates.

**Return type**

list

**pcvs.backend.report module****class pcvs.backend.report.Report**Bases: `object`

Map a Report interface, to handle request from frontends.

**add\_session(path)**

Insert new session to be managed.

**Parameters**`path (str)` – the build path (root dir)**Return type**`None`**classmethod dict\_convert\_list\_to\_cnt(l)**

Convert dict of arrays to a dict of array lengths.

Used to convert dict of per-status jobs to a summary of them.

**Parameters**`l (dict)` – the dict of arrays**Returns**

a summary of given dict

**Return type**`dict`**load\_alive\_sessions()**

Load currently active sessions as reference in PATH\_SESSION.

A issue with this function, as invalid sessions are not managet yet.

**Return type**`None`**property session\_ids: List[int]**

Get the list of session ids managed by this instance.

**Returns**

a list of session ids

**Return type**

list of integers

**session\_infos()**

Get sesion metadata for each session currently loaded into the instance.

**Returns**

the list of metadata (as dict)

**Return type**`list`**single\_session\_build\_path(sid)**

Get build prefix of a given session.

**Parameters**`sid (int)` – session ID

**Returns**

build path

**Return type**

str

**single\_session\_config(*sid*)****Return type**

dict

**single\_session\_get\_view(*sid*, *name*, *subset*=None, *summary*=False)**

Get a specific view from a given session.

A view consists in a per-status split of jobs depending on the purpose of the stored view. PCVS currently provide automatically: \* Per status \* Per tags \* Per labels

If *subset* is provided, only the nodes matching the key will be returned. If *summary* is True, a job count will be returned instead of actual job ids.**Parameters**

- **sid** (*int*) – Session ID
- **name** (*str*) – view name
- **subset** (*str*, *optional*) – only a selection of the view, defaults to None
- **summary** (*bool*, *optional*) – Should it be summarized, defaults to False

**Returns**

the result dict

**Return type**

dict

**single\_session\_job\_cnt(*sid*)**

Get per session number of job.

**Parameters****sid** (*int*) – the session ID**Returns**

The number of jobs (total)

**Return type**

int

**single\_session\_labels(*sid*)**

Get per-session available labels.

Outputs a per-status dict.

**Parameters****sid** (*int*) – Session ID**Returns**

dict of statuses

**Return type**

dict

**single\_session\_map\_id**(*sid, jid*)

For a given session id, convert a job it into its relative clas:*Test* object.

**Parameters**

- **sid** (*int*) – Session ID
- **jid** (*int*) – Job ID

**Returns**

the Actual test object

**Return type**

class:*Test*

**single\_session\_status**(*sid, filter=None*)

Get per-session status infos

**Parameters**

- **sid** (*int*) – Session id to extract info from.
- **filter** (*str, optional*) – optional status to filter in, defaults to None

**Returns**

A dict of statuses (or a single list if the filter is used)

**Return type**

dict or list

**single\_session\_tags**(*sid*)

Get per-session available tags.

Outputs a per-status dict.

**Parameters**

**sid** (*int*) – Session ID

**Returns**

dict of statuses

**Return type**

dict

**pcvs.backend.report.build\_static\_pages**(*buildir*)

From a given build directory, generate static pages.

This can be used only for already run test-suites (no real-time support) and when Flask cannot/don't want to be used.

**Parameters**

**buildir** (*str*) – the build directory to load

**Return type**

None

**pcvs.backend.report.start\_server**(*report*)

Initialize the Flask server, default to 5000.

A random port is picked if the default is already in use. :type report: *Report* :param report: The model to be used. :type report: class:*Report* :return: the application handler :rtype: class:*Flask*

`pcvs.backend.report.upload_buildir_results(buildir)`

Upload a whole test-suite from disk to the server data model.

**Parameters**

`buildir (str)` – the build directory

**Return type**

`None`

**pcvs.backend.run module**

`pcvs.backend.run.anonymize_archive()`

Erase from results any undesired output from the generated archive.

This process is disabled by default as it may increase significantly the validation process on large test bases. .. note:

It does **not** alter results **in-place**, only the generated archive. To preserve the anonymization, only the archive must be exported/shared, **not** the actual build directory.

`pcvs.backend.run.build_env_from_configuration(current_node, parent_prefix='pcvs')`

create a flat dict of variables mapping to the actual configuration.

In order to “pcvs.setup” to read current configuration, the whole config is serialized into shell variables. Purpose of this function is to flatten the configuration tree into env vars, each tree level being divided with an underscore.

This function is called recursively to walk through the whole tree.

**Example**

The `compiler.cc` config node become `$compiler_cc_program=<...>`

**Parameters**

- `current_node (dict)` – current node to flatten
- `parent_prefix (str, optional)` – prefix used to name vars at this depth, defaults to “pcvs”

**Returns**

a flat dict of the whole configuration, keys are shell variables.

**Return type**

`dict`

`pcvs.backend.run.display_summary(the_session)`

Display a summary for this run, based on profile & CLI arguments.

`pcvs.backend.run.dup_another_build(build_dir, outdir)`

Clone another build directory to start this validation upon it.

It allows to save test-generation time if the validation is re-run under the exact same terms (identical configuration & tests).

**Parameters**

- `build_dir (str)` – the build directory to copy resource from
- `outdir (str)` – where data will be copied to.

**Returns**

the whole configuration loaded from the dup'd build directory

**Return type**

`dict`

`pcvs.backend.run.find_files_to_process(path_dict)`

Lookup for test files to process, from the list of paths provided as parameter.

The given `path_dict` is a dict, where keys are path labels given by the user, while values are the actual path. This function then returns a two-list tuple, one being files needing preprocessing (setup), the other being static configuration files (pcvs.yml)

**Each list element is a tuple:**

- origin label
- subtree from this label leading to the actual file
- file basename (either “pcvs.setup” or “pcvs.yml”)

**Parameters**

`path_dict` (`dict`) – tree of paths to look for

**Returns**

a tuple with two lists

**Return type**

`tuple`

`pcvs.backend.run.prepare()`

Prepare the environment for a validation run.

This function prepares the build dir, create trees...

`pcvs.backend.run.print_progbar_walker(elt)`

Walker used to pretty-print progress bar element within Click.

**Parameters**

`elt` (`tuple`) – the element to pretty-print, containing the label & subprefix

**Returns**

the formatted string

**Return type**

`str`

`pcvs.backend.run.process_dyn_setup_scripts(setup_files)`

Process dynamic test files and generate associated tests.

This function executes pcvs.setup files after deploying the environment (to let these scripts access it). It leads to generate “pcvs.yml” files, then processed to construct tests.

**Parameters**

`setup_files` (`tuple`) – list of tuples, each mapping a single pcvs.setup file

**Returns**

list of errors encountered while processing.

**Return type**

`list`

`pcvs.backend.run.process_files()`

Process the test-suite generation.

It includes walking through user directories to find definitions AND generating the associated tests.

**Raises**

`TestUnfoldError` – An error occurred while processing files

`pcvs.backend.run.process_main_workflow(the_session=None)`

Main run.py entry point, triggering a PCVS validation run.

This function is called by session management and may be run within an active terminal or as a detached process.

**Parameters**

`the_session` (`Session`, optional) – the session handler this run is connected to, defaults to

`None`

`pcvs.backend.run.process_spack()`

`pcvs.backend.run.process_static_yaml_files(yaml_files)`

Process ‘pcvs.yml’ files to construct the test base.

**Parameters**

`yaml_files` (`list`) – list of tuples, each describing a single input file.

**Returns**

list of encountered errors while processing

**Return type**

`list`

`pcvs.backend.run.stop_pending_jobs(exc=None)`

`pcvs.backend.run.terminate()`

Finalize a validation run.

This includes generating & anonymizing (if needed) the archive.

**Raises**

`ProgramError` – Problem occurred while invoking the archive tool.

## pcvs.backend.session module

`class pcvs.backend.session.Session(date=None, path='.')`

Bases: `object`

Object representing a running validation (detached or not).

Despite the fact it is designed for managing concurrent runs, it takes a callback and can be derived for other needs.

**Parameters**

- `_func` (`Callable`) – user function to be called once the session starts
- `_sid` (`int`) – session id, automatically generated
- `_session_infos` (`dict`) – session infos dict

`class State(value)`

Bases: `IntEnum`

Enum of possible Session states.

**COMPLETED** = 2

**ERROR** = 3

**IN\_PROGRESS** = 1

**WAITING** = 0

**classmethod** **from\_yaml**(constructor, node)

Construct a *Session.State* from its YAML representation.

Relies on the fact the node contains a ‘Session.State’ tag. :param loader: the YAML loader :type loader: yaml.FullLoader :type node: :param node: the YAML representation :type node: Any :return: The session State as an object :rtype: *Session.State*

**classmethod** **to\_yaml**(representer, data)

Convert a Test.State to a valid YAML representation.

A new tag is created: ‘Session.State’ as a scalar (str). :param dumper: the YAML dumper object :type dumper: YAML().dumper :type data: :param data: the object to represent :type data: class:*Session.State* :return: the YAML representation :rtype: Any

### **property** **id**

Getter to session id.

#### **Returns**

session id

#### **Return type**

int

### **property** **infos**

Getter to session infos.

#### **Returns**

session infos

#### **Return type**

dict

### **load\_from**(sid, data)

Update the current object with session infos read from global file.

#### **Parameters**

- **sid** (int) – session id read from file
- **data** (dict) – session infos read from file

### **property**(kw)

Access specific data from the session stored info session.yml.

#### **Parameters**

**kw** (str) – the information to retrieve. kw must be a valid key

#### **Returns**

the requested session infos if exist

#### **Return type**

Any

**property rc**

Gett to final RC.

**Returns**

rc

**Return type**

int

**register\_callback(callback)**

Register the callback used as main function once the session is started.

**Parameters**

**callback** (*Callable*) – function to invoke

**run(\*args, \*\*kwargs)**

Run the session normally, without detaching the focus.

Arguments are user function ones. This function is also in charge of redirecting I/O properly (stdout, file, logs)

**Parameters**

**args** (*tuple*) – user function positional arguments

:param kwargs user function keyword-based arguments. :type kwargs: tuple

**run\_detached(\*args, \*\*kwargs)**

Run the session is detached mode.

Arguments are for user function only. :type args: :param args: user function positional arguments :type args: tuple :param kwargs user function keyword-based arguments. :type kwargs: tuple

**Returns**

the Session id created for this run.

**Return type**

int

**property state**

Getter to session status.

**Returns**

session status

**Return type**

int

**pcvs.backend.session.list\_alive\_sessions()**

Load and return the complete dict from session.yml file

**Returns**

the session dict

**Return type**

dict

**pcvs.backend.session.lock\_session\_file(timeout=None)**

Acquire the lockfil before manipulating the session.yml file.

This ensure safety between multiple PCVS instances. Be sure to call *unlock\_session\_file()* once completed

**Parameters**

**timeout** (*int*) – return from blocking once timeout is expired (raising *TimeoutError*)

---

`pcvs.backend.session.main_detached_session(sid, user_func, *args, **kwargs)`

Main function processed when running in detached mode.

This function is called by `Session.run_detached()` and is launched from cloned process (same global env, new main function).

#### Raises

`Exception` – any error occurring during the main process is re-raised.

#### Parameters

- `sid` – the session id
- `user_func` – the Python function used as the new `main()`
- `args` (`tuple`) – `user_func()` arguments
- `kwargs` (`dict`) – `user_func()` arguments

`pcvs.backend.session.remove_session_from_file(sid)`

clear a session from logs.

#### Parameters

`sid` (`int`) – the session id to remove.

`pcvs.backend.session.store_session_to_file(c)`

Save a new session into the session file (in HOME dir).

#### Parameters

`c` (`dict`) – session infos to store

#### Returns

the sid associated to new create session id.

#### Return type

`int`

`pcvs.backend.session.unlock_session_file()`

Release the lock after manipulating the `session.yml` file.

The call won't fail if the lockfile is not taken before unlocking.

`pcvs.backend.session.update_session_from_file(sid, update)`

Update data from a running session from the global file.

This only add/replace keys present in argument dict. Other keys remain.

#### Parameters

- `sid` (`int`) – the session id
- `update` – the keys to update. If already existing, content is replaced

#### Type

`dict`

## pcvs.backend.utilities module

```
class pcvs.backend.utilities.AutotoolsBuildSystem(root, dirs=None, files=None)
```

Bases: *BuildSystem*

Derived BuildSystem targeting Autotools projects.

**fill()**

Populate the dict relatively to the build system to build the proper YAML representation.

```
class pcvs.backend.utilities.BuildSystem(root, dirs=None, files=None)
```

Bases: *object*

Manage a generic build system discovery service.

### Variables

- **\_root** – the root directory the discovery service is attached to.
- **\_dirs** – list of directory found in \_root.
- **\_files** – list of files found in \_root
- **\_stream** – the resulted dict, representing targeted YAML architecture

**fill()**

This function should be overriden by overriden classes.

Nothing to do, by default.

```
generate_file(filename='pcvs.yml', force=False)
```

Build the YAML test file, based on path introspection and build model.

### Parameters

- **filename** (*str*) – test file suffix
- **force** (*bool*) – erase target file if exist.

```
class pcvs.backend.utilities.CMakeBuildSystem(root, dirs=None, files=None)
```

Bases: *BuildSystem*

Derived BuildSystem targeting CMake projects.

**fill()**

Populate the dict relatively to the build system to build the proper YAML representation.

```
class pcvs.backend.utilities.MakefileBuildSystem(root, dirs=None, files=None)
```

Bases: *BuildSystem*

Derived BuildSystem targeting Makefile-based projects.

**fill()**

Populate the dict relatively to the build system to build the proper YAML representation.

```
pcvs.backend.utilities.compute_scriptpath_from_testname(testname, output=None)
```

Locate the proper ‘list\_of\_tests.sh’ according to a fully-qualified test name.

### Parameters

- **testname** (*str*) – test name belonging to the script
- **output** (*str, optional*) – prefix to walk through, defaults to current directory

**Returns**

the associated path with testname

**Return type**

`str`

`pcvs.backend.utilities.get_logged_output(prefix, testname)`

`pcvs.backend.utilities.locate_scriptpaths(output=None)`

Path lookup to find all ‘list\_of\_tests’ script within a given prefix.

**Parameters**

`output (str, optional)` – prefix to walk through, defaults to current directory

**Returns**

the list of scripts found in prefix

**Return type**

`List[str]`

`pcvs.backend.utilities.process_check_configs(conversion=True)`

Analyse available configurations to ensure their correctness relatively to their respective schemes.

**Returns**

caught errors, as a dict, where the keys is the errmsg base64

**Return type**

`dict`

`pcvs.backend.utilities.process_check_directory(dir, pf_name='default', conversion=True)`

Analyze a directory to ensure defined test files are valid.

**Parameters**

`dir (str)` – the directory to process.

**Returns**

a dict of caught errors

**Return type**

`dict`

`pcvs.backend.utilities.process_check_profiles(conversion=True)`

Analyse available profiles and check their correctness relatively to the base scheme.

**Returns**

list of caught errors as a dict, where keys are error msg base64

**Return type**

`dict`

`pcvs.backend.utilities.process_check_setup_file(root, prefix, run_configuration)`

Check if a given pcvs.setup could be parsed if used in a regular process.

**Parameters**

- `filename (str)` – the pcvs.setup filepath
- `prefix (str)` – the subtree the setup is extract from (used as argument)

**Returns**

a tuple (err msg, icon to print, parsed data)

**Return type**

`tuple`

`pcvs.backend.utilities.process_discover_directory(path, override=False, force=False)`

Path discovery to detect & initialize build systems found.

#### Parameters

- `path` (`str`) – the root path to start with
- `override` (`bool`) – True if test files should be generated, default to False
- `force` (`bool`) – True if test files should be replaced if exist, default to False

### Module contents

## 19.1.2 `pcvs.cli` package

### Submodules

#### `pcvs.cli.cli_bank` module

`pcvs.cli.cli_bank.compl_bank_projects(ctx, args, incomplete)`

bank project completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

`pcvs.cli.cli_bank.compl_list_banks(ctx, args, incomplete)`

bank name completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

#### `pcvs.cli.cli_config` module

`pcvs.cli.cli_config.compl_list_templates(ctx, args, incomplete)`

Config template completion.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

#### Return type

`list`

`pcvs.cli.cli_config.compl_list_token(ctx, args, incomplete)`

config name completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

#### Return type

`list`

`pcvs.cli.cli_config.config_list_single_kind(kind, scope)`

Related to ‘config list’ command, handling a single ‘kind’ at a time.

#### Parameters

- `kind` (`str`) – config kind
- `scope` (`str`) – config scope

#### Return type

`None`

## pcvs.cli.cli\_profile module

`pcvs.cli.cli_profile.compl_list_templates(ctx, args, incomplete)`

the profile template completion.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

`pcvs.cli.cli_profile.compl_list_token(ctx, args, incomplete)`

profile name completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

`pcvs.cli.cli_profile.profile_interactive_select()`

Interactive selection of config blocks to build a profile.

Based on user input, this function displays, for each kind, possible blocks and waits for a selection. A final profile is built from them.

#### Returns

concatenation of basic blocks

#### Return type

`dict`

## `pcvs.cli.cli_report module`

### `pcvs.cli.cli_run module`

`pcvs.cli.cli_run.compl_list_dirs(ctx, args, incomplete)`

directory completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

#### Return type

`list`

`pcvs.cli.cli_run.handle_build_lockfile(exc=None)`

Remove the file lock in build dir if the application stops abruptly.

This function will automatically forward the raising exception to the next handler.

#### Raises

`Exception` – any exception triggering this handler

#### Parameters

`exc` (`Exception`) – The raising exception.

`pcvs.cli.cli_run.iterate_dirs(ctx, param, value)`

Validate directories provided by users & format them correctly.

Set the default label for a given path if not specified & Configure default directories if none was provided.

#### Parameters

- `ctx` (`Click.Context`) – Click Context
- `param` (`str`) – The arg targeting the function
- `value` (`List[str] or str`) – The value given by the user:

#### Returns

properly formatted dict of user directories, keys are labels.

#### Return type

`dict`

## `pcvs.cli.cli_session module`

`pcvs.cli.cli_session.compl_session_token(ctx, args, incomplete)`

Session name completion function.

#### Parameters

- `ctx` (`Click.Context`) – Click context
- `args` (`str`) – the option/argument requesting completion.
- `incomplete` (`str`) – the user input

#### Return type

`list`

**pcvs.cli.cli\_utilities module****Module contents****19.1.3 pcvs.testing package****Submodules****pcvs.testing.tedesc module**

**class** `pcvs.testing.tedesc.TEDescriptor(name, node, label, subprefix)`

Bases: `object`

A Test Descriptor (named TD, TE or TED), maps a test program representation, as defined by a root node in a single test files.

A TE Descriptor is not a test but a definition of a program (how to use it, to compile it...), leading to a collection once combined with a profile (providing on which MPI processes to run it, for instance).

**Variables**

- `_te_name` – YAML root node name, part of its unique id
- `_te_label` – which user directory this TE is coming from
- `_te_subtree` – subprefix, relative to label, where this TE is located
- `_full_name` – fully-qualified te-name
- `_srcdir` – absolute path pointing to the YAML testfile dirname
- `_buildir` – absolute path pointing to build equivalent of `_srcdir`
- `_skipped` – flag if this TE should be unfolded to tests or not
- `_effective_cnt` – number of tests created by this single TE
- `_program_criterion` – extra criterion defined by the TE
- `others` – used yaml node references.

**construct\_tests()**

Construct a collection of tests (build & run) from a given TE.

This function will process a YAML node and, through a generator, will create each test coming from it.

**get\_attr(name, dft=None)****get\_debug()**

Build information debug for the current TE.

**Returns**

the debug info

**Return type**

`dict`

**classmethod init\_system\_wide(base\_criterion\_name)**

Initialize system-wide information (to shorten accesses).

**Parameters**

`base_criterion_name` (`str`) – iterator name used as scheduling resource.

**property name**

Getter to the current TE name.

**Returns**

te\_name

**Return type**

str

`pcvs.testing.tedesc.build_job_deps(deps_node, pkg_label, pkg_prefix)`

Build the dependency list from a given dependency YAML node.

A depends\_on is used by test to establish their relationship. It looks like:

**Example**

**depends\_on:**

[“list\_of\_test\_name”]

**Parameters**

- **deps\_node** (`dict`) – the TE/job YAML node.
- **pkg\_label** (`str`) – the label where this TE is from (to compute depnames)
- **pkg\_prefix** – the subtree where this TE is from (to compute depnames)

:type pkg\_prefix, str or NoneType

**Returns**

a list of dependencies, either as depnames or PManager objects

**Return type**

list

`pcvs.testing.tedesc.build_pm_deps(deps_node)`

Build the dependency list from a given YAML node.

This only initialize package-manager oriented deps. For job deps, see build\_job\_deps

**Parameters**

**deps\_node** (`str`) – contains package\_manager YAML information

**Returns**

a list of PM objects, one for each entry

**Return type**

List[PManager]

`pcvs.testing.tedesc.detect_source_lang(array_of_files)`

Determine compilation language for a target file (or list of files).

Only one language is detected at once.

**Parameters**

**array\_of\_files** (`list`) – list of files to identify

**Returns**

the language code

**Return type**

str

**pcvs.testing.tedesc.extract\_compiler\_config(*lang*, *variants*)**

Build resource to compile based on language and variants involved.

**Parameters**

- ***lang*** (*str*) – target language
- ***variants*** (*list*) – list of enabled variants

**Returns**

the program, its args and env modifiers (in that order)

**Return type**

*tuple*

**pcvs.testing.test module****class pcvs.testing.Test(\*\*kwargs)**

Bases: *object*

Smallest component of a validation process.

A test is basically a shell command to run. Depending on its post-execution status, a success or a failure can be determined. To handle such component in a convenient way, more information can be attached to the command like a name, the elapsed time, the output, etc.

In order to make test content flexible, there is no fixed list of attributes. A Test() constructor is initialized via (\*args, \*\*kwargs), to populate a dict *\_array*.

**Variables**

- **Timeout\_RC** (*int*) – special constant given to jobs exceeding their time limit.
- **NOSTART\_STR** (*str*) – constant, setting default output when job cannot be run.

**MAXATTEMPTS\_STR = b'This test has failed to be scheduled too many times. Discarded.'**

**NOSTART\_STR = b'This test cannot be started.'**

**SCHED\_MAX\_ATTEMPTS = 50**

**class State(*value*)**

Bases: *IntEnum*

Provide Status management, specifically for tests/jobs.

Defined as an enum, it represents different states a job can take during its lifetime. As tests are then serialized into a JSON file, there is no need for construction/representation (as done for Session states).

**Variables**

- **WAITING** (*int*) – Job is currently waiting to be scheduled
- **IN\_PROGRESS** (*int*) – A running Set() handle the job, and is scheduled for run.
- **SUCCEED** (*int*) – Job successfully run and passes all checks (rc, matchers...)
- **FAILED** (*int*) – Job didn't succeed, at least one condition failed.
- **ERR\_DEP** (*int*) – Special cases to manage jobs descheduled because at least one of its dependencies have failed to complete.
- **ERR\_OTHER** (*int*) – Any other uncaught situation.

```
ERR_DEP = 4
ERR_OTHER = 5
EXECUTED = 6
FAILURE = 3
IN_PROGRESS = 1
SUCCESS = 2
WAITING = 0

Timeout_RC = 127

property basename: str
been_executed()
    Cehck if job has been executed (not waiting or in progress).

    Returns
        False if job is waiting for scheduling or in progress.

    Return type
        bool

property combination
    Getter to the test combination dict.

    Returns
        test comb dict.

    Return type
        dict

property command
    Getter for the full command.

    This is a real command, executed in a shell, coming from user's specificaiton. It should not be confused with invocation_command.

    Returns
        unescaped command line

    Return type
        str

classmethod compute_fq_name(label, subtree, name, combination=None, suffix=None)
    Generate the fully-qualified (dq) name for a test, based on : - the label & subtree (original FS tree) - the name (the TE name it is originated) - a potential extra suffix - the combination PCVS computed for this iteration.

display()
    Print the Test into stdout (through the manager).

property encoded_output: bytes

evaluate()
    TODO:
```

**executed(state=None)**

Set current Test as executed.

**Parameters**

- **state** – give a special state to the test, defaults to FAILED
- **state** – *Test.State*, optional

**extract\_metrics()**

TODO:

**first\_incomplete\_dep()**

Retrive the first ready-for-schedule dep.

This is mainly used to ease the scheduling process by following the job dependency graph.

**Returns**

a Test object if possible, None otherwise

**Return type**

*Test* or *NoneType*

**from\_json(test\_json)**

Replace the whole Test structure based on input JSON.

**Parameters**

**json** (*test-result-valid JSON-formated str*) – the json used to set this Test

**Return type**

*None*

**from\_minimal\_json(json)****generate\_script(srcfile)**

Serialize test logic to its Shell representation.

This script provides the shell sequence to put in a shell script switch-case, in order to reach that test from script arguments.

**Parameters**

**srcfile** (*str*) – script filepath, to store the actual wrapped command.

**Returns**

the shell-compliant instruction set to build the test

**Return type**

*str*

**get\_dep\_graph()****get\_dim(unit='n\_node')**

Return the orch-dimension value for this test.

The dimension can be defined by the user and let the orchestrator knows what resource are, and how to ‘count’ them’. This accessor allow the orchestrator to extract the information, based on the key name.

**Parameters**

**unit** (*str*) – the resource label, such label should exist within the test

**Returns**

The number of resource this Test is requesting.

**Return type**`int``classmethod get_jid_from_name(name)``get_raw_output(encoding='utf-8')`**Return type**`bytes``has_completed_deps()`

Check if the test can be scheduled.

It ensures it hasn't been executed yet (or currently running) and all its deps are resolved and successfully run.

**Returns**

True if the job can be scheduled

**Return type**`bool``has_failed_dep()`

Check if at least one dep is blocking this job from ever be scheduled.

**Returns**

True if at least one dep is shown a *Test.State.FAILURE* state.

**Return type**`bool``property invocation_command`

Getter for the list\_of\_test.sh invocation leading to run the job.

This command is under the form: `sh /path/list_of_tests.sh <test-name>`

**Returns**

wrapper command line

**Return type**`str``property jid: str`

Getter for unique Job ID within a run.

This attribute is generally set by the manager once job is uploaded to the dataset. :return: the job id :rtype: an positive integer of -1 if not set

`property job_depnames`

Getter to the list of deps, as an array of names.

This array is emptied when all deps are converted to objects.

**Returns**

the array of dep names

**Return type**`list`

**property job\_deps**

“Getter to the dependency list for this job.

The dependency struct is an array, where for each name (=key), the associated Job is stored (value) :return: the list of object-converted deps :rtype: list

**property label**

Getter to the test label.

**Returns**

the label

**Return type**

str

**property mod\_deps**

Getter to the list of pack-manager rules defined for this job.

There is no need for a \_depnames version as these deps are provided as PManger objects directly.

**Returns**

the list of package-manager based deps.

**Return type**

list

**property name**

Getter for fully-qualified job name.

**Returns**

test name.

**Return type**

str

**not\_picked()****property output: str****property output\_info: dict****pick()**

Flag the job as picked up for scheduling.

**pick\_count()**

```
res_scheme = <pcvs.helpers.system.ValidationScheme object>
```

**resolve\_a\_dep(name, obj)**

Resolve the dep object for a given dep name.

**Parameters**

- **name** (`str`) – the dep name to resolve, should be a valid dep.
- **obj** (`Test`) – the dep object, should be a Test()

**property retcode**

**save\_final\_result**(*rc=0, time=None, out=b'', state=None*)

Build the final Test result node.

**Parameters**

- **rc** (*int, optional*) – return code, defaults to 0
- **time** (*float, optional*) – elapsed time, defaults to 0.0
- **out** (*bytes, optional*) – standard out/err, defaults to b''
- **state** (*Test.State*, optional) – Job final status (if override needed), defaults to FAILED

**save\_raw\_run**(*out=None, rc=None, time=None*)

TODO:

**save\_status**(*state*)

**property state**

Getter for current job state.

**Returns**

the job current status.

**Return type**

*Test.State*

**property subtree**

Getter to the test subtree.

**Returns**

test subtree.

**Return type**

str.

**property tags**

Getter for the full list of tags.

**Returns**

the list of of tags

**Return type**

list

**property te\_name**

Getter to the test TE name.

**Returns**

test TE name.

**Return type**

str.

**property time**

TODO:

**property timeout**

Getter for Test timeout in seconds.

It cumulates timeout + tolerance, this value being passed to the subprocess.timeout.

**Returns**

an integer if a timeout is defined, None otherwise

**Return type**

`int` or `NoneType`

**`to_json(strstate=False)`**

Serialize the whole Test as a JSON object.

**Returns**

a JSON object mapping the test

**Return type**

`str`

**`to_minimal_json()`****pcvs.testing.testfile module****`class pcvs.testing.testfile.TestFile(file_in, path_out, data=None, label=None, prefix=None)`**

Bases: `object`

A TestFile manipulates source files to be processed as benchmarks (pcvs.yml & pcvs.setup).

It handles global informations about source imports & building one execution script (`list_of_tests.sh`) per input file.

**`param _in`**

YAML input file

**`type _in`**

`str`

**`param _path_out`**

prefix where to store output artifacts

**`type _path_out`**

`str`

**`param _raw`**

stream to populate the TestFile rather than opening input file

**`type _raw`**

`dict`

**`param _label`**

label the test file comes from

**`type _label`**

`str`

**`param _prefix`**

subtree the test file has been extracted

**`type _prefix`**

`str`

**`param _tests`**

list of tests handled by this file

**`type _tests`**

`list`

```
param _debug
    debug instructions (concatenation of TE debug infos)

type _debug
    dict

cc_pm_string = ''

flush_sh_file()
    Store the given input file into their destination.

generate_debug_info()
    Dump debug info to the appropriate file for the input object.

load_from_file(f)
load_from_str(data)
    Fill a File object from stream.

    This allows reusability (by loading only once).

    Parameters
        data (YAML-formatted str) – the YAML-formatted input stream.

property nb_descs

process()
    Load the YAML file and map YAML nodes to Test().

rt_pm_string = ''

save_yaml()
val_scheme = None

validate(allow_conversion=True)

    Return type
        bool

pcvs.testing.testfile.init_constant_tokens()
    Initialize global tokens to be replaced.

    The dict is built from profile specifications. The exact location for this function is still to be determined.

pcvs.testing.testfile.replace_special_token(content, src, build, prefix, list=False)
```

## Module contents

```
pcvs.testing.generate_local_variables(label, subprefix)
```

Return directories from PCVS working tree :

- the base source directory
- the current source directory
- the base build directory
- the current build directory

### Parameters

- **label** (`str`) – name of the object used to generate paths
- **subprefix** (`str`) – path to the subdirectories in the base path

**Raises**

`CommonException.NotFoundError` – the label is not recognized as to be validated

**Returns**

paths for PCVS working tree

**Return type**

`tuple`

## 19.1.4 `pcvs.converter` package

### Submodules

#### `pcvs.converter.yaml_converter` module

`pcvs.converter.yaml_converter.check_if_key_matches(key, value, ref_array)`

list all matches for the current key in the new YAML description.

**Return type**

`tuple`

`pcvs.converter.yaml_converter.compute_new_key(k, v, m)`

replace in ‘k’ any pattern found in ‘m’. ‘k’ is a string with placeholders, while ‘m’ is a match result with groups named after placeholders. This function will also expand the placeholder if ‘call:’ token is used to execute python code on the fly (complex transformation)

**Return type**

`str`

`pcvs.converter.yaml_converter.flatten(dd, prefix="")`

make the n-depth dict ‘dd’ a “flat” version, where the successive keys are chained in a tuple. for instance: {‘a’: {‘b’: {‘c’: value}}}} → {(‘a’, ‘b’, ‘c’): value}

**Return type**

`dict`

`pcvs.converter.yaml_converter.print_version(ctx, param, value)`

print converter version number, tied to PCVS version number

**Return type**

`None`

`pcvs.converter.yaml_converter.process(data, ref_array=None, warn_if_missing=True)`

Process YAML dict ‘data’ and return a transformed dict

**Return type**

`dict`

`pcvs.converter.yaml_converter.process_modifiers(data)`

applies rules in-place for the data dict. Rules are present in the desc\_dict[‘first’] sub-dict.

`pcvs.converter.yaml_converter.replace_placeholder(tmp, refs)`

The given TMP should be a dict, where keys contain placeholders, wrapped with “<>”. Each placeholder will be replaced (i.e. key will be changed) by the associated value in refs.

**Return type**

dict

`pcvs.converter.yaml_converter.separate_key_and_value(s, c)`

helper to split the key and value from a string

**Return type**

tuple

`pcvs.converter.yaml_converter.set_with(data, klist, val, append=False)`

Add a value to a n-depth dict where the depth is declared as a list of intermediate keys. the ‘append’ flag indicates if the given ‘value’ should be appended or replace the original content

## Module contents

### 19.1.5 `pcvs.helpers` package

#### Submodules

##### `pcvs.helpers.criterion module`

`class pcvs.helpers.criterion.Combination(crit_desc, dict_comb)`

Bases: object

A combination maps the actual concretization from multiple criterion.

For a given set of criterion, a Combination carries, for each kind, its associated value in order to generate the appropriate test

`get(k, dflt=None)`

Retrieve the actual value for a given combination element :type k: :param k: value to retrieve :type k: str :type dflt: :param dflt: default value if k is not a valid key :type: object

`items()`

Get the combination dict.

**Returns**

the whole combination dict.

**Return type**

dict

`translate_to_command()`

Translate the actual combination is tuple of three elements, based on the representation of each criterion in the test semantic. It builds tokens to provide to properly build the test command. It can either be:

1. an environment variable to export before the test to run (gathering system-scope and program-scope elements)
2. a runtime argument
3. a program-level argument (through custom-made iterators)

`translate_to_dict()`

Translate the combination into a dictionary.

**Returns**

configuration in the shape of a python dict

**Return type**`dict`**`translate_to_str()`**

Translate the actual combination in a pretty-format string. This is mainly used to generate actual test names

```
class pcvs.helpers.criterion.Criterion(name, description, local=False, numeric=False)
```

Bases: `object`

A Criterion is the representation of a component each program (i.e. test binary) should be run against. A criterion comes with a range of possible values, each leading to a different test

**`aliased_value(val)`**

Check if the given value has an alias for the current criterion. An alias is the value replacement to use instead of the one defined by test configuration. This allows to split test logic from runtime semantics.

For instance, TEs manipulate ‘ib’ as a value to depict the ‘infiniband’ network layer. But once the test has to be built, the term will change depending on the runtime carrying it, the value may be different from a runtime to another :type val: :param val: string with aliases to be replaced

**`concretize_value(val=“”)`**

Return the exact string mapping this criterion, according to the specification. (is it aliased ? should the option be put before/after the value?...) :type val: :param val: value to add with prefix :type val: str :return: values with aliases replaced :rtype: str

**`expand_values(reference=None)`**

Browse values for the current criterion and make it ready to generate combinations

**`property expanded`****`intersect(other)`**

Update the calling Criterion with the intersection of the current range of possible values with the one given as a parameters.

This is used to refine overriden per-TE criterion according to system-wide’s

**`is_discarded()`**

Should this criterion be ignored from the current TE generaiton ?

**`is_empty()`**

Is the current set of values empty May lead to errors, as it may indicates no common values has been found between user and system specifications

**`is_env()`**

Is this criterion targeting a component used as an env var ?

**`is_local()`**

Is the criterion local ? (program-scoped)

**`property max_value`****`property min_value`****`property name`**

Get the `name` attribute of this criterion.

**Returns**

name of this criterion

**Return type**

str

**property numeric**

Get the numeric attribute of this criterion.

**Returns**

numeric of this criterion

**Return type**

str

**override(desc)**

**Replace the value of the criterion using a descriptor containing the said value**

**Parameters**

**desc** (*dict*) – descriptor supposedly containing a ``value`` entry

**sanitize\_values()**

Check for any inconsistent values in the current Criterion.

Currently, only scalar items or dict (=> sequence) are allowed. Will raise an exception in case of inconsistency (Maybe this should be managed in another way through the error handling)

**property subtitle**

Get the subtitle attribute of this criterion.

**Returns**

subtitle of this criterion

**Return type**

str

**property values**

Get the value attribute of this criterion.

**Returns**

values of this criterion

**Return type**

list

**class pcvs.helpers.criterion.Serie(*dict\_of\_criterion*)**

Bases: *object*

A serie ties a test expression (TEDescriptor) to the possible values which can be taken for each criterion to build test sets. A serie can be seen as the Combination generator for a given TEDescriptor

**generate()**

Generator to build each combination

**classmethod register\_sys\_criterion(*system\_criterion*)**

copy/inherit the system-defined criterion (shortcut to global config)

**pcvs.helpers.criterion.initialize\_from\_system()**

Initialise system-wide criterions

TODO: Move this function elsewhere.

**pcvs.helpers.criterion.valid\_combination(dic)**

Check if dict is a valid criterion combination .

**Parameters**

**dic** (*dict*) – dict to check

**Returns**

True if dic is a valid combination

**Return type**

*bool*

**pcvs.helpers.exceptions module****class pcvs.helpers.exceptions.BankException**

Bases: *CommonException*

Bank-specific exceptions.

**exception ProjectNameError(reason='Unkown error', help\_msg='Please check pcvs --help for more information.', dbg\_info={})**

Bases: *GenericException*

name is not a valid project under the given bank.

**class pcvs.helpers.exceptions.CommonException**

Bases: *object*

Gathers exceptions commonly encountered by more specific namespaces.

**exception AlreadyExistError(reason='Already Exist', \*\*kwargs)**

Bases: *GenericException*

The content already exist as it should.

**exception BadTokenError(reason='Unkown error', help\_msg='Please check pcvs --help for more information.', dbg\_info={})**

Bases: *GenericException*

Badly formatted string, unable to parse.

**exception IOError(reason='Unkown error', help\_msg='Please check pcvs --help for more information.', dbg\_info={})**

Bases: *GenericException*

Communication error (FS, process) while processing data.

**exception NotFoundError(reason='Unkown error', help\_msg='Please check pcvs --help for more information.', dbg\_info={})**

Bases: *GenericException*

Content haven't been found based on specifications.

**exception NotImplementedError(reason='Unkown error', help\_msg='Please check pcvs --help for more information.', dbg\_info={})**

Bases: *GenericException*

Missing implementation for this particular feature.

```
exception NotPCVSRelated(reason='Unknown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})

Bases: GenericException

exception TimeoutError(reason='Unknown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})

Bases: GenericException

The parent class timeout error.

exception UnclassifiableError(reason='Unknown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})

Bases: GenericException

Unable to classify this common error.

exception WIPError(reason='Unknown error', help_msg='Please check pcvs --help for more information.',
dbg_info={})

Bases: GenericException

Work in Progress, not a real error.

class pcvs.helpers.exceptions.ConfigException

Bases: CommonException

Config-specific exceptions.

exception pcvs.helpers.exceptions.GenericException(reason='Unknown error', help_msg='Please check
pcvs --help for more information.', dbg_info={})

Bases: Exception

Generic error (custom errors will inherit of this).

add_dbg(**kwargs)

property dbg
    returns the extra infos of the exceptions (if any).

    Returns
        only the debug infos.

    Return type
        str

property dbg_str
    Stringify the debug infos. These infos are stored as a dict initially.

    return
        a itemized string.

    rtype
        str

property err
    returns the error part of the exceptions.

    Returns
        only the error part

    Return type
        str
```

**property help**

returns the help part of the exceptions.

**Returns**

only the help part

**Return type**

str

**class** pcvs.helpers.exceptions.**GitException**

Bases: *CommonException*

**exception** **BadEntryError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

**class** pcvs.helpers.exceptions.**LockException**

Bases: *CommonException*

Lock-specific exceptions.

**exception** **BadOwnerError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

Attempt to manipulate the lock while the current process is not the owner.

**exception** **TimeoutError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

Timeout reached before lock.

**class** pcvs.helpers.exceptions.**OrchestratorException**

Bases: *CommonException*

Execution-specific errors.

**exception** **CircularDependencyError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

Circular dep detected while processing job dep tree.

**exception** **UndefDependencyError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

Declared job dep cannot be fully qualified, not defined.

**class** pcvs.helpers.exceptions.**PluginException**

Bases: *CommonException*

Plugin-related exceptions.

**exception** **BadStepError**(*reason='Unkown error'*, *help\_msg='Please check pcvs --help for more information.'*, *dbg\_info={}*)

Bases: *GenericException*

targeted pass does not exist.

```
exception LoadError(reason='Issue(s) while loading plugin', **kwargs)
    Bases: GenericException
    Unable to load plugin directory.

class pcvs.helpers.exceptions.ProfileException
    Bases: CommonException
    Profile-specific exceptions.

exception IncompleteError(reason='Unkown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})
    Bases: GenericException
    A configuration block is missing to build the profile.

class pcvs.helpers.exceptions.PublisherException
    Bases: CommonException

exception AlreadyExistJobError(reason='Unkown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})
    Bases: GenericException
    A single ID leads to multiple jobs.

exception BadMagicTokenError(reason='Unkown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})
    Bases: GenericException
    Issue with token stored to file to check consistency

exception UnknownJobError(reason='Unkown error', help_msg='Please check pcvs --help for more
information.', dbg_info={})
    Bases: GenericException
    Unable to identify a job by its ID

class pcvs.helpers.exceptions.RunException
    Bases: CommonException
    Run-specific exceptions.

exception InProgressError(reason='Build directory currently used by another instance', **kwargs)
    Bases: GenericException
    A run is currently occurring in the given dir.

exception NonZeroSetupScript(reason='A setup script failed to complete', **kwargs)
    Bases: GenericException
    a setup script (=pcvs.setup) completed but returned non-zero exit code.

exception ProgramError(reason='A program cannot be found', **kwargs)
    Bases: GenericException
    The given program cannot be found.

class pcvs.helpers.exceptions.RunnerException
    Bases: CommonException
```

---

```
exception LaunchError(reason='Unknown error', help_msg='Please check pcvs --help for more information.', dbg_info={}())
    Bases: GenericException
    Unable to run a remote container

class pcvs.helpers.exceptions.TestException
    Bases: CommonException
    Test-specific exceptions.

exception TestExpressionError(reason='Issue(s) while parsing a Test Descriptor', **kwargs)
    Bases: GenericException
    Test description is wrongly formatted.

class pcvs.helpers.exceptions.ValidationError
    Bases: CommonException
    Validation-specific exceptions.

exception FormatError(reason='Invalid format', **kwargs)
    Bases: GenericException
    The content does not comply the required format (schemas).

exception SchemeError(reason='Invalid Scheme provided', **kwargs)
    Bases: GenericException
    The content is not a valid format (scheme).

exception WrongTokenError(reason='Invalid token(s) used as Placeholders', **kwargs)
    Bases: GenericException
    A unknown token is found in validated content
```

## pcvs.helpers.git module

```
class pcvs.helpers.git.Blob(repo, id, prefix='', data='')
    Bases: Tree
    Maps a Git ‘blob’ object, dedicated to hold data (“leaves” in Git trees)

class pcvs.helpers.git.Branch(repo, name='master')
    Bases: Reference
    Maps to a regular Git branch.

property name

class pcvs.helpers.git.Commit(repo, obj, metadata={})
    Bases: Reference
    Maps to a regular Git commit

get_info()
    Return commit metadata stored as a dict.

    It may contains extra infos compared to what a commit usually contains
```

`class pcvs.helpers.git.GitByAPI(prefix=None)`

Bases: `GitByGeneric`

Manage repository through a third-party Python module.

Currently, this work is based on pygit2.

**property branches**

Returns the list of available local branch names from this repo.

This is an abstract function as its behavior depends on derived classes.

**close()**

Unlock the repository.

`commit(tree, msg='No data', timestamp=None, parent=None, orphan=False)`

Create a commit from changes.

**Parameters**

- **tree** (`any`) – the changes tree to store as a commit
- **msg** (`str`) – the commit msg
- **parent** (`any`) – the parent commit
- **orphan** (`boolean`) – flag to create a dangling commit (=no-parent)

**Timestamp**

a commit date (current if not provided)

**Type**

`int`

`diff_tree(prefix=None, src_rev=None, dst_rev=None)`

Compare & return the list of patches

**gc()**

Run the garbage collector

`get_branch_from_str(name)`

`get_parents(ref)`

Retrieve parents for a given ref.

This method is not a part of a Reference object as the approach changes depending on the Git method used (lazy resolution).

**Parameters**

`ref` (`Reference`) – the revision

`get_tree(rev=None, prefix= '')`

Retrieve data associated with a given prefix. A tree can be used to set which ref should be used.

**Param[in] tree**

the ref from where get the data

**Param[in] prefix**

the unique prefix associated with data

**insert\_tree**(*prefix*, *data*, *root=None*)

Create a new tree mapping a prefix filled with ‘data’.

**Param[in] prefix**

the prefix under Git tree.

**Param[in] data**

the data to store.

**is\_open()**

Is the directory currently open ?

**iterate\_over**(*rev=None*)

starting from the ref, iterate references backwards (from newest to oldest).

**Parameters**

**ref** ([Reference](#)) – the starting point

**list\_commits**(*rev=None*, *since=None*, *until=None*)

List past commits finishing with ‘rev’.

The list can be shrunk with a start & end

**Parameters**

- **rev** – the revision to extract commit from
- **since** (*date*) – the oldest commit should be newer than this date
- **until** (*date*) – the newest commit should be older than this date

**Typ rev**

any

**list\_files**(*rev=None*, *prefix=''*)

For a given revision, list files (not only changed ones).

**Parameters**

**rev** ([Reference](#)) – the revision

**new\_branch**(*name*, *cid=None*)**open**(*bare=True*)

Open a new directory. Also lock to avoid races.

**revparse**(*ref*)

Convert a revision (tag, branch, commit) to a regular reference.

**Parameters**

**rev** ([Reference](#)) – Reference

**set\_branch**(*branch*, *commit*)**class** `pcvs.helpers.git.GitByCLI`(*prefix=''*)

Bases: [GitByGeneric](#)

Git endpoint ot manipulate a repository through basic CLI.

Currently relying on the `sh` module.

**property branches**

Returns the list of available local branch names from this repo.

This is an abstract function as its behavior depends on derived classes.

**close()**

Unlock the repository.

**commit(tree, msg='VOID', timestamp=None, parent=None, orphan=False)**

Create a commit from changes.

**Parameters**

- **tree** (any) – the changes tree to store as a commit
- **msg** (str) – the commit msg
- **parent** (any) – the parent commit
- **orphan** (boolean) – flag to create a dangling commit (=no-parent)

**Timestamp**

a commit date (current if not provided)

**Type**

int

**diff\_tree(prefix=None, src\_rev=None, dst\_rev=None)**

Compare & return the list of patches

**gc()**

Run the garbage collector

**get\_branch\_from\_str(name)**

**get\_parents(ref)**

Retrieve parents for a given ref.

This method is not a part of a Reference object as the approach changes depending on the Git method used (lazy resolution).

**Parameters**

**ref** (Reference) – the revision

**get\_tree(rev=None, prefix="")**

Retrieve data associated with a given prefix. A tree can be used to set which ref should be used.

**Param[in] tree**

the ref from where get the data

**Param[in] prefix**

the unique prefix associated with data

**insert\_tree(prefix, data, root=None)**

Create a new tree mapping a prefix filled with ‘data’.

**Param[in] prefix**

the prefix under Git tree.

**Param[in] data**

the data to store.

**is\_open()**

Is the directory currently open ?

**iterate\_over(*ref*)**

starting from the ref, iterate references backwards (from newest to oldest).

**Parameters**

**ref** ([Reference](#)) – the starting point

**list\_commits(*rev=None, since='', until=''*)**

List past commits finishing with ‘rev’.

The list can be shrunk with a start & end

**Parameters**

- **rev** – the revision to extract commit from
- **since** ([date](#)) – the oldest commit should be newer than this date
- **until** ([date](#)) – the newest commit should be older than this date

**Type rev**

any

**list\_files(*prefix*)**

For a given revision, list files (not only changed ones).

**Parameters**

**rev** ([Reference](#)) – the revision

**new\_branch(*name, cid=None*)****open(*bare=True*)**

Open a new directory. Also lock to avoid races.

**revparse(*rev*)**

Convert a revision (tag, branch, commit) to a regular reference.

**Parameters**

**rev** ([Reference](#)) – Reference

**set\_branch(*branch, commit*)****class pcv . helpers . git . GitByGeneric(*prefix=None, head='unknown/00000000'*)**

Bases: [ABC](#)

Create a Git endpoint able to discuss efficiently with repositories.

This base classe serves abstract methods to be implemented to create a new derived class. Currently are provided: - GitByAPI: relies on python module pygit2 (requires libgit2) - GitByCLI: based on regular Git program invocations (require git program)

**abstract property branches**

Returns the list of available local branche names from this repo.

This is an abstract function as its behavior depends on derived classes.

**abstract close()**

Unlock the repository.

**abstract commit**(*tree*, *msg*=‘No data’, *timestamp*=None, *parent*=None, *orphan*=False)

Create a commit from changes.

**Parameters**

- **tree** (any) – the changes tree to store as a commit
- **msg** (*str*) – the commit msg
- **parent** (any) – the parent commit
- **orphan** (boolean) – flag to create a dangling commit (=no-parent)

**Timestamp**

a commit date (current if not provided)

**Type**

*int*

**abstract diff\_tree**(*prefix*, *src\_rev*, *dst\_rev*)

Compare & return the list of patches

**abstract gc()**

Run the garbage collector

**get\_head()**

Get the current repo’s HEAD (used when no default)

**Returns**

a ref to the HEAD as a branch

**Return type**

*Branch*

**abstract get\_parents**(*ref*)

Retrieve parents for a given ref.

This method is not a part of a Reference object as the approach changes depending on the Git method used (lazy resolution).

**Parameters**

**ref** ([Reference](#)) – the revision

**abstract get\_tree**(*tree*, *prefix*)

Retrieve data associated with a given prefix. A tree can be used to set which ref should be used.

**Param[in] tree**

the ref from where get the data

**Param[in] prefix**

the unique prefix associated with data

**abstract insert\_tree**(*prefix*, *data*)

Create a new tree mapping a prefix filled with ‘data’.

**Param[in] prefix**

the prefix under Git tree.

**Param[in] data**

the data to store.

**abstract is\_open()**

Is the directory currently open ?

**abstract iterate\_over(*ref*)**

starting from the ref, iterate references backwards (from newest to oldest).

**Parameters**

**ref** ([Reference](#)) – the starting point

**abstract list\_commits(*rev, since, until*)**

List past commits finishing with ‘rev’.

The list can be shrunk with a start & end

**Parameters**

- **rev** – the revision to extract commit from
- **since** ([date](#)) – the oldest commit should be newer than this date
- **until** ([date](#)) – the newest commit should be older than this date

**Typ rev**

any

**abstract list\_files(*rev*)**

For a given revision, list files (not only changed ones).

**Parameters**

**rev** ([Reference](#)) – the revision

**abstract open()**

Open a new directory. Also lock to avoid races.

**abstract revparse(*rev*)**

Convert a revision (tag, branch, commit) to a regular reference.

**Parameters**

**rev** ([Reference](#)) – Reference

**set\_head(*new\_head*)**

Move the repo HEAD (used when no default ref is provided)

**set\_identity(*authname, authmail, commname, commmail*)**

Identities to be used if a commit is created.

**Parameters**

- **authname** ([str](#)) – author’s name
- **authmail** ([str](#)) – author’s email
- **commname** ([str](#)) – Committer’s name
- **commmail** ([str](#)) – Committer’s email

**set\_path(*prefix*)**

Associate a new directory to this bank.

**Parameters**

**prefix** ([str](#)) – the prefix locating the Git repo

**class pcvs.helpers.git.Reference(*repo*)**

Bases: [object](#)

Maps an object which can be “pointed” (as a Git semantic). It can usually be used to refer a commit, a simple hash or a branch.

**property repo**

Getter to the repo this reference comes from.

**class** pcvshelpers.git.**Tree**(repo, id, prefix='', children=[])

Bases: [Reference](#)

Maps to a git-lowlevel Tree object

**classmethod** **as\_root**(repo, hdl, children=[])

Create a Tree and attach it with the git-specific handler (if any)

**Parameters**

- **repo** (any) – the repo handle
- **hdl** (any) – the git-specific root handle
- **children** (any) – any prebuild children for this root node

**Returns**

the created Tree object

**Return type**

[Tree](#)

**pcvs.helpers.git.elect\_handler(prefix=None)**

Select the proper repository handler based on python support

Python 3.7+-based PCVS installations come with pygit2, thanks to provided wheels. Older versions are relying on regular Git commands (as wheels are not provided for Python3.6 and older & building pygit2 requires specific libgit2 version to be installed, hardening the installation process)

**pcvs.helpers.git.generate\_data\_hash(data)**

Hash data with git protocol.

**Parameters**

**data** ([str](#)) – data to hash

**Returns**

hashed data

**Return type**

[str](#)

**pcvs.helpers.git.get\_current\_usermail()**

Get the git user mail.

**Returns**

git user mail

**Return type**

[str](#)

**pcvs.helpers.git.get\_current\_username()**

Get the git username.

**Returns**

git username

**Return type**

[str](#)

**pcvs.helpers.git.request\_git\_attr(*k*)**

Get a git configuration.

**Parameters**

- ***k*** (*str*) – parameter to get

**Returns**

- a git configuration

**Return type**

- str*

**pcvs.helpers.log module****pcvs.helpers.log.init(*v=0, e=False, l=100, quiet=False*)**

initializes a global manager for everyone to use

**Parameters**

- ***v*** (*int*, *optional*) – verbosity level, defaults to 0
- ***e*** (*bool*, *optional*) – True to enable unicode alphabet, False to use ascii, defaults to False
- ***l*** (*int*, *optional*) – length of the terminal, defaults to 100
- ***quiet*** (*bool*, *optional*) – False to write to stdout, defaults to False

**pcvs.helpers.pm module****class pcvs.helpers.pm.ModuleManager(*spec*)**

Bases: *PManager*

handles Module package manager

**get(*load=True, install=False*)**

get the command to install the specified package

**Parameters**

- ***load*** (*bool*, *optional*) – load the specified package, defaults to True
- ***install*** (*bool*, *optional*) – install the specified package, defaults to False

**Returns**

command to install/load the package

**Return type**

- str*

**class pcvs.helpers.pm.PManager(*spec=None*)**

Bases: *object*

generic Package Manager

**get(*load, install*)**

Get specified packages for this manager

**Parameters**

- ***load*** (*bool*) – True to load the package

- **install** (*bool*) – True to install the package

**install()**

install specified packages

**class pcvs.helpers.pm.SpackManager(spec)**

Bases: *PManager*

handles Spack package manager

**get(*load=True*, *install=True*)**

get the commands to install the specified package

**Parameters**

- **load** (*bool*, *optional*) – load the specified package, defaults to True
- **install** (*bool*, *optional*) – install the specified package, defaults to True

**Returns**

command to install/load the package

**Return type**

*str*

**pcvs.helpers.pm.identify(*pm\_node*)**

identifies where

**Parameters**

**pm\_node** ([*type*]) – [description]

**Returns**

[description]

**Return type**

[*type*]

**pcvs.helpers.system module****class pcvs.helpers.system.Config(*d={}*, \**args*, \*\**kwargs*)**

Bases: *MetaDict*

a ‘Config’ is a dict extension (an *MetaDict*), used to manage all configuration fields. While it can contain arbitrary data, the whole PCVS configuration is composed of 5 distinct ‘categories’, each being a single Config. These are then gathered in a *MetaConfig* object (see below)

**from\_dict(*d*)**

Fill the current Config from a given dict :type d: :param d: dictionary to add :type d: dict

**from\_file(*filename*)**

Fill the current config from a given file

**Raises**

*CommonException.IOError* – file does not exist OR badly formatted

**set\_ifdef(*k*, *v*)**

shortcut function: init self[k] only if v is not None :type k: :param k: name of value to add :type k: str :type v: :param v: value to add :type v: str

---

**set\_nosquash(*k, v*)**  
 shortcut function: init self[k] only if *v* is not already set :type *k*: :param *k*: name of value to add :type *k*: str :type *v*: :param *v*: value to add :type *v*: str

**to\_dict()**  
 Convert the Config() to regular dict.

**validate(*kw*)**  
 Check if the Config instance matches the expected format as declared in schemes/. As the ‘category’ is not carried by the object itself, it is provided by the function argument.

**Parameters**  
**kw** (*str*) – keyword describing the configuration to be validated (scheme)

**class pcv.s.helpers.system.MetaConfig(\*args, \*\*kwargs)**  
 Bases: *MetaDict*

Root configuration object. It is composed of Config(), categorizing each configuration blocks. This MetaConfig() contains the whole profile along with any validation and current run information. This configuration is used as a dict extension.

To avoid carrying a global instanced object over the whole code, a class-scoped attribute allows to browse the global configuration from anywhere through *Metaconfig.root*”

**bootstrap\_compiler(*node*)**  
 “Specific initialize for compiler config block :type *node*: :param *node*: compiler block to initialize :type *node*: dict :return: added node :rtype: dict

**bootstrap\_criterion(*node*)**  
 “Specific initialize for criterion config block :type *node*: :param *node*: criterion block to initialize :type *node*: dict :return: initialized node :rtype: dict

**bootstrap\_from\_profile(*pf\_as\_dict*)**

**bootstrap\_generic(*subnode, node*)**  
 “Initialize a Config() object and store it under name ‘*node*’ :type *subnode*: :param *subnode*: node name :type *subnode*: str :type *node*: :param *node*: node to initialize and add :type *node*: dict :return: added *subnode* :rtype: dict

**bootstrap\_group(*node*)**  
 “Specific initialize for group config block. There is currently nothing to here but calling bootstrap\_generic() :type *node*: :param *node*: runtime block to initialize :type *node*: dict :return: added node :rtype: dict

**bootstrap\_machine(*node*)**  
 “Specific initialize for machine config block :type *node*: :param *node*: machine block to initialize :type *node*: dict :return: initialized node :rtype: dict

**bootstrap\_runtime(*node*)**  
 “Specific initialize for runtime config block :type *node*: :param *node*: runtime block to initialize :type *node*: dict :return: added node :rtype: dict

**bootstrap\_subnode(\**default\_subnode\_dict*)**

**bootstrap\_validation(*node*)**  
 “Specific initialize for validation config block :type *node*: :param *node*: validation block to initialize :type *node*: dict :return: initialized node :rtype: dict

**bootstrap\_validation\_from\_file(filepath)**

Specific initialize for validation config block. This function loads a file containing the validation dict.

**Parameters**

**filepath** (`os.path`, `str`) – path to file to be validated

**Raises**

`CommonException.IOError` – file is not found or badly formatted

**dump\_for\_export()**

Export the whole configuration as a dict. Prune any \_\_internal node beforehand.

**get\_internal(k)**

manipulate the internal MetaConfig() node to load not-exportable data :type k: :param k: value to get :type k: str

**root = None****set\_internal(k, v)**

manipulate the internal MetaConfig() node to store not-exportable data :type k: :param k: name of value to add :type k: str :type v: :param v: value to add :type v: str

**validation\_default\_file = '/home/docs/.pcvs/validation.yml'****class pcvs.helpers.system.MetaDict(\*args, \*\*kwargs)**

Bases: `Dict`

Helps with managing large configuration sets, based on dictionaries.

Once instanciated, an arbitrary subnode can be initialized like:

```
o = MetaDict()
o.field_a.subnode2 = 4
```

Currently, this class is just derived from `addict.Dict`. It is planned to remove this adherence.

**to\_dict()**

Convert the object to a regular dict.

**Returns**

a regular Python dict

**Return type**

`Dict`

**class pcvs.helpers.system.ValidationScheme(name)**

Bases: `object`

Object manipulating schemes (yaml) to enforce data formats. A `ValidationScheme` is instancied according to a ‘model’ (the format to validate). This instance can be used multiple times to check multiple streams belonging to the same model.

**avail\_list = None****classmethod available\_schemes()**

return list of currently supported formats to be validated. The list is extracted from `INSTALL/schemes/<model>-scheme.yml`

**validate(content, filepath=None)**

Validate a given datastructure (dict) agasint the loaded scheme.

**Parameters**

- **content** (*dict*) – json to validate
- **filepath** –

**Raises**

- *ValidationException.FormatError* – data are not valid
- *ValidationException.SchemeError* – issue while applying scheme

**pcvs.helpers.utils module**

**class** `pcvs.helpers.utils.Program(cmd=None)`

Bases: `object`

Simple class to encapsulate process management.

This is better and should be preferred as importing subprocess everywhere.

**property exception**

Getter, raised exception (for any reason)

**Returns**

an Exception-derived object

**Return type**

`Exception`

**property out**

Getter to actual execution output.

**Returns**

stderr/stdout combined

**Return type**

`str`

**property rc**

Getter, effective return code.

**Returns**

return code

**Return type**

`integer`

**run**(*input*='', *shell*=*False*, *timeout*=0)

Run the given program and capture I/Os

**Parameters**

- **input** (*str*) – raw data to be used as stdin
- **shell** (*boolean*) – is the provided command to be run within a shell
- **timeout** (*positive integer*) – allowed time before automatically killing the process

**Returns**

zero if the process started successfully, non-zero otherwise.

**Return type**

`integer`

`pcvs.helpers.utils.check_is_archive(f)`

`pcvs.helpers.utils.check_is_build_or_archive(x)`

`pcvs.helpers.utils.check_is_builddir(p)`

`pcvs.helpers.utils.check_valid_program(p, succ=None, fail=None, raise_if_fail=True)`

Check if p is a valid program, using the which function.

**Parameters**

- `p (str)` – program to check
- `succ (optional)` – function to call in case of success, defaults to None
- `fail (optional)` – function to call in case of failure, defaults to None
- `raise_if_fail (bool, optional)` – Raise an exception in case of failure, defaults to True

**Raises**

`RunException.ProgramError` – p is not a valid program

**Returns**

True if p is a program, False otherwise

**Return type**

`bool`

`pcvs.helpers.utils.check_valid_scope(s)`

Check if argument is a valid scope (local, user, global).

**Parameters**

`s (str)` – scope to check

**Raises**

`CommonException.BadTokenError` – the argument is not a valid scope

`pcvs.helpers.utils.copy_file(src, dest)`

Copy a source file into a destination directory.

**Parameters**

- `src (str)` – source file to copy.
- `dest (str)` – destination directory, may not exist yet.

`pcvs.helpers.utils.create_home_dir()`

Create a home directory

`pcvs.helpers.utils.create_or_clean_path(prefix, dir=False)`

Create a path or cleans it if it already exists.

**Parameters**

- `prefix (os.path, str)` – prefix of the path to create
- `dir (bool, optional)` – True if the path is a directory, defaults to False

`pcvs.helpers.utils.cwd(path)`

Change the working directory.

**Parameters**

`path (os.path, str)` – new working directory

---

`pcvs.helpers.utils.extract_infos_from_token(s, pair='right', single='right', maxsplit=3)`

Extract fields from tokens (a, b, c) from user's string.

**Parameters**

- `s (str)` – the input string
- `pair (str, optional)` – padding side when only 2 tokens found, defaults to “right”
- `single (str, optional)` – padding side when only 1 token found, defaults to “right”
- `maxsplit (int, optional)` – maximum split number for s, defaults to 3

**Returns**

3-string-tuple: mapping (scope, kind, name), any of them may be null

**Return type**

`tuple`

`pcvs.helpers.utils.find_buildir_from_prefix(path)`

Find the build directory from the path prefix.

**Parameters**

`path (os.path, str)` – path to search the build directory from

**Raises**

`CommonException.NotFoundError` – the build directory is not found

**Returns**

the path of the build directory

**Return type**

`os.path`

`pcvs.helpers.utils.get_lock_owner(f)`

The lock file will contain the process ID owning the lock. This function returns it.

**Parameters**

`f (str)` – the original file to mutex

**Returns**

the process ID

**Return type**

`int`

`pcvs.helpers.utils.get_lockfile_name(f)`

From a file to mutex, return the file lock name associated with it.

For instance for /a/b.yml, the lock file name will be /a/.b.yml.lck

**Parameters**

`f (str)` – the file to mutex

`pcvs.helpers.utils.is_locked(f)`

Is the given file locked somewhere else ?

**Parameters**

`f (str)` – the file to test

**Returns**

a boolean indicating whether the lock is held or not.

**Return type**

bool

`pcvs.helpers.utils.list_valid_archive_in_dir(p)`

`pcvs.helpers.utils.list_valid_builddirs_in_dir(p)`

`pcvs.helpers.utils.lock_file(f, reentrant=False, timeout=None, force=True)`

Try to lock a directory.

**Parameters**

- **f** (`os.path`) – name of lock
- **reentrant** (`bool`, *optional*) – True if this process may have locked this file before, defaults to False
- **timeout** (`int (seconds)`, *optional*) – time before timeout, defaults to None

**Raises**

`LockException.TimeoutError` – timeout is reached before the directory is locked

**Returns**

True if the file is reached, False otherwise

**Return type**

bool

`pcvs.helpers.utils.program_timeout(sig, frame)`

Timeout handler, called when a SIGALRM is received.

**Parameters**

- **sig** (`int`) – signal number
- **frame** – the callee (unused)

**Raises**

`CommonException.TimeoutError` – timeout is reached

`pcvs.helpers.utils.set_local_path(path)`

Set the prefix for the local storage.

**Parameters**

`path (os.path)` – path of the local storage

`pcvs.helpers.utils.start_autokill(timeout=None)`

Initialize a new time to automatically stop the current process once time is expired.

**Parameters**

`timeout (positive integer)` – value in seconds before the autokill will be raised

`pcvs.helpers.utils.storage_order()`

Return scopes in order of searching.

**Returns**

a list of scopes

**Return type**

list

`pcvs.helpers.utils.str_dict_as_envvar(d)`

Convert a dict to a list of shell-compliant variable strings.

The final result is a regular multiline str, each line being an entry.

**Parameters**

`d` (`dict`) – the dict containing env vars to serialize

**Returns**

the str, containing mulitple lines, each of them being a var.

**Return type**

`str`

`pcvs.helpers.utils.trylock_file(f, reentrant=False)`

Try to lock a file (used in lock\_file).

**Parameters**

- `f` (`os.path`) – name of lock
- `reentrant` (`bool`, `optional`) – True if this process may have locked this file before, defaults to False

**Returns**

True if the file is reached, False otherwise

**Return type**

`bool`

`pcvs.helpers.utils.unlock_file(f)`

Remove lock from a directory.

**Parameters**

`f` (`os.path`) – file locking the directory

**Module contents****19.1.6 pcvs.orchestration package****Submodules****pcvs.orchestration.publishers module**`class pcvs.orchestration.publishers.BuildDirectoryManager(build_dir='.)`

Bases: `object`

This class is intended to serve a build directory from a single entry point. Any module requiring to deal with resources from a run should be compliant with this interface. It provides basic mechanism to load/save any past, present or future executions.

`add_cache_entry(idx=0)``clean(*args)`

Prepare the build directory for a new execution by removing anything not relevant for a new run.

Please note this function will erase anything not relative to PCVS. As an argument, one may specify a specific prefix to be removed. Paths should relative to root build directory.

**Return type**

None

**clean\_archives()**

Prepare the build directory for a new execution by moving any previous archive to the backup directory named after NAME\_BUILD\_ARCHIVE\_DIR.

**Return type**

None

**property config: *MetaConfig***

Return the configuration associated with the current build directory

**Returns**

config struct

**Return type**

class:*MetaConfig*

**create\_archive(*timestamp=None*)**

Generate an archive for the build directory.

This archive will be stored in the root directory..

**Parameters**

**timestamp** (*Datetime, optional*) – file suffix, defaults to current timestamp

**Returns**

the archive path name

**Return type**

str

**finalize()**

Close & release the current instance.

It should not be used to save tests after this call.

**get\_cache\_entry(*idx=0*)**

**get\_config()**

Return the loaded configuration for the current build directory.

**Returns**

a dict representantion of yaml config

**Return type**

dict

**init\_results(*per\_file\_max\_sz=0*)**

Initialize the result handler.

This function is not called directly from the `__init__` method as this isntance may be used for both reading & writing into the destination directory. This function implies storing a new execution.

**Parameters**

**per\_file\_max\_sz** (*int, optional*) – max file size, defaults to unlimited

**load\_config()**

Load config stored onto disk & populate the current instance.

**Returns**

the loaded config

**Return type**class:*MetaConfig***classmethod** **load\_from\_archive**(*archive\_path*)

Populate the instance from an archive.

This object is initially built to load data from a build directory. This way, the object is mapped with an existing archive.

**Warning:** This method does not support (yet) to save tests after an archive has been loaded (as no output directory has been configured).

**Parameters****archive\_path** (*\_type\_*) – *\_description\_***Returns***\_description\_***Return type***\_type\_***property** **prefix**

Get the build directory prefix

**Returns**

the build path

**Return type**

str

**prepare**(*reuse=False*)

Prepare the dir for a new run.

This function is not included as part of the *\_\_init\_\_* function as this instance may be used both for reading & writing into the destination directory. This function implies all previous results be cleared off.**Parameters****reuse** (*bool*, optional) – keep previously generated YAML test-files, defaults to False**property** **results**

Getter to the result handler, for direct access

**Returns**

the result handler

**Return type**class:*ResultFileManager***save\_config**(*config*)

Save the config &amp; store it directly into the build directory.

**Parameters****config** (class:*MetaConfig*) – config**Return type**

None

**save\_extras**(*rel\_filename*, *data*='', *dir*=*False*, *export*=*False*)

Register a specific build-relative path, to be saved into the directory.

The only entry-point to save a resource into it. Resources can be files (with or without content) or directory.

If *export* is set to True, resource (file or whole directory) will also be copied to the final archive.

**Parameters**

- **rel\_filename** (*str*) – the filepath, relative to build dir.
- **data** (*Any, optional*) – data to be saved into the target file, defaults to ''
- **dir** (*bool, optional*) – is it a directory to save, defaults to False
- **export** (*bool, optional*) – should the target be also exported in final archive, defaults to False

**Return type**

*None*

**property scratch\_location**

Returns where third-party artifacts must be stored

**Returns**

the scratch directory

**Return type**

*str*

**property sid: Optional[int]**

Return the run ID as per configured with the current build directory.

If not found, this function may return None

**Returns**

the session ID

**Return type**

*int*

**use\_as\_global\_config()****class pcvs.orchestration.publishers.ResultFile**(*filepath*, *filename*)

Bases: *object*

A instance manages a pair of file dedicated to load/store PCVS job results on disk.

A job result is stored in two different files whens given to a single ResultFile: \* <prefix>.json, containing metadata (rc, command...) \* <prefix>.bz2 BZ2-compressed job data.

a MAGIC\_TOKEN is used to detect file/data corruption.

**MAGIC\_TOKEN** = 'PCVS-START-RAW-OUTPUT'

**close()**

Close the current instance (flush to disk)

**property content****property count**

Get the number of jobs in this handler.

---

**Returns**  
job count

**Return type**  
`int`

**extract\_output**(*offset*, *length*)

**Return type**  
`str`

**flush()**  
Sync cache with disk

**load()**  
Load job data from disk to populate the cache.

**property metadata\_prefix**  
Getter to the actual metadata file name

**Returns**  
filename

**Return type**  
`str`

**property prefix**  
Getter to the build prefix

**Returns**  
build prefix

**Return type**  
`str`

**property rawdata\_prefix**  
Getter to the actual rawdata file name

**Returns**  
file name

**Return type**  
`str`

**retrieve\_test**(*id=None*, *name=None*)  
Find jobs based on its id or name and return associated Test object.  
Only id OR name should be set (not both). To handle multiple matches, this function returns a list of class:*Test*.

**Parameters**

- **id** (`int`, optional) – job id, defaults to None
- **name** (`str`, optional) – test name (full), defaults to None

**Returns**  
A list of class:*Test*

**Return type**  
`list`

**save**(*id*, *data*, *output*)

Save a new job to this instance.

**Parameters**

- **id** (*int*) – job id
- **data** (*dict*) – metadata
- **output** (*bytes*) – raw output

**property size**

Current rawdata size

**Returns**

length of the rawdata file.

**Return type**

*int*

```
class pcvs.orchestration.publishers.ResultFileManager(prefix='.', per_file_max_ent=0,
                                                       per_file_max_sz=0)
```

Bases: *object*

Manages multiple class:*ResultFile*. Main purpose is to manage files to ensure files stored on disk remain consistent.

**browse\_tests()**

Iterate over every job stored into this build directory.

**Returns**

an iterable of Tests

**Return type**

List of tests

**Yield**

Test

**Return type**

Iterator[*Test*]

**build\_bidir\_map\_data()**

Rebuild global views from partial storage on disk.

For optimization reasons, information that may be rebuilt are not stored on disk to save space.

**Return type**

*None*

**create\_new\_result\_file()**

Initialize a new result file handler upon request.

**Return type**

*None*

**discover\_result\_files()**

Load existing results from prefix.

**Return type**

*None*

---

**file\_format = 'jobs-{}'**

**finalize()**

Flush & close the current manager.

This instance should not be used again after this call.

**flush()**

Ensure everything is in sync with persistent storage.

**Return type**

`None`

**increment = 0**

**map\_id(id)**

Commvert a job ID into its class:*Test* representation.

**Parameters**

`id (int)` – job id

**Returns**

the associated *Test* object or `None` if not found

**Return type**

class:*Test* or `None`

**property maps**

Returns available views from the current instance.

**Returns**

the maps

**Return type**

`dict`

**reconstruct\_map\_data()**

**Return type**

`None`

**reconstruct\_view\_data()**

**Return type**

`None`

**register\_view(name)**

Initialize a new view for this result manager.

**Parameters**

`name (str)` – the view name

**Return type**

`None`

**register\_view\_item(view, item)**

Initialize a single item within a view.

**Parameters**

- `view (str)` – the view name (created if not exist)
- `item (str)` – the item

**Return type**

None

**retrieve\_test(*id*)**

Build the Test object mapped to the given job id.

If such ID does not exist, it will return None.

**Parameters**

**id** (*\_type\_*) – \_description\_

**Returns**

\_description\_

**Return type**

List[*Test*]

**retrieve\_tests\_by\_name(*name*)**

Locate a test by its name.

As multiple matches could occur, this function return a list of class:*Test*

**Parameters**

**name** (*str*) – the test name

**Returns**

the actual list of test, empty if no one is found

**Return type**

list

**save(*job*)**

Add a new job to be saved to the result directory.

May not be flushed righ away to disk, some caching may be used to improve performance. While adding the Test to low-level manager, this function also updates view & maps accordingly.

**Parameters**

**job** (class:*Test*) – the job element to store

**property status\_view**

Returns the status view provided by PCVS.

**Returns**

a view

**Return type**

dict

**subtree\_view(*subtree*)**

Get a subset of the ‘tree’ view. Any LABEL/subtree combination is valid.

**Parameters**

**subtree** (*str*) – the prefix to look for

**Returns**

the dict mapping tests to the request

**Return type**

dict

**property tags\_view**

Get the tags view provided by PCVS.

**Returns**

a view

**Return type**

dict

**property total\_cnt**

Returns the total number of jobs from that directory (=run).

**Returns**

number of jobs

**Return type**

int

**property tree\_view**

Get the tree view, provided by default.

**Returns**

a view

**Return type**

dict

**property views**

Returns available views for the current instance.

**Returns**

the views

**Return type**

dict

## Module contents

**class pcvs.orchestration.Orchestrator**

Bases: `object`

The job orchestrator, managing test processing through the whole test base.

**Variables**

- `_conf` – global configuration object
- `_pending_sets` – started Sets not completed yet
- `_max_res` – number of resources allowed to be used
- `_publisher` – Result File Manager
- `_manager` – job manager
- `_maxconcurrent` – Max number of sets started at the same time.

**`add_new_job(job)`**

Append a new job to be scheduled.

**Parameters**

`job` (`Test`) – job to append

```
print_infos()  
    display pre-run infos.  
  
run(session)  
    Start the orchestrator.  
  
    Parameters  
        session (Session) – container owning the run.  
  
start_new_runner()  
    Start a new Runner thread & register comm queues.  
  
start_run(the_session=None, restart=False)  
    Start the orchestrator.  
  
    Parameters  
        • the_session (Session) – container owning the run.  
        • restart (False for a brand new run.) – whether the run is starting from scratch  
  
classmethod stop()  
    Request runner threads to stop.  
  
stop_runners()  
    Stop all previously started runners.  
    Wait for their completion.  
  
pcvs.orchestration.global_stop(e)
```

## 19.1.7 pcvs.webview package

### Module contents

```
pcvs.webview.create_app(iface)  
    Start and run the Flask application.  
  
    Returns  
        the application  
  
    Return type  
        Flask
```

## 19.2 Submodules

### 19.2.1 pcvs.main module

```
pcvs.main.print_version(ctx, param, value)  
    Print current version.  
  
This is used as an option formatter, PCVS is not even loaded yet.  
  
    Parameters  
        • ctx (Click.Context) – Click Context.
```

- **param** (*str*) – the option triggering the callback (unused here)
- **value** – the value provided with the option (unused here)

## 19.3 Module contents



## PYTHON MODULE INDEX

### p

pcvs, 119  
pcvs.backend, 72  
pcvs.backend.bank, 49  
pcvs.backend.config, 53  
pcvs.backend.profile, 57  
pcvs.backend.report, 61  
pcvs.backend.run, 64  
pcvs.backend.session, 66  
pcvs.backend.utilities, 70  
pcvs.cli, 75  
pcvs.cli.cli\_bank, 72  
pcvs.cli.cli\_config, 72  
pcvs.cli.cli\_profile, 73  
pcvs.cli.cli\_report, 74  
pcvs.cli.cli\_run, 74  
pcvs.cli.cli\_session, 74  
pcvs.cli.cli\_utilities, 75  
pcvs.converter, 86  
pcvs.converter.yaml\_converter, 85  
pcvs.helpers, 109  
pcvs.helpers.criterion, 86  
pcvs.helpers.exceptions, 89  
pcvs.helpers.git, 93  
pcvs.helpers.log, 101  
pcvs.helpers.pm, 101  
pcvs.helpers.system, 102  
pcvs.helpers.utils, 105  
pcvs.main, 118  
pcvs.orchestration, 117  
pcvs.orchestration.publishers, 109  
pcvs.testing, 84  
pcvs.testing.tedesc, 75  
pcvs.testing.test, 77  
pcvs.testing.testfile, 83  
pcvs.webview, 118



# INDEX

## A

add\_banklink() (*in module pcvs.backend.bank*), 52  
add\_cache\_entry() (*pcvs.orchestration.publishers.BuildDirectoryManager method*), 109  
add\_dbg() (*pcvs.helpers.exceptions.GenericException method*), 90  
add\_new\_job() (*pcvs.orchestration.Orchestrator method*), 117  
add\_session() (*pcvs.backend.report.Report method*), 61  
aliased\_value() (*pcvs.helpers.criterion.Criterion method*), 87  
anonymize\_archive() (*in module pcvs.backend.run*), 64  
as\_root() (*pcvs.helpers.git.Tree class method*), 100  
AutotoolsBuildSystem (*class in pcvs.backend.utilities*), 70  
avail\_list (*pcvs.helpers.system.ValidationScheme attribute*), 104  
available\_schemes() (*pcvs.helpers.system.ValidationScheme class method*), 104

## B

Bank (*class in pcvs.backend.bank*), 49  
BankException (*class in pcvs.helpers.exceptions*), 89  
BankException.ProjectNameError, 89  
BANKS (*in module pcvs.backend.bank*), 49  
basename (*pcvs.testing.test.Test property*), 78  
been\_executed() (*pcvs.testing.test.Test method*), 78  
Blob (*class in pcvs.helpers.git*), 93  
bootstrap\_compiler() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_criterion() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_from\_profile() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_generic() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_group() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_machine() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_runtime() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_subnode() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_validation() (*pcvs.helpers.system.MetaConfig method*), 103  
bootstrap\_validation\_from\_file() (*pcvs.helpers.system.MetaConfig method*), 103  
Branch (*class in pcvs.helpers.git*), 93  
branches (*pcvs.helpers.git.GitByAPI property*), 94  
branches (*pcvs.helpers.git.GitByCLI property*), 95  
branches (*pcvs.helpers.git.GitByGeneric property*), 97  
browse\_tests() (*pcvs.orchestration.publishers.ResultFileManager method*), 114  
build\_bidir\_map\_data() (*pcvs.orchestration.publishers.ResultFileManager method*), 114  
build\_env\_from\_configuration() (*in module pcvs.backend.run*), 64  
build\_job\_deps() (*in module pcvs.testing.tedesc*), 76  
build\_pm\_deps() (*in module pcvs.testing.tedesc*), 76  
build\_static\_pages() (*in module pcvs.backend.report*), 63  
build\_target\_branch\_name() (*pcvs.backend.bank.Bank method*), 49  
BuildDirectoryManager (*class in pcvs.orchestration.publishers*), 109  
BuildSystem (*class in pcvs.backend.utilities*), 70

## C

cc\_pm\_string (*pcvs.testing.testfile.TestFile attribute*), 84

check() (pcvs.backend.config.ConfigurationBlock method), 53  
check() (pcvs.backend.profile.Profile method), 57  
check\_if\_key\_matches() (in module pcvs.converter.yaml\_converter), 85  
check\_is\_archive() (in module pcvs.helpers.utils), 105  
check\_is\_build\_or\_archive() (in module pcvs.helpers.utils), 106  
check\_is\_builddir() (in module pcvs.helpers.utils), 106  
check\_valid\_kind() (in module pcvs.backend.config), 56  
check\_valid\_program() (in module pcvs.helpers.utils), 106  
check\_valid\_scope() (in module pcvs.helpers.utils), 106  
clean() (pcvs.orchestration.publishers.BuildDirectoryManager method), 109  
clean\_archives() (pcvs.orchestration.publishers.BuildDirectoryManager method), 110  
clone() (pcvs.backend.config.ConfigurationBlock method), 53  
clone() (pcvs.backend.profile.Profile method), 57  
close() (pcvs.helpers.git.GitByAPI method), 94  
close() (pcvs.helpers.git.GitByCLI method), 96  
close() (pcvs.helpers.git.GitByGeneric method), 97  
close() (pcvs.orchestration.publishers.ResultFile method), 112  
CMakeBuildSystem (class in pcvs.backend.utilities), 70  
Combination (class in pcvs.helpers.criterion), 86  
combination (pcvs.testing.test.Test property), 78  
command (pcvs.testing.test.Test property), 78  
Commit (class in pcvs.helpers.git), 93  
commit() (pcvs.helpers.git.GitByAPI method), 94  
commit() (pcvs.helpers.git.GitByCLI method), 96  
commit() (pcvs.helpers.git.GitByGeneric method), 97  
CommonException (class in pcvs.helpers.exceptions), 89  
CommonException.AlreadyExistError, 89  
CommonException.BadTokenError, 89  
CommonException.IOError, 89  
CommonException.NotFoundError, 89  
CommonException.NotImplementedError, 89  
CommonException.NotPCVSRelated, 89  
CommonException.TimeoutError, 90  
CommonException.UnclassifiableError, 90  
CommonException.WIPError, 90  
compiler (pcvs.backend.profile.Profile property), 57  
compl\_bank\_projects() (in module pcvs.cli.cli\_bank), 72  
compl\_list\_banks() (in module pcvs.cli.cli\_bank), 72  
compl\_list\_dirs() (in module pcvs.cli.cli\_run), 74  
compl\_list\_templates() (in module pcvs.cli.cli\_config), 72  
compl\_list\_templates() (in module pcvs.cli.cli\_profile), 73  
compl\_list\_token() (in module pcvs.cli.cli\_config), 72  
compl\_list\_token() (in module pcvs.cli.cli\_profile), 73  
compl\_session\_token() (in module pcvs.cli.cli\_session), 74  
COMPLETED (pcvs.backend.session.Session.State attribute), 66  
compute\_fq\_name() (pcvs.testing.test.Test class method), 78  
compute\_new\_key() (in module pcvs.converter.yaml\_converter), 85  
compute\_scriptpath\_from\_testname() (in module pcvs.backend.utilities), 70  
concretize\_value() (pcvs.helpers.criterion.Criterion method), 87  
config (class in pcvs.helpers.system), 102  
config (pcvs.orchestration.publishers.BuildDirectoryManager config property), 110  
config\_list\_single\_kind() (in module pcvs.cli.cli\_config), 73  
ConfigException (class in pcvs.helpers.exceptions), 90  
ConfigurationBlock (class in pcvs.backend.config), 53  
construct\_tests() (pcvs.testing.tedesc.TEDescriptor method), 75  
content (pcvs.orchestration.publishers.ResultFile property), 112  
copy\_file() (in module pcvs.helpers.utils), 106  
count (pcvs.orchestration.publishers.ResultFile property), 112  
create\_app() (in module pcvs.webview), 118  
create\_archive() (pcvs.orchestration.publishers.BuildDirectoryManager method), 110  
create\_home\_dir() (in module pcvs.helpers.utils), 106  
create\_new\_result\_file() (pcvs.orchestration.publishers.ResultFileManager method), 114  
create\_or\_clean\_path() (in module pcvs.helpers.utils), 106  
Criterion (class in pcvs.helpers.criterion), 87  
criterion (pcvs.backend.profile.Profile property), 57  
cwd() (in module pcvs.helpers.utils), 106

## D

dbg (pcvs.helpers.exceptions.GenericException property), 90  
dbg\_str (pcvs.helpers.exceptions.GenericException property), 90  
default\_project (pcvs.backend.bank.Bank property), 50  
delete() (pcvs.backend.config.ConfigurationBlock method), 53  
delete() (pcvs.backend.profile.Profile method), 57

`detect_source_lang()` (in module `pcvs.testing.tedesc`), 76  
`dict_convert_list_to_cnt()` (pcvs.backend.report.Report class method), 61  
`diff_tree()` (pcvs.helpers.git.GitByAPI method), 94  
`diff_tree()` (pcvs.helpers.git.GitByCLI method), 96  
`diff_tree()` (pcvs.helpers.git.GitByGeneric method), 98  
`discover_result_files()` (pcvs.orchestration.publishers.ResultFileManager method), 114  
`display()` (pcvs.backend.config.ConfigurationBlock method), 54  
`display()` (pcvs.backend.profile.Profile method), 57  
`display()` (pcvs.testing.test.Test method), 78  
`display_summary()` (in module pcvs.backend.run), 64  
`dump()` (pcvs.backend.config.ConfigurationBlock method), 54  
`dump()` (pcvs.backend.profile.Profile method), 58  
`dump_for_export()` (pcvs.helpers.system.MetaConfig method), 104  
`dup_another_build()` (in module pcvs.backend.run), 64

## E

`edit()` (pcvs.backend.config.ConfigurationBlock method), 54  
`edit()` (pcvs.backend.profile.Profile method), 58  
`edit_plugin()` (pcvs.backend.config.ConfigurationBlock method), 54  
`edit_plugin()` (pcvs.backend.profile.Profile method), 58  
`elect_handler()` (in module pcvs.helpers.git), 100  
`encoded_output` (pcvs.testing.test.Test property), 78  
`err` (pcvs.helpers.exceptions.GenericException property), 90  
`ERR_DEP` (pcvs.testing.test.Test.State attribute), 77  
`ERR_OTHER` (pcvs.testing.test.Test.State attribute), 78  
`ERROR` (pcvs.backend.session.Session.State attribute), 67  
`evaluate()` (pcvs.testing.test.Test method), 78  
`exception` (pcvs.helpers.utils.Program property), 105  
`EXECUTED` (pcvs.testing.test.Test.State attribute), 78  
`executed()` (pcvs.testing.test.Test method), 78  
`exists()` (pcvs.backend.bank.Bank method), 50  
`expand_values()` (pcvs.helpers.criterion.Criterion method), 87  
`expanded` (pcvs.helpers.criterion.Criterion property), 87  
`extract_compiler_config()` (in module pcvs.testing.tedesc), 76  
`extract_infos_from_token()` (in module pcvs.helpers.utils), 106  
`extract_metrics()` (pcvs.testing.test.Test method), 79  
`extract_output()` (pcvs.orchestration.publishers.ResultFile method), 113

## F

`FAILURE` (pcvs.testing.test.Test.State attribute), 78  
`file_format` (pcvs.orchestration.publishers.ResultFileManager attribute), 114  
`fill()` (pcvs.backend.config.ConfigurationBlock method), 54  
`fill()` (pcvs.backend.profile.Profile method), 58  
`fill()` (pcvs.backend.utilities.AutotoolsBuildSystem method), 70  
`fill()` (pcvs.backend.utilities.BuildSystem method), 70  
`fill()` (pcvs.backend.utilities.CMakeBuildSystem method), 70  
`fill()` (pcvs.backend.utilities.MakefileBuildSystem method), 70  
`finalize()` (pcvs.orchestration.publishers.BuildDirectoryManager method), 110  
`finalize()` (pcvs.orchestration.publishers.ResultFileManager method), 115  
`find_buildir_from_prefix()` (in module pcvs.helpers.utils), 107  
`find_files_to_process()` (in module pcvs.backend.run), 65  
`first_incomplete_dep()` (pcvs.testing.test.Test method), 79  
`flatten()` (in module pcvs.converter.yaml\_converter), 85  
`flush()` (pcvs.orchestration.publishers.ResultFileManager method), 113  
`flush()` (pcvs.orchestration.publishers.ResultFileManager method), 115  
`flush_sh_file()` (pcvs.testing.testfile.TestFile method), 84  
`flush_to_disk()` (in module pcvs.backend.bank), 52  
`flush_to_disk()` (pcvs.backend.config.ConfigurationBlock method), 54  
`flush_to_disk()` (pcvs.backend.profile.Profile method), 58  
`from_dict()` (pcvs.helpers.system.Config method), 102  
`from_file()` (pcvs.helpers.system.Config method), 102  
`from_json()` (pcvs.testing.test.Test method), 79  
`from_minimal_json()` (pcvs.testing.test.Test method), 79  
`from_yaml()` (pcvs.backend.session.Session.State class method), 67  
`full_name` (pcvs.backend.config.ConfigurationBlock property), 54  
`full_name` (pcvs.backend.profile.Profile property), 58

## G

`gc()` (pcvs.helpers.git.GitByAPI method), 94  
`gc()` (pcvs.helpers.git.GitByCLI method), 96  
`gc()` (pcvs.helpers.git.GitByGeneric method), 98  
`generate()` (pcvs.helpers.criterion.Serie method), 88

generate\_data\_hash() (*in module* `pcvs.helpers.git`), 100  
generate\_debug\_info() (`pcvs.testing.testfile.TestFile method`), 84  
generate\_file() (`pcvs.backend.utilities.BuildSystem method`), 70  
generate\_local\_variables() (*in module* `pcvs.testing`), 84  
generate\_script() (`pcvs.testing.test.Test method`), 79  
GenericException, 90  
get() (`pcvs.helpers.criterion.Combination method`), 86  
get() (`pcvs.helpers.pm.ModuleManager method`), 101  
get() (`pcvs.helpers.pm.PManager method`), 101  
get() (`pcvs.helpers.pm.SpackManager method`), 102  
get\_attr() (`pcvs.testing.tedesc.TEDescriptor method`), 75  
get\_branch\_from\_str() (`pcvs.helpers.git.GitByAPI method`), 94  
get\_branch\_from\_str() (`pcvs.helpers.git.GitByCLI method`), 96  
get\_cache\_entry() (`pcvs.orchestration.publishers.BuildDirectoryManager method`), 110  
get\_config() (`pcvs.orchestration.publishers.BuildDirectoryManager method`), 110  
get\_count() (`pcvs.backend.bank.Bank method`), 50  
get\_current\_usermail() (*in module* `pcvs.helpers.git`), 100  
get\_current\_username() (*in module* `pcvs.helpers.git`), 100  
get\_debug() (`pcvs.testing.tedesc.TEDescriptor method`), 75  
get\_dep\_graph() (`pcvs.testing.test.Test method`), 79  
get\_dim() (`pcvs.testing.test.Test method`), 79  
get\_head() (`pcvs.helpers.git.GitByGeneric method`), 98  
get\_info() (`pcvs.helpers.git.Commit method`), 93  
get\_internal() (`pcvs.helpers.system.MetaConfig method`), 104  
get\_jid\_from\_name() (`pcvs.testing.test.Test class method`), 80  
get\_lock\_owner() (*in module* `pcvs.helpers.utils`), 107  
get\_lockfile\_name() (*in module* `pcvs.helpers.utils`), 107  
get\_logged\_output() (*in module* `pcvs.backend.utilities`), 71  
get\_parents() (`pcvs.helpers.git.GitByAPI method`), 94  
get\_parents() (`pcvs.helpers.git.GitByCLI method`), 96  
get\_parents() (`pcvs.helpers.git.GitByGeneric method`), 98  
get\_raw\_output() (`pcvs.testing.test.Test method`), 80  
get\_tree() (`pcvs.helpers.git.GitByAPI method`), 94  
get\_tree() (`pcvs.helpers.git.GitByCLI method`), 96  
get\_tree() (`pcvs.helpers.git.GitByGeneric method`), 98  
get\_unique\_id() (`pcvs.backend.profile.Profile method`), 59  
GitByAPI (*class in* `pcvs.helpers.git`), 93  
GitByCLI (*class in* `pcvs.helpers.git`), 95  
GitByGeneric (*class in* `pcvs.helpers.git`), 97  
GitException (*class in* `pcvs.helpers.exceptions`), 91  
GitException.BadEntryError, 91  
global\_stop() (*in module* `pcvs.orchestration`), 118  
group (`pcvs.backend.profile.Profile property`), 59

## H

handle\_build\_lockfile() (*in module* `pcvs.cli.cli_run`), 74  
has\_completed\_deps() (`pcvs.testing.test.Test method`), 80  
has\_failed\_dep() (`pcvs.testing.test.Test method`), 80  
help (`pcvs.helpers.exceptions.GenericException property`), 90

## I

id (`pcvs.backend.session.Session property`), 67  
identify() (*in module* `pcvs.helpers.pm`), 102  
`IN_PROGRESS` (`pcvs.backend.session.Session.State attribute`), 67  
`IN_PROGRESS` (`pcvs.testing.test.Test.State attribute`), 78  
increment (`pcvs.orchestration.publishers.ResultFileManager attribute`), 115  
infos (`pcvs.backend.session.Session property`), 67  
init() (*in module* `pcvs.backend.bank`), 52  
init() (*in module* `pcvs.backend.config`), 56  
init() (*in module* `pcvs.backend.profile`), 60  
init() (*in module* `pcvs.helpers.log`), 101  
init\_constant\_tokens() (*in module* `pcvs.testing.testfile`), 84  
init\_results() (`pcvs.orchestration.publishers.BuildDirectoryManager method`), 110  
init\_system\_wide() (`pcvs.testing.tedesc.TEDescriptor class method`), 75  
initialize\_from\_system() (*in module* `pcvs.helpers.criterion`), 88  
insert\_tree() (`pcvs.helpers.git.GitByAPI method`), 94  
insert\_tree() (`pcvs.helpers.git.GitByCLI method`), 96  
insert\_tree() (`pcvs.helpers.git.GitByGeneric method`), 98  
install() (`pcvs.helpers.pm.PManager method`), 102  
intersect() (`pcvs.helpers.criterion.Criterion method`), 87  
invocation\_command (`pcvs.testing.test.Test property`), 80  
is\_discarded() (`pcvs.helpers.criterion.Criterion method`), 87  
is\_empty() (`pcvs.helpers.criterion.Criterion method`), 87  
is\_env() (`pcvs.helpers.criterion.Criterion method`), 87  
is\_found() (`pcvs.backend.config.ConfigurationBlock method`), 55

`is_found()` (*pcvs.backend.profile.Profile method*), 59  
`is_local()` (*pcvs.helpers.criterion.Criterion method*), 87  
`is_locked()` (*in module pcvs.helpers.utils*), 107  
`is_open()` (*pcvs.helpers.git.GitByAPI method*), 95  
`is_open()` (*pcvs.helpers.git.GitByCLI method*), 96  
`is_open()` (*pcvs.helpers.git.GitByGeneric method*), 98  
`items()` (*pcvs.helpers.criterion.Combination method*), 86  
`iterate_dirs()` (*in module pcvs.cli.cli\_run*), 74  
`iterate_over()` (*pcvs.helpers.git.GitByAPI method*), 95  
`iterate_over()` (*pcvs.helpers.git.GitByCLI method*), 97  
`iterate_over()` (*pcvs.helpers.git.GitByGeneric method*), 98

**J**

`jid` (*pcvs.testing.test.Test property*), 80  
`job_depnames` (*pcvs.testing.test.Test property*), 80  
`job_deps` (*pcvs.testing.test.Test property*), 80

**L**

`label` (*pcvs.testing.test.Test property*), 81  
`list_alive_sessions()` (*in module pcvs.backend.session*), 68  
`list_banks()` (*in module pcvs.backend.bank*), 52  
`list_blocks()` (*in module pcvs.backend.config*), 56  
`list_commits()` (*pcvs.helpers.git.GitByAPI method*), 95  
`list_commits()` (*pcvs.helpers.git.GitByCLI method*), 97  
`list_commits()` (*pcvs.helpers.git.GitByGeneric method*), 99  
`list_files()` (*pcvs.helpers.git.GitByAPI method*), 95  
`list_files()` (*pcvs.helpers.git.GitByCLI method*), 97  
`list_files()` (*pcvs.helpers.git.GitByGeneric method*), 99  
`list_profiles()` (*in module pcvs.backend.profile*), 60  
`list_templates()` (*in module pcvs.backend.config*), 56  
`list_templates()` (*in module pcvs.backend.profile*), 60  
`list_valid_archive_in_dir()` (*in module pcvs.helpers.utils*), 108  
`list_valid_buildirs_in_dir()` (*in module pcvs.helpers.utils*), 108  
`load()` (*pcvs.orchestration.publishers.ResultFile method*), 113  
`load_alive_sessions()` (*pcvs.backend.report.Report method*), 61  
`load_config()` (*pcvs.orchestration.publishers.BuildDirectoryManager method*), 110  
`load_from()` (*pcvs.backend.session.Session method*), 67  
`load_from_archive()` (*pcvs.orchestration.publishers.BuildDirectoryManager* class method), 111  
`load_from_disk()` (*pcvs.backend.config.ConfigurationBlock method*), 55  
`load_from_disk()` (*pcvs.backend.profile.Profile method*), 59  
`load_from_file()` (*pcvs.testing.testfile.TestFile method*), 84  
`load_from_str()` (*pcvs.testing.testfile.TestFile method*), 84  
`load_template()` (*pcvs.backend.config.ConfigurationBlock method*), 55  
`load_template()` (*pcvs.backend.profile.Profile method*), 59  
`locate_scriptpaths()` (*in module pcvs.backend.utilities*), 71  
`lock_file()` (*in module pcvs.helpers.utils*), 108  
`lock_session_file()` (*in module pcvs.backend.session*), 68  
`LockException` (*class in pcvs.helpers.exceptions*), 91  
`LockException.BadOwnerError`, 91  
`LockException.TimeoutError`, 91

**M**

`machine` (*pcvs.backend.profile.Profile property*), 59  
`MAGIC_TOKEN` (*pcvs.orchestration.publishers.ResultFile attribute*), 112  
`main_detached_session()` (*in module pcvs.backend.session*), 68  
`MakefileBuildSystem` (*class in pcvs.backend.utilities*), 70  
`map_id()` (*pcvs.orchestration.publishers.ResultFileManager method*), 115  
`maps` (*pcvs.orchestration.publishers.ResultFileManager property*), 115  
`max_value` (*pcvs.helpers.criterion.Criterion property*), 87  
`MAXATTEMPTS_STR` (*pcvs.testing.test.Test attribute*), 77  
`MetaConfig` (*class in pcvs.helpers.system*), 103  
`metadata_prefix` (*pcvs.orchestration.publishers.ResultFile property*), 113  
`MetaDict` (*class in pcvs.helpers.system*), 104  
`min_value` (*pcvs.helpers.criterion.Criterion property*), 87  
`mod_deps` (*pcvs.testing.test.Test property*), 81  
`module`  
  `pcvs`, 119  
  `pcvs.backend`, 72  
  `pcvs.backend.bank`, 49  
  `pcvs.backend.config`, 53  
`PCVSBackend.profile`, 57  
`pcvs.backend.report`, 61  
`pcvs.backend.run`, 64  
`pcvs.backend.session`, 66  
`pcvs.backend.utilities`, 70

pcvs.cli, 75  
pcvs.cli.cli\_bank, 72  
pcvs.cli.cli\_config, 72  
pcvs.cli.cli\_profile, 73  
pcvs.cli.cli\_report, 74  
pcvs.cli.cli\_run, 74  
pcvs.cli.cli\_session, 74  
pcvs.cli.cli\_utilities, 75  
pcvs.converter, 86  
pcvs.converter.yaml\_converter, 85  
pcvs.helpers, 109  
pcvs.helpers.criterion, 86  
pcvs.helpers.exceptions, 89  
pcvs.helpers.git, 93  
pcvs.helpers.log, 101  
pcvs.helpers.pm, 101  
pcvs.helpers.system, 102  
pcvs.helpers.utils, 105  
pcvs.main, 118  
pcvs.orchestration, 117  
pcvs.orchestration.publishers, 109  
pcvs.testing, 84  
pcvs.testing.tedesc, 75  
pcvs.testing.test, 77  
pcvs.testing.testfile, 83  
pcvs.webview, 118  
ModuleManager (*class in pcvs.helpers.pm*), 101

## N

name (*pcvs.backend.bank.Bank property*), 50  
name (*pcvs.helpers.criterion.Criterion property*), 87  
name (*pcvs.helpers.git.Branch property*), 93  
name (*pcvs.testing.tedesc.TEDescriptor property*), 75  
name (*pcvs.testing.test.Test property*), 81  
name\_exist() (*pcvs.backend.bank.Bank method*), 50  
nb\_descs (*pcvs.testing.testfile.TestFile property*), 84  
new\_branch() (*pcvs.helpers.git.GitByAPI method*), 95  
new\_branch() (*pcvs.helpers.git.GitByCLI method*), 97  
NOSTART\_STR (*pcvs.testing.test.Test attribute*), 77  
not\_picked() (*pcvs.testing.test.Test method*), 81  
numeric (*pcvs.helpers.criterion.Criterion property*), 88

## O

open() (*pcvs.helpers.git.GitByAPI method*), 95  
open() (*pcvs.helpers.git.GitByCLI method*), 97  
open() (*pcvs.helpers.git.GitByGeneric method*), 99  
Orchestrator (*class in pcvs.orchestration*), 117  
OrchestratorException (*class in pcvs.helpers.exceptions*), 91  
OrchestratorException.CircularDependencyError, 91  
OrchestratorException.UndefDependencyError, 91  
out (*pcvs.helpers.utils.Program property*), 105

output (*pcvs.testing.test.Test property*), 81  
output\_info (*pcvs.testing.test.Test property*), 81  
override() (*pcvs.helpers.criterion.Criterion method*), 88

## P

path\_exist() (*pcvs.backend.bank.Bank method*), 50  
pcvs  
    module, 119  
pcvs.backend  
    module, 72  
pcvs.backend.bank  
    module, 49  
pcvs.backend.config  
    module, 53  
pcvs.backend.profile  
    module, 57  
pcvs.backend.report  
    module, 61  
pcvs.backend.run  
    module, 64  
pcvs.backend.session  
    module, 66  
pcvs.backend.utilities  
    module, 70  
pcvs.cli  
    module, 75  
pcvs.cli.cli\_bank  
    module, 72  
pcvs.cli.cli\_config  
    module, 72  
pcvs.cli.cli\_profile  
    module, 73  
pcvs.cli.cli\_report  
    module, 74  
pcvs.cli.cli\_run  
    module, 74  
pcvs.cli.cli\_session  
    module, 74  
pcvs.cli.cli\_utilities  
    module, 75  
pcvs.converter  
    module, 86  
pcvs.converter.yaml\_converter  
    module, 85  
pcvs.helpers  
    module, 109  
        pcvs.helpers.criterion  
            module, 86  
        pcvs.helpers.exceptions  
            module, 89  
        pcvs.helpers.git  
            module, 93  
        pcvs.helpers.log

```

    module, 101
pcvs.helpers.pm
    module, 101
pcvs.helpers.system
    module, 102
pcvs.helpers.utils
    module, 105
pcvs.main
    module, 118
pcvs.orchestration
    module, 117
pcvs.orchestration.publishers
    module, 109
pcvs.testing
    module, 84
pcvs.testing.tedesc
    module, 75
pcvs.testing.test
    module, 77
pcvs.testing.testfile
    module, 83
pcvs.webview
    module, 118
pick() (pcvs.testing.test.Test method), 81
pick_count() (pcvs.testing.test.Test method), 81
PluginException (class in pcvs.helpers.exceptions), 91
PluginException.BadStepError, 91
PluginException.LoadError, 91
PManager (class in pcvs.helpers.pm), 101
prefix (pcvs.backend.bank.Bank property), 51
prefix(pcvs.orchestration.publishers.BuildDirectoryManager
    property), 111
prefix (pcvs.orchestration.publishers.ResultFileManager
    property), 113
prepare() (in module pcvs.backend.run), 65
prepare() (pcvs.orchestration.publishers.BuildDirectoryManager
    method), 111
print_infos() (pcvs.orchestration.Orchestrator
    method), 117
print_progbar_walker() (in
    module
    pcvs.backend.run), 65
print_version() (in
    module
    pcvs.converter.yaml_converter), 85
print_version() (in module pcvs.main), 118
process() (in module pcvs.converter.yaml_converter),
    85
process() (pcvs.testing.testfile.TestFile method), 84
process_check_configs() (in
    module
    pcvs.backend.utilities), 71
process_check_directory() (in
    module
    pcvs.backend.utilities), 71
process_check_profiles() (in
    module
    pcvs.backend.utilities), 71
process_check_setup_file() (in
    module
    pcvs.backend.utilities), 71
process_discover_directory() (in
    module
    pcvs.backend.utilities), 72
process_dyn_setup_scripts() (in
    module
    pcvs.backend.run), 65
process_files() (in module pcvs.backend.run), 65
process_main_workflow() (in
    module
    pcvs.backend.run), 66
process_modifiers() (in
    module
    pcvs.converter.yaml_converter), 85
process_spack() (in module pcvs.backend.run), 66
process_static_yaml_files() (in
    module
    pcvs.backend.run), 66
Profile (class in pcvs.backend.profile), 57
profile_interactive_select() (in
    module
    pcvs.cli.cli_profile), 73
ProfileException (class in pcvs.helpers.exceptions),
    92
ProfileException.IncompleteError, 92
Program (class in pcvs.helpers.utils), 105
program_timeout() (in module pcvs.helpers.utils), 108
property() (pcvs.backend.session.Session method), 67
PublisherException (class
    in
    pcvs.helpers.exceptions), 92
PublisherException.AlreadyExistJobError, 92
PublisherException.BadMagicTokenError, 92
PublisherException.UnknownJobError, 92

R
readdata_prefix (pcvs.orchestration.publishers.ResultFileManager
    property), 113
rc (pcvs.backend.session.Session property), 67
rc (pcvs.helpers.utils.Program property), 105
reconstruct_map_data()
reconstruct_view_data() (pcvs.orchestration.publishers.ResultFileManager
    method), 115
ref_file (pcvs.backend.config.ConfigurationBlock
    property), 55
Reference (class in pcvs.helpers.git), 99
register_callback() (pcvs.backend.session.Session
    method), 68
register_sys_criterion()
    (pcvs.helpers.criterion.Serie class
        method), 88
register_view() (pcvs.orchestration.publishers.ResultFileManager
    method), 115
register_view_item() (pcvs.orchestration.publishers.ResultFileManager
    method), 115

```

remove\_session\_from\_file() (in `pcvs.backend.session`), 69  
replace\_placeholder() (in `pcvs.converter.yaml_converter`), 85  
replace\_special\_token() (in `pcvs.testing.testfile`), 84  
repo (`pcvs.helpers.git.Reference` property), 99  
Report (class in `pcvs.backend.report`), 61  
request\_git\_attr() (in module `pcvs.helpers.git`), 100  
res\_scheme (`pcvs.testing.test.Test` attribute), 81  
resolve\_a\_dep() (`pcvs.testing.test.Test` method), 81  
ResultFile (class in `pcvs.orchestration.publishers`), 112  
ResultFileManager (class in `pcvs.orchestration.publishers`), 114  
results (`pcvs.orchestration.publishers.BuildDirectoryManager` property), 111  
retcode (`pcvs.testing.test.Test` property), 81  
retrieve\_file() (`pcvs.backend.config.ConfigurationBlock` method), 55  
retrieve\_test() (`pcvs.orchestration.publishers.ResultFile` method), 113  
retrieve\_test() (`pcvs.orchestration.publishers.ResultFileManager` method), 116  
retrieve\_tests\_by\_name() (`pcvs.orchestration.publishers.ResultFileManager` method), 116  
revparse() (`pcvs.helpers.git.GitByAPI` method), 95  
revparse() (`pcvs.helpers.git.GitByCLI` method), 97  
revparse() (`pcvs.helpers.git.GitByGeneric` method), 99  
rm\_banklink() (in module `pcvs.backend.bank`), 52  
root (`pcvs.helpers.system.MetaConfig` attribute), 104  
rt\_pm\_string (`pcvs.testing.testfile.TestFile` attribute), 84  
run() (`pcvs.backend.Session` method), 68  
run() (`pcvs.helpers.utils.Program` method), 105  
run() (`pcvs.orchestration.Orchestrator` method), 118  
run\_detached() (`pcvs.backend.Session` method), 68  
RunException (class in `pcvs.helpers.exceptions`), 92  
RunException.InProgressError, 92  
RunException.NonZeroSetupScript, 92  
RunException.ProgramError, 92  
RunnerException (class in `pcvs.helpers.exceptions`), 92  
RunnerException.LaunchError, 92  
runtime (`pcvs.backend.profile.Profile` property), 59

**S**

sanitize\_values() (`pcvs.helpers.criterion.Criterion` method), 88  
save() (`pcvs.orchestration.publishers.ResultFile` method), 113  
save() (`pcvs.orchestration.publishers.ResultFileManager` method), 116

module save\_config() (`pcvs.orchestration.publishers.BuildDirectoryManager` method), 111  
module save\_extras() (`pcvs.orchestration.publishers.BuildDirectoryManager` method), 111  
module save\_final\_result() (`pcvs.testing.test.Test` method), 81  
save\_from\_archive() (`pcvs.backend.bank.Bank` method), 51  
save\_from\_buildir() (`pcvs.backend.bank.Bank` method), 51  
save\_new\_run() (`pcvs.backend.bank.Bank` method), 51  
save\_new\_run\_from\_instance() (`pcvs.backend.bank.Bank` method), 51  
save\_raw\_run() (`pcvs.testing.test.Test` method), 82  
save\_status() (`pcvs.testing.test.Test` method), 82  
save\_to\_global() (`pcvs.backend.bank.Bank` method), 51  
save\_yaml() (`pcvs.testing.testfile.TestFile` method), 84  
SCHED\_MAX\_ATTEMPTS (`pcvs.testing.test.Test` attribute), 77  
scope (`pcvs.backend.config.ConfigurationBlock` property), 55  
scope (`pcvs.backend.profile.Profile` property), 60  
scratch\_location (`pcvs.orchestration.publishers.BuildDirectoryManager` property), 112  
separate\_key\_and\_value() (in module `pcvs.converter.yaml_converter`), 86  
Serie (class in `pcvs.helpers.criterion`), 88  
Session (class in `pcvs.backend.session`), 66  
Session.State (class in `pcvs.backend.session`), 66  
session\_ids (`pcvs.backend.report.Report` property), 61  
session\_infos() (`pcvs.backend.report.Report` method), 61  
set\_branch() (`pcvs.helpers.git.GitByAPI` method), 95  
set\_branch() (`pcvs.helpers.git.GitByCLI` method), 97  
set\_head() (`pcvs.helpers.git.GitByGeneric` method), 99  
set\_identity() (`pcvs.helpers.git.GitByGeneric` method), 99  
set\_ifdef() (`pcvs.helpers.system.Config` method), 102  
set\_internal() (`pcvs.helpers.system.MetaConfig` method), 104  
set\_local\_path() (in module `pcvs.helpers.utils`), 108  
set\_nosquash() (`pcvs.helpers.system.Config` method), 102  
set\_path() (`pcvs.helpers.git.GitByGeneric` method), 99  
set\_with() (in module `pcvs.converter.yaml_converter`), 86  
short\_name (`pcvs.backend.config.ConfigurationBlock` property), 55  
show() (`pcvs.backend.bank.Bank` method), 51  
sid (`pcvs.orchestration.publishers.BuildDirectoryManager` property), 112  
single\_session\_build\_path() (`pcvs.backend.report.Report` method), 61

**s**

- single\_session\_config()  
*(pcvs.backend.report.Report method)*, 62
- single\_session\_get\_view()  
*(pcvs.backend.report.Report method)*, 62
- single\_session\_job\_cnt()  
*(pcvs.backend.report.Report method)*, 62
- single\_session\_labels()  
*(pcvs.backend.report.Report method)*, 62
- single\_session\_map\_id()  
*(pcvs.backend.report.Report method)*, 62
- single\_session\_status()  
*(pcvs.backend.report.Report method)*, 63
- single\_session\_tags()  
*(pcvs.backend.report.Report method)*, 63
- size (*pcvs.orchestration.publishers.ResultFileManager property*), 114
- SpackManager (*class in pcvs.helpers.pm*), 102
- split\_into\_configs()  
*(pcvs.backend.profile.Profile method)*, 60
- start\_automkill()  
*(in module pcvs.helpers.utils)*, 108
- start\_new\_runner()  
*(pcvs.orchestration.Orchestrator method)*, 118
- start\_run()  
*(pcvs.orchestration.Orchestrator method)*, 118
- start\_server()  
*(in module pcvs.backend.report)*, 63
- state (*pcvs.backend.session.Session property*), 68
- state (*pcvs.testing.test.Test property*), 82
- status\_view(*pcvs.orchestration.publishers.ResultFileManager property*), 116
- stop()  
*(pcvs.orchestration.Orchestrator class method)*, 118
- stop\_pending\_jobs()  
*(in module pcvs.backend.run)*, 66
- stop\_runners()  
*(pcvs.orchestration.Orchestrator method)*, 118
- storage\_order()  
*(in module pcvs.helpers.utils)*, 108
- store\_session\_to\_file()  
*(in module pcvs.backend.session)*, 69
- str\_dict\_as\_envvar()  
*(in module pcvs.helpers.utils)*, 108
- subtitle (*pcvs.helpers.criterion.Criterion property*), 88
- subtree (*pcvs.testing.test.Test property*), 82
- subtree\_view()  
*(pcvs.orchestration.publishers.ResultFileManager property)*, 116
- SUCCESS (*pcvs.testing.test.Test.State attribute*), 78

**T**

- tags (*pcvs.testing.test.Test property*), 82
- tags\_view(*pcvs.orchestration.publishers.ResultFileManager property*), 116
- te\_name (*pcvs.testing.test.Test property*), 82
- TEDescriptor (*class in pcvs.testing.tedesc*), 75
- terminate()  
*(in module pcvs.backend.run)*, 66
- Test (*class in pcvs.testing.test*), 77

**U**

- unlock\_file()  
*(in module pcvs.helpers.utils)*, 109
- unlock\_session\_file()  
*(in module pcvs.backend.session)*, 69
- update\_session\_from\_file()  
*(in module pcvs.backend.session)*, 69
- upload\_buildir\_results()  
*(in module pcvs.backend.report)*, 63
- use\_as\_global\_config()  
*(pcvs.orchestration.publishers.BuildDirectoryManager method)*, 112

**V**

- val\_scheme (*pcvs.testing.testfile.TestFile attribute*), 84
- ValidCombination()  
*(in module pcvs.helpers.criterion)*, 88
- validate()  
*(pcvs.helpers.system.Config method)*, 103
- validate()  
*(pcvs.helpers.system.ValidationScheme method)*, 104
- validate()  
*(pcvs.testing.testfile.TestFile method)*, 84
- validation\_default\_file  
*(pcvs.helpers.system.MetaConfig attribute)*, 104
- ValidationException  
*(class in pcvs.helpers.exceptions)*, 93
- ValidationException.FormatError, 93

`ValidationException.SchemeError`, 93  
`ValidationException.WrongTokenError`, 93  
`ValidationScheme` (*class in pcvr.helpers.system*), 104  
`values` (*pcvs.helpers.criterion.Criterion property*), 88  
`views` (*pcvs.orchestration.publishers.ResultFileManager property*), 117

## W

`WAITING` (*pcvs.backend.session.Session.State attribute*),  
67  
`WAITING` (*pcvs.testing.test.Test.State attribute*), 78