

---

# **salomeTools Documentation**

*Release 5.0.0dev*

**CEA DEN/DANS/DM2S/STMF/LGLS**

**Jun 26, 2018**



# CONTENTS

<b>1 Quick start</b>	<b>3</b>
1.1 Installation . . . . .	3
1.2 Configuration . . . . .	3
1.3 Usage of SAlomeTools . . . . .	5
<b>2 List of Commands</b>	<b>7</b>
2.1 Command doc . . . . .	8
2.2 Command config . . . . .	9
2.3 Command prepare . . . . .	11
2.4 Command compile . . . . .	13
2.5 Command launcher . . . . .	15
2.6 Command application . . . . .	16
2.7 Command log . . . . .	17
2.8 Command environ . . . . .	18
2.9 Command clean . . . . .	21
2.10 Command package . . . . .	22
2.11 Command generate . . . . .	24
<b>3 Developer documentation</b>	<b>25</b>
3.1 Add a user custom command . . . . .	26
<b>4 Code documentation</b>	<b>29</b>
4.1 src . . . . .	29
<b>5 Release Notes</b>	<b>71</b>
5.1 Release notes . . . . .	71
<b>Python Module Index</b>	<b>73</b>
<b>Index</b>	<b>75</b>





**Warning:** This documentation is under construction.

The **SalomeTools** (`sat`) is a suite of commands that can be used to perform operations on **SALOME**<sup>1</sup>.

For example, `sat` allows you to compile SALOME's codes (prerequisites, products) create application, run tests, create package, etc.

This utility code is a set of **Python**<sup>2</sup> scripts files.

Find a [pdf version of this documentation](#)

---

<sup>1</sup> <http://www.salome-platform.org>

<sup>2</sup> <https://docs.python.org/2.7>



## QUICK START

### 1.1 Installation

Usually user could find (and use) command **sat** directly after a ‘detar’ installation of SALOME.

```
tar -xf ../SALOME_xx.tgz
cd SALOME_xx
ls -l sat      # sat -> salomeTools/sat
```

Python package (scripts of *salomeTools*) actually remains in directory *salomeTools*.

### 1.2 Configuration

*salomeTools* uses files to store its configuration parameters.

There are several configuration files which are loaded by *salomeTools* in a specific order. When all the files are loaded a *config* object is created. Then, this object is passed to all command scripts.

#### 1.2.1 Syntax

The configuration files use a python-like structure format (see [config module](#)<sup>3</sup> for a complete description).

- {} define a dictionary,
- [] define a list,
- @ can be used to include a file,
- \$**prefix** reference to another parameter (ex: \$PRODUCT.name),
- # comments.

---

**Note:** in this documentation a reference to a configuration parameter will be noted XXX.YYY.

---

#### 1.2.2 Description

##### VARS section

This section is dynamically created by *salomeTools* at run time.

It contains information about the environment: date, time, OS, architecture etc.

---

<sup>3</sup> <http://www.red-dove.com/config-doc/>

```
# to get the current setting
sat config --value VARS
```

## PRODUCTS section

This section is defined in the product file.

It contains instructions on how to build a version of SALOME (list of prerequisites-products and versions)

```
# to get the current setting
sat config SALOME-xx --value PRODUCTS
```

## APPLICATION section

This section is optional, it is also defined in the product file.

It gives additional parameters to create an application based on SALOME, as versions of products to use.

```
# to get the current setting
sat config SALOME-xx --value APPLICATION
```

## USER section

This section is defined by the user configuration file, `~/.salomeTools/salomeTools.pyconf`.

The USER section defines some parameters (not exhaustive):

- **workDir** :
  - The working directory.
  - Each product will be usually installed here (in sub-directories).
- **browser** : The web browser to use (*firefox*).
- **editor** : The editor to use (*vi*, *pluma*).
- and other user preferences.

```
# to get the current setting
sat config SALOME-xx --value USER
```



## 1.3 Usage of SALomeTools

### 1.3.1 Usage

sat usage is a Command Line Interface (CLI<sup>4</sup>).

```
sat [generic_options] [command] [product] [command_options]
```

#### Options of sat

Useful *not exhaustive* generic options of *sat* CLI.

##### ***-help or -h***

Get help as simple text.

```
sat --help           # get the list of existing commands
sat --help compile  # get the help on a specific command 'compile'
```

##### ***-debug or -g***

Execution in debug mode allows to see more trace and *stack* if an exception is raised.

##### ***-verbose or -v***

Change verbosity level (default is 3).

```
# for product 'SALOME_xx' for example
# execute compile command in debug mode with trace level 4
sat -g -v 4 compile SALOME_xx
```

### 1.3.2 Build a SALOME product

#### Get the list of available products

To get the list of the current available products in your context:

```
sat config --list
```

#### Prepare sources of a product

To prepare (get) *all* the sources of a product (*SALOME\_xx* for example):

```
sat prepare SALOME_xx
```

The sources are usually copied in directories

$\$USER.workDir + SALOME\_xx \dots + SOURCES + \$PRODUCT.name$

<sup>4</sup> [https://en.wikipedia.org/wiki/Command-line\\_interface](https://en.wikipedia.org/wiki/Command-line_interface)

## Compile SALOME

To compile products:

```
# compile all prerequisites/products
sat compile SALOME_xx

# compile only 2 products (KERNEL and SAMPLES), if not done yet
sat compile SALOME_xx --products KERNEL,SAMPLES

# compile only 2 products, unconditionally
sat compile SALOME_xx ---products SAMPLES --clean_all
```

The products are usually build in the directories

*\$USER.workDir + SALOME\_xx... + BUILD + \$PRODUCT.name*

The products are usually installed in the directories

*\$USER.workDir + SALOME\_xx... + INSTALL + \$PRODUCT.name*

**LIST OF COMMANDS**

## 2.1 Command doc

### 2.1.1 Description

The **doc** command displays sat documentation.

### 2.1.2 Usage

- Show (in a web browser) the sat documentation in format xml/html:

```
sat doc --xml
```

- Show (in evince, for example) the (same) sat documentation in format pdf:

```
sat doc --pdf
```

- Edit and modify in your preference user editor the sources files (rst) of sat documentation:

```
sat doc --edit
```

- get information how to compile locally sat documentation (from the sources files):

```
sat doc --compile
```

### 2.1.3 Some useful configuration paths

- **USER**
  - **browser** : The browser used to show the html files (*firefox* for example).
  - **pdf\_viewer** : The viewer used to show the pdf files (*evince* for example).
  - **editor** : The editor used to edit ascii text files (*pluma* or *gedit* for example).

## 2.2 Command config

### 2.2.1 Description

The **config** command manages sat configuration. It allows display, manipulation and operation on configuration files

### 2.2.2 Usage

- Edit the user personal configuration file `$HOME/.salomeTools/SAT.pyconf`. It is used to store the user personal choices, like the favorite editor, browser, pdf viewer:

```
sat config --edit
```

- List the available applications (they come from the sat projects defined in `data/local.pyconf`):

```
sat config --list
```

- Edit the configuration of an application:

```
sat config <application> --edit
```

- Copy an application configuration file into the user personal directory:

```
sat config <application> --copy [new_name]
```

- Print the value of a configuration parameter.  
Use the automatic completion to get recursively the parameter names.  
Use `--no_label` option to get *only* the value, *without* label (useful in automatic scripts).  
Examples (with *SALOME-xx* as *SALOME-8.4.0*):

```
# sat config --value <parameter_path>
sat config --value . # all the configuration
sat config --value LOCAL
sat config --value LOCAL.workdir

# sat config <application> --value <parameter_path>
sat config SALOME-xx --value APPLICATION.workdir
sat config SALOME-xx --no_label --value APPLICATION.workdir
```

- Print in one-line-by-value mode the value of a configuration parameter, with its source *expression*, if any.  
This is a debug mode, useful for developers.  
Prints the parameter path, the source expression if any, and the final value:

```
sat config SALOME-xx -g USER
```

---

**Note:** And so, *not only for fun*, to get **all expressions** of configuration

```
sat config SALOME-xx -g . | grep -e "-->"
```

---

- Print the patches that are applied:

```
sat config SALOME-xx --show_patches
```

- Get information on a product configuration:

```
# sat config <application> --info <product>
sat config SALOME-xx --info KERNEL
sat config SALOME-xx --info qt
```

## 2.2.3 Some useful configuration paths

Exploring a current configuration.

- **PATHS:** To get list of directories where to find files.
- **USER:** To get user preferences (editor, pdf viewer, web browser, default working dir).

sat commands:

```
sat config SALOME-xx -v PATHS
sat config SALOME-xx -v USERS
```

## 2.3 Command prepare

### 2.3.1 Description

The **prepare** command brings the sources of an application in the *sources application directory*, in order to compile them with the `compile` command.

The sources can be prepared from VCS software (*cvs*, *svn*, *git*), an archive or a directory.

**Warning:** When `sat` prepares a product, it first removes the existing directory, except if the development mode is activated. When you are working on a product, you need to declare in the application configuration this product in **dev** mode.

### 2.3.2 Remarks

#### VCS bases (*git*, *svn*, *cvs*)

The *prepare* command does not manage authentication on the *cvs* server. For example, to prepare modules from a *cvs* server, you first need to login once.

To avoid typing a password for each product, you may use a *ssh* key with passphrase, or store your password (in *.cvspass* or *.gitconfig* files). If you have security concerns, it is also possible to use a *bash* agent and type your password only once.

#### Dev mode

By default *prepare* uses *export* mode: it creates an image of the sources, corresponding to the tag or branch specified, without any link to the VCS base. To perform a *checkout* (*svn*, *cvs*) or a *git clone* (*git*), you need to declare the product in dev mode in your application configuration: edit the application configuration file (*pyconf*) and modify the product declaration:

```
sat config <application> -e
# and edit the product section:
# <product> : {tag : "my_tag", dev : "yes", debug : "yes"}
```

The first time you will execute the *sat prepare* command, your module will be downloaded in *checkout* mode (inside the *SOURCES* directory of the application). Then, you can develop in this repository, and finally push them in the base when they are ready. If you type during the development process by mistake a *sat prepare* command, the sources in dev mode will not be altered/removed (Unless you use *-f* option)

### 2.3.3 Usage

- Prepare the sources of a complete application in *SOURCES* directory (all products):

```
sat prepare <application>
```

- Prepare only some modules:

```
sat prepare <application> --products <product1>,<product2> ...
```

- Use *-force* to force to prepare the products in development mode (this will remove the sources and do a new clone/checkout):

```
sat prepare <application> --force
```

- Use `--force_patch` to force to apply patch to the products in development mode (otherwise they are not applied):

```
sat prepare <application> --force_patch
```

### 2.3.4 Some useful configuration paths

Command `sat prepare` uses the *pyconf file configuration* of each product to know how to get the sources.

---

**Note:** to verify configuration of a product, and get name of this *pyconf files configuration*

```
sat config <application> --info <product>
```

---

- **get\_method:** the method to use to prepare the module, possible values are cvs, git, archive, dir.
- **git\_info :** (used if `get_method = git`) information to prepare sources from git.
- **svn\_info :** (used if `get_method = svn`) information to prepare sources from cvs.
- **cvs\_info :** (used if `get_method = cvs`) information to prepare sources from cvs.
- **archive\_info :** (used if `get_method = archive`) the path to the archive.
- **dir\_info :** (used if `get_method = dir`) the directory with the sources.



## 2.4 Command compile

### 2.4.1 Description

The **compile** command allows compiling the products of a SALOME<sup>5</sup> application.

### 2.4.2 Usage

- Compile a complete application:

```
sat compile <application>
```

- Compile only some products:

```
sat compile <application> --products <product1>,<product2> ...
```

- Use *sat -t* to duplicate the logs in the terminal (by default the log are stored and displayed with *sat log* command):

```
sat -t compile <application> --products <product1>
```

- Compile a module and its dependencies:

```
sat compile <application> --products med --with_fathers
```

- Compile a module and the modules depending on it (for example plugins):

```
sat compile <application> --products med --with_children
```

- Clean the build and install directories before starting compilation:

```
sat compile <application> --products GEOM --clean_all
```

---

#### Note:

a warning will be shown if option *-products* is missing  
(as it will clean everything)

---

- Clean only the install directories before starting compilation:

```
sat compile <application> --clean_install
```

- Add options for make:

```
sat compile <application> --products <product> --make_flags <flags>
```

- Use the *-check* option to execute the unit tests after compilation:

```
sat compile <application> --check
```

- Remove the build directory after successful compilation (some build directory like qt are big):

```
sat compile <application> --products qt --clean_build_after
```

- Stop the compilation as soon as the compilation of a module fails:

---

<sup>5</sup> <http://www.salome-platform.org>

```
sat compile <product> --stop_first_fail
```

- Do not compile, just show if products are installed or not, and where is the installation:

```
sat compile <application> --show
```

### 2.4.3 Some useful configuration pathes

The way to compile a product is defined in the *pyconf file configuration*. The main options are:

- **build\_source** : the method used to build the product (cmake/autotools/script)
- **compil\_script** : the compilation script if build\_source is equal to “script”
- **cmake\_options** : additional options for cmake.
- **nb\_proc** : number of jobs to use with make for this product.

## 2.5 Command launcher

### 2.5.1 Description

The **launcher** command creates a SALOME launcher, a python script file to start SALOME<sup>6</sup>.

### 2.5.2 Usage

- Create a launcher:

```
sat launcher <application>
```

Generate a launcher in the application directory, i.e \$APPLICATION.workdir.

- Create a launcher with a given name (default name is APPLICATION.profile.launcher\_name)

```
sat launcher <application> --name ZeLauncher
```

The launcher will be called *ZeLauncher*.

- Set a specific resources catalog:

```
sat launcher <application> --catalog <path of a salome resources catalog>
```

Note that the catalog specified will be copied to the profile directory.

- Generate the catalog for a list of machines:

```
sat launcher <application> --gencat <list of machines>
```

This will create a catalog by querying each machine (memory, number of processor) with ssh.

- Generate a mesa launcher (if mesa and llvm are parts of the application). Use this option only if you have to use salome through ssh and have problems with ssh X forwarding of OpenGL modules (like Paravis):

```
sat launcher <application> --use_mesa
```

### 2.5.3 Configuration

Some useful configuration pathes:

- **APPLICATION.profile**
  - **product** : the name of the profile product (the product in charge of holding the application stuff, like logos, splashscreen)
  - **launcher\_name** : the name of the launcher.

<sup>6</sup> <http://www.salome-platform.org>

## 2.6 Command application

### 2.6.1 Description

The **application** command creates a virtual SALOME<sup>7</sup> application. Virtual SALOME applications are used to start SALOME when distribution is needed.

### 2.6.2 Usage

- Create an application:

```
sat application <application>
```

Create the virtual application directory in the salomeTool application directory \$APPLICATION.workdir.

- Give a name to the application:

```
sat application <application> --name <my_application_name>
```

*Remark:* this option overrides the name given in the virtual\_app section of the configuration file \$APPLICATION.virtual\_app.name.

- Change the directory where the application is created:

```
sat application <application> --target <my_application_directory>
```

- Set a specific SALOME<sup>8</sup> resources catalog (it will be used for the distribution of components on distant machines):

```
sat application <application> --catalog <path_to_catalog>
```

Note that the catalog specified will be copied to the application directory.

- Generate the catalog for a list of machines:

```
sat application <application> --gencat machine1,machine2,machine3
```

This will create a catalog by querying each machine through ssh protocol (memory, number of processor) with ssh.

- Generate a mesa application (if mesa and llvm are parts of the application). Use this option only if you have to use salome through ssh and have problems with ssh X forwarding of OpenGL modules (like Paravis):

```
sat launcher <application> --use_mesa
```

### 2.6.3 Some useful configuration paths

The virtual application can be configured with the virtual\_app section of the configuration file.

- **APPLICATION.virtual\_app**
  - **name** : name of the launcher (to replace the default runAppli).
  - **application\_name** : (optional) the name of the virtual application directory, if missing the default value is \$name + \_appli.

---

<sup>7</sup> <http://www.salome-platform.org>

<sup>8</sup> <http://www.salome-platform.org>

## 2.7 Command log

### 2.7.1 Description

The **log** command displays sat log in a web browser or in a terminal.

### 2.7.2 Usage

- Show (in a web browser) the log of the commands corresponding to an application:

```
sat log <application>
```

- Show the log for commands that do not use any application:

```
sat log
```

- The `-terminal` (or `-t`) display the log directly in the terminal, through a [CLI](#)<sup>9</sup> interactive menu:

```
sat log <application> --terminal
```

- The `-last` option displays only the last command:

```
sat log <application> --last
```

- To access the last compilation log in terminal mode, use `-last_terminal` option:

```
sat log <application> --last_terminal
```

- The `-clean` (int) option erases the n older log files and print the number of remaining log files:

```
sat log <application> --clean 50
```

### 2.7.3 Some useful configuration pathes

- **USER**
  - **browser** : The browser used to show the log (by default *firefox*).
  - **log\_dir** : The directory used to store the log files.

<sup>9</sup> [https://en.wikipedia.org/wiki/Command-line\\_interface](https://en.wikipedia.org/wiki/Command-line_interface)

## 2.8 Command environ

### 2.8.1 Description

The **environ** command generates the environment files used to run and compile your application (as SALOME<sup>10</sup> is an example).

---

**Note:** these files are **not** required, salomeTool set the environment himself, when compiling. And so does the salome launcher.

These files are useful when someone wants to check the environment. They could be used in debug mode to set the environment for *gdb*.

---

The configuration part at the end of this page explains how to specify the environment used by sat (at build or run time), and saved in some files by *sat environ* command.

### 2.8.2 Usage

- Create the shell environment files of the application:

```
sat environ <application>
```

- Create the environment files of the application for a given shell. Options are bash, bat (for windows) and cfg (the configuration format used by SALOME<sup>11</sup>):

```
sat environ <application> --shell [bash|cfg|all]
```

- Use a different prefix for the files (default is 'env'):

```
# This will create file <prefix>_launch.sh, <prefix>_build.sh  
sat environ <application> --prefix <prefix>
```

- Use a different target directory for the files:

```
# This will create file env_launch.sh, env_build.sh  
# in the directory corresponding to <path>  
sat environ <application> --target <path>
```

- Generate the environment files only with the given products:

```
# This will create the environment files only for the given products  
# and their prerequisites.  
# It is useful when you want to visualise which environment uses  
# sat to compile a given product.  
sat environ <application> --product <product1>,<product2>, ...
```

### 2.8.3 Configuration

The specification of the environment can be done through several mechanisms.

1. For salome products (the products with the property `is_SALOME_module` as `yes`) the environment is set automatically by sat, in respect with SALOME<sup>12</sup> requirements.

---

<sup>10</sup> <http://www.salome-platform.org>

<sup>11</sup> <http://www.salome-platform.org>

<sup>12</sup> <http://www.salome-platform.org>

- For other products, the environment is set with the use of the `environ` section within the `pyconf` file of the product. The user has two possibilities, either set directly the environment within the section, or specify a python script which will be used to set the environment programmatically.

Within the section, the user can define environment variables. He can also modify `PATH` variables, by appending or prepending directories. In the following example, we prepend `<install_dir>/lib` to `LD_LIBRARY_PATH` (note the *left first* underscore), append `<install_dir>/lib` to `PYTHONPATH` (note the *right last* underscore), and set `LAPACK_ROOT_DIR` to `<install_dir>`:

```
environ :
{
  _LD_LIBRARY_PATH : $install_dir + $VARS.sep + "lib"
  PYTHONPATH_     : $install_dir + $VARS.sep + "lib"
  LAPACK_ROOT_DIR : $install_dir
}
```

It is possible to distinguish the build environment from the launch environment: use a subsection called *build* or *launch*. In the example below, `LD_LIBRARY_PATH` and `PYTHONPATH` are only modified at run time, not at compile time:

```
environ :
{
  build :
  {
    LAPACK_ROOT_DIR : $install_dir
  }
  launch :
  {
    LAPACK_ROOT_DIR : $install_dir
    _LD_LIBRARY_PATH : $install_dir + $VARS.sep + "lib"
    PYTHONPATH_     : $install_dir + $VARS.sep + "lib"
  }
}
```

- The last possibility is to set the environment with a python script. The script should be provided in the `products/env_scripts` directory of the `sat` project, and its name is specified in the environment section with the key `environ.env_script`:

```
environ :
{
  env_script : 'lapack.py'
}
```

Please note that the two modes are complementary and are both taken into account. Most of the time, the first mode is sufficient.

The second mode can be used when the environment has to be set programmatically. The developer implements a `handle` (as a python method) which is called by `sat` to set the environment. Here is an example:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import os.path
import platform

def set_env(env, prereq_dir, version):
    env.set("TRUST_ROOT_DIR", prereq_dir)
    env.prepend('PATH', os.path.join(prereq_dir, 'bin'))
    env.prepend('PATH', os.path.join(prereq_dir, 'include'))
    env.prepend('LD_LIBRARY_PATH', os.path.join(prereq_dir, 'lib'))
    return
```

SalomeTools defines four handles:

- **set\_env(env, prereq\_dir, version)** : used at build and run time.
- **set\_env\_launch(env, prereq\_dir, version)** : used only at run time (if defined!)
- **set\_env\_build(env, prereq\_dir, version)** : used only at build time (if defined!)
- **set\_native\_env(env)** : used only for native products, at build and run time.



## 2.9 Command clean

### 2.9.1 Description

The **clean** command removes products in the *source*, *build*, or *install* directories of an application. These directories are usually named `SOURCES`, `BUILD`, `INSTALL`.

Use the options to define what directories you want to suppress and to set the list of products

### 2.9.2 Usage

- Clean all previously created *build* and *install* directories (example application as *SALOME\_xx*):

```
# take care, is long time to restore, sometimes
sat clean SALOME-xx --build --install
```

- Clean previously created *build* and *install* directories, only for products with property *is\_salome\_module*:

```
sat clean SALOME-xxx --build --install \
    --properties is_salome_module:yes
```

### 2.9.3 Availables options

- **-products** : Products to clean.
- **-properties** :

Filter the products by their properties.

Syntax: `-properties <property>:<value>`

- **-sources** : Clean the product source directories.
- **-build** : Clean the product build directories.
- **-install** : Clean the product install directories.
- **-all** : Clean the product source, build and install directories.
- **-sources\_without\_dev** :

Do not clean the products in development mode, (they could have [VCS](https://en.wikipedia.org/wiki/Version_control)<sup>13</sup> commits pending).

### 2.9.4 Some useful configuration paths

No specific configuration.

<sup>13</sup> [https://en.wikipedia.org/wiki/Version\\_control](https://en.wikipedia.org/wiki/Version_control)

## 2.10 Command package

### 2.10.1 Description

The **package** command creates a SALOME<sup>14</sup> archive (usually a compressed Tar<sup>15</sup> file .tgz). This tar file is used later to install SALOME on other remote computer.

Depending on the selected options, the archive includes sources and binaries of SALOME products and prerequisites.

Usually utility *salomeTools* is included in the archive.

---

**Note:** By default the package includes the sources of prerequisites and products. To select a subset use the `-without_property` or `-with_vcs` options.

---

### 2.10.2 Usage

- Create a package for a product (example as *SALOME\_xx*):

```
sat package SALOME_xx
```

This command will create an archive named *SALOME\_xx.tgz* in the working directory (`USER.workDir`). If the archive already exists, do nothing.

- Create a package with a specific name:

```
sat package SALOME_xx --name YourSpecificName
```

---

**Note:** By default, the archive is created in the working directory of the user (`USER.workDir`).

If the option `-name` is used with a path (relative or absolute) it will be used.

If the option `-name` is not used and binaries (prerequisites and products) are included in the package, the OS<sup>16</sup> architecture will be appended to the name (example: *SALOME\_xx-CO7.tgz*).

Examples:

```
# Creates SALOME_xx.tgz in $USER.workDir
sat package SALOME_xx

# Creates SALOME_xx<arch>.tgz in $USER.workDir
sat package SALOME_xx --binaries

# Creates MySpecificName.tgz in $USER.workDir
sat package SALOME_xx --name MySpecificName
```

- 
- Force the creation of the archive (if it already exists):

```
sat package SALOME_xx --force
```

- Include the binaries in the archive (products and prerequisites):

```
sat package SALOME_xx --binaries
```

This command will create an archive named *SALOME\_xx \_<arch>.tgz* where `<arch>` is the OS<sup>17</sup> ar-

---

<sup>14</sup> <http://www.salome-platform.org>

<sup>15</sup> [https://en.wikipedia.org/wiki/Tar\\_\(computing\)](https://en.wikipedia.org/wiki/Tar_(computing))

<sup>16</sup> [https://en.wikipedia.org/wiki/Operating\\_system](https://en.wikipedia.org/wiki/Operating_system)

<sup>17</sup> [https://en.wikipedia.org/wiki/Operating\\_system](https://en.wikipedia.org/wiki/Operating_system)

chitecture of the machine.

- Do not delete Version Control System (VCS<sup>18</sup>) informations from the configurations files of the embedded salomeTools:

```
sat package SALOME_xx --with_vcs
```

The version control systems known by this option are CVS<sup>19</sup>, SVN<sup>20</sup> and Git<sup>21</sup>.

### 2.10.3 Some useful configuration pathes

No specific configuration.

---

<sup>18</sup> [https://en.wikipedia.org/wiki/Version\\_control](https://en.wikipedia.org/wiki/Version_control)

<sup>19</sup> [https://fr.wikipedia.org/wiki/Concurrent\\_versions\\_system](https://fr.wikipedia.org/wiki/Concurrent_versions_system)

<sup>20</sup> [https://en.wikipedia.org/wiki/Apache\\_Subversion](https://en.wikipedia.org/wiki/Apache_Subversion)

<sup>21</sup> <https://git-scm.com>

## 2.11 Command generate

### 2.11.1 Description

The **generate** command generates and compile SALOME modules from cpp modules using YACSGEN.

---

**Note:** This command uses YACSGEN to generate the module. It needs to be specified with `-yacsgen` option, or defined in the product or by the environment variable `$YACSGEN_ROOT_DIR`.

---

### 2.11.2 Remarks

- This command will only apply on the CPP modules of the application, those who have both properties:

```
cpp : "yes"  
generate : "yes"
```

- The cpp module are usually computational components, and the generated module brings the CORBA layer which allows distributing the component on remote machines. cpp modules should conform to YACSGEN/hxx2salome requirements (please refer to YACSGEN documentation)

### 2.11.3 Usage

- Generate all the modules of a product:

```
sat generate <application>
```

- Generate only specific modules:

```
sat generate <application> --products <list_of_products>
```

Remark: modules which don't have the *generate* property are ignored.

- Use a specific version of YACSGEN:

```
sat generate <application> --yacsgen <path_to_yacsgen>
```

**DEVELOPER DOCUMENTATION**

## 3.1 Add a user custom command

### 3.1.1 Introduction

---

**Note:** This documentation is for Python<sup>22</sup> developers.

---

The salomeTools product provides a simple way to develop commands. The first thing to do is to add a file with `.py` extension in the `commands` directory of salomeTools.

Here are the basic requirements that must be followed in this file in order to add a command.

### 3.1.2 Basic requirements

By adding a file `mycommand.py` in the `commands` directory, salomeTools will define a new command named `mycommand`.

In `mycommand.py`, there must be the following method:

```
def run(args, runner, logger):
    # your algorithm ...
    pass
```

In fact, at this point, the command will already be functional. But there are some useful services provided by salomeTools :

- You can give some options to your command:

```
import src

# Define all possible option for mycommand command : 'sat mycommand <options>'
parser = src.options.Options()
parser.add_option('m', 'myoption', \
                 'boolean', 'myoption', \
                 'My option changes the behavior of my command.')

def run(args, runner, logger):
    # Parse the options
    (options, args) = parser.parse_args(args)
    # algorithm
```

- You can add a `description` method that will display a message when the user will call the help:

```
import src

# Define all possible option for mycommand command : 'sat mycommand <options>'
parser = src.options.Options()
parser.add_option('m', 'myoption', \
                 'boolean', 'myoption', \
                 'My option changes the behavior of my command.')

def description():
    return _("The help of mycommand.")

def run(args, runner, logger):
    # Parse the options
    (options, args) = parser.parse_args(args)
    # algorithm
```

---

<sup>22</sup> <https://docs.python.org/2.7>

### 3.1.3 HowTo access salomeTools config and other commands

The *runner* variable is a python instance of *Sat* class. It gives access to *runner.cfg* which is the data model defined from all *configuration pyconf files* of salomeTools For example, *runner.cfg.APPLICATION.workdir* contains the root directory of the current application.

The *runner* variable gives also access to other commands of salomeTools:

```
# as CLI_ 'sat prepare ...'
runner.prepare(runner.cfg.VARS.application)
```

### 3.1.4 HowTo logger

The *logger* variable is an instance of the *Logger* class. It gives access to the *write* method.

When this method is called, the message passed as parameter will be displayed in the terminal and written in an xml log file.

```
logger.write("My message", 3) # 3 as default
```

The second argument defines the level of verbosity that is wanted for this message. It has to be between 1 and 5 (the most verbose level).

### 3.1.5 HELLO example

Here is a *hello* command, file *commands/hello.py*:

```
import src

"""
hello.py
Define all possible options for hello command:
sat hello <options>
"""

parser = src.options.Options()
parser.add_option('f', 'french', 'boolean', 'french', "french set hello message in_
↪french.")

def description():
    return _("The help of hello.")

def run(args, runner, logger):
    # Parse the options
    (options, args) = parser.parse_args(args)
    # algorithm
    if not options.french:
        logger.write('HELLO! WORLD!\n')
    else:
        logger.write('Bonjour tout le monde!\n')
```

A first call of *hello*:

```
# Get the help of hello:
./sat --help hello

# To get bonjour
./sat hello --french
Bonjour tout le monde!
```

(continues on next page)

(continued from previous page)

```
# To get hello
./sat hello
HELLO! WORLD!

# To get the log
./sat log
```



## CODE DOCUMENTATION

### 4.1 src

#### 4.1.1 src package

##### Subpackages

##### src.colorama package

##### Submodules

##### src.colorama.ansi module

This module generates ANSI character codes to printing colors to terminals. See: [http://en.wikipedia.org/wiki/ANSI\\_escape\\_code](http://en.wikipedia.org/wiki/ANSI_escape_code)

**class** src.colorama.ansi.**AnsiBack**

Bases: *src.colorama.ansi.AnsiCodes* (page 29)

**BLACK** = 40

**BLUE** = 44

**CYAN** = 46

**GREEN** = 42

**LIGHTBLACK\_EX** = 100

**LIGHTBLUE\_EX** = 104

**LIGHTCYAN\_EX** = 106

**LIGHTGREEN\_EX** = 102

**LIGHTMAGENTA\_EX** = 105

**LIGHTRED\_EX** = 101

**LIGHTWHITE\_EX** = 107

**LIGHTYELLOW\_EX** = 103

**MAGENTA** = 45

**RED** = 41

**RESET** = 49

**WHITE** = 47

**YELLOW** = 43

```
class src.colorama.ansi.AnsiCodes
    Bases: object

class src.colorama.ansi.AnsiCursor
    Bases: object

    BACK (n=1)

    DOWN (n=1)

    FORWARD (n=1)

    POS (x=1, y=1)

    UP (n=1)

class src.colorama.ansi.AnsiFore
    Bases: src.colorama.ansi.AnsiCodes (page 29)

    BLACK = 30

    BLUE = 34

    CYAN = 36

    GREEN = 32

    LIGHTBLACK_EX = 90

    LIGHTBLUE_EX = 94

    LIGHTCYAN_EX = 96

    LIGHTGREEN_EX = 92

    LIGHTMAGENTA_EX = 95

    LIGHTRED_EX = 91

    LIGHTWHITE_EX = 97

    LIGHTYELLOW_EX = 93

    MAGENTA = 35

    RED = 31

    RESET = 39

    WHITE = 37

    YELLOW = 33

class src.colorama.ansi.AnsiStyle
    Bases: src.colorama.ansi.AnsiCodes (page 29)

    BRIGHT = 1

    DIM = 2

    NORMAL = 22

    RESET_ALL = 0

src.colorama.ansi.clear_line (mode=2)
src.colorama.ansi.clear_screen (mode=2)
src.colorama.ansi.code_to_chars (code)
src.colorama.ansi.set_title (title)
```

**src.colorama.ansitowin32 module**

**class** `src.colorama.ansitowin32.AnsiToWin32` (*wrapped, convert=None, strip=None, autoreset=False*)

Bases: `object`

Implements a 'write()' method which, on Windows, will strip ANSI character sequences from the text, and if outputting to a tty, will convert them into win32 function calls.

**ANSI\_CSI\_RE** = `<_sre.SRE_Pattern object>`

**ANSI\_OSC\_RE** = `<_sre.SRE_Pattern object>`

**call\_win32** (*command, params*)

**convert\_ansi** (*paramstring, command*)

**convert\_osc** (*text*)

**extract\_params** (*command, paramstring*)

**get\_win32\_calls** ()

**reset\_all** ()

**should\_wrap** ()

True if this class is actually needed. If false, then the output stream will not be affected, nor will win32 calls be issued, so wrapping stdout is not actually required. This will generally be False on non-Windows platforms, unless optional functionality like autoreset has been requested using kwargs to `init()`

**write** (*text*)

**write\_and\_convert** (*text*)

Write the given text to our wrapped stream, stripping any ANSI sequences from the text, and optionally converting them into win32 calls.

**write\_plain\_text** (*text, start, end*)

**class** `src.colorama.ansitowin32.StreamWrapper` (*wrapped, converter*)

Bases: `object`

Wraps a stream (such as stdout), acting as a transparent proxy for all attribute access apart from method 'write()', which is delegated to our Converter instance.

**write** (*text*)

`src.colorama.ansitowin32.is_a_tty` (*stream*)

`src.colorama.ansitowin32.is_stream_closed` (*stream*)

**src.colorama.initialise module**

`src.colorama.initialise.colorama_text` (*\*args, \*\*kwds*)

`src.colorama.initialise.deinit` ()

`src.colorama.initialise.init` (*autoreset=False, convert=None, strip=None, wrap=True*)

`src.colorama.initialise.reinit` ()

`src.colorama.initialise.reset_all` ()

`src.colorama.initialise.wrap_stream` (*stream, convert, strip, autoreset, wrap*)

**src.colorama.win32 module**`src.colorama.win32.SetConsoleTextAttribute (*_)``src.colorama.win32.winapi_test (*_)`**src.colorama.winterm module****class** `src.colorama.winterm.WinColor`Bases: `object``BLACK = 0``BLUE = 1``CYAN = 3``GREEN = 2``GREY = 7``MAGENTA = 5``RED = 4``YELLOW = 6`**class** `src.colorama.winterm.WinStyle`Bases: `object``BRIGHT = 8``BRIGHT_BACKGROUND = 128``NORMAL = 0`**class** `src.colorama.winterm.WinTerm`Bases: `object``back` (*back=None, light=False, on\_stderr=False*)`cursor_adjust` (*x, y, on\_stderr=False*)`erase_line` (*mode=0, on\_stderr=False*)`erase_screen` (*mode=0, on\_stderr=False*)`fore` (*fore=None, light=False, on\_stderr=False*)`get_attrs` ()`get_position` (*handle*)`reset_all` (*on\_stderr=None*)`set_attrs` (*value*)`set_console` (*attrs=None, on\_stderr=False*)`set_cursor_position` (*position=None, on\_stderr=False*)`set_title` (*title*)`style` (*style=None, on\_stderr=False*)

## Module contents

### Submodules

#### src.ElementTree module

```
src.ElementTree.Comment (text=None)
src.ElementTree.dump (elem)
src.ElementTree.Element (tag, attrib={}, **extra)
class src.ElementTree.ElementTree (element=None, file=None)

    find (path)
    findall (path)
    findtext (path, default=None)
    getiterator (tag=None)
    getroot ()
    parse (source, parser=None)
    write (file, encoding='us-ascii')
src.ElementTree.fromstring (text)
src.ElementTree.iselement (element)
class src.ElementTree.iterparse (source, events=None)

    next ()
src.ElementTree.parse (source, parser=None)
src.ElementTree.PI (target, text=None)
src.ElementTree.ProcessingInstruction (target, text=None)
class src.ElementTree.QName (text_or_uri, tag=None)
src.ElementTree.SubElement (parent, tag, attrib={}, **extra)
src.ElementTree.tostring (element, encoding=None)
class src.ElementTree.TreeBuilder (element_factory=None)

    close ()
    data (data)
    end (tag)
    start (tag, attrs)
src.ElementTree.XML (text)
class src.ElementTree.XMLTreeBuilder (html=0, target=None)

    close ()
    doctype (name, pubid, system)
    feed (data)
```

## src.architecture module

In this file : all the stuff that can change with the architecture on which SAT is running

`src.architecture.get_distrib_version(distrib, codes)`

Gets the version of the distribution

### Parameters

- **str** (*distrib*) – The distribution on which the version will be found.
- **L{Mapping}** (*codes*) – The map containing distribution correlation table.

**Returns** The version of the distribution on which salomeTools is running, regarding the distribution correlation table contained in codes variable.

**Return type** str

`src.architecture.get_distribution(codes)`

Gets the code for the distribution

**Parameters** **L{Mapping}** (*codes*) – The map containing distribution correlation table.

**Returns** The distribution on which salomeTools is running, regarding the distribution correlation table contained in codes variable.

**Return type** str

`src.architecture.get_nb_proc()`

**Gets the number of processors of the machine** on which salomeTools is running.

**Returns** the number of processors.

**Return type** str

`src.architecture.get_python_version()`

Gets the version of the running python.

**Returns** the version of the running python.

**Return type** str

`src.architecture.get_user()`

method that gets the username that launched sat

**Return type** str

`src.architecture.is_windows()`

method that checks windows OS

**Return type** boolean

## src.compilation module

**class** `src.compilation.Builder` (*config, logger, product\_info, options=OptResult( ), check\_src=True*)

Class to handle all construction steps, like cmake, configure, make, ...

**build\_configure** (*options=""*)

**check** (*command=""*)

**cmake** (*options=""*)

**complete\_environment** (*make\_options*)

**configure** (*options=""*)

**do\_batch\_script\_build** (*script, nb\_proc*)

```

do_default_build (build_conf_options="", configure_options="", show_warning=True)
do_python_script_build (script, nb_proc)
do_script_build (script, number_of_proc=0)
hack_libtool ()
install ()
log (text, level, showInfo=True)
log_command (command)
make (nb_proc, make_opt="")
prepare ()
put_txt_log_in_appli_log_dir (file_name)

```

Put the txt log (that contain the system logs, like make command output) in the directory <APPLICATION DIR>/LOGS/<product\_name>/

**Parameters** `Str (file_name)` – the name of the file to write

```
wmake (nb_proc, opt_nb_proc=None)
```

## src.debug module

This file assume DEBUG functionalities use. Print salomeTools debug messages in sys.stderr. Show pretty print debug representation from instances of SAT classes (pretty print src.pyconf.Config)

Warning: supposedly show messages in SAT development phase, not production

Usage:

```

>> import debug as DBG
>> DBG.write("aTitle", aVariable) # not shown in production
>> DBG.write("aTitle", aVariable, True) # unconditionally shown (as show=True)

```

to set show message as development phase:

```
>> DBG.push_debug(True)
```

to set no show message as production phase:

```
>> DBG.push_debug(False)
```

to set show message temporary as development phase, only in a method:

```

>> def aMethodToDebug(...):
>> DBG.push_debug(True) #force show as appended status
>> etc. method code with some DBG.write()
>> DBG.pop_debug() #restore previous status (show or not show)
>> return

```

to set a message for future fix, as temporary problem to not forget:

```
DBG.tofix("aTitle", aVariable, True/False) #True/False in production shown, or not
```

in command line interface you could redirect stderr to file 'myDebug.log':

```

>> sat compile ... 2> myDebug.log # only stderr
>> sat compile ... &> myDebug.log # stdout and stderr

```

```
class src.debug.InStream (buf="")
    Bases: StringIO.StringIO
    utility class for pyconf.Config input iostream

class src.debug.OutStream (buf="")
    Bases: StringIO.StringIO
    utility class for pyconf.Config output iostream

close ()
    because Config.__save__ calls close() stream as file keep value before lost as self.value

src.debug.format_color_exception (msg, limit=None, trace=None)
    Format a stack trace and the exception information. as traceback.format_exception(), with color with trace-
    back only if (_debug) or (DBG._user in DBG._developpers)

src.debug.format_exception (msg, limit=None, trace=None)
    Format a stack trace and the exception information. as traceback.format_exception(), without color with
    traceback only if (_debug) or (DBG._user in DBG._developpers)

src.debug.getLocalEnv ()
    get string for environment variables representation

src.debug.getStrConfigDbg (config)
    set string as saveConfigDbg, as (path expression evaluation) for debug

src.debug.getStrConfigStd (config)
    set string as saveConfigStd, as file .pyconf

src.debug.indent (text, amount=2, ch=' ')
    indent multi lines message

src.debug.isTypeConfig (var)
    To know if var is instance from Config/pyconf

src.debug.pop_debug ()
    restore previous debug outputs status

src.debug.push_debug (aBool)
    set debug outputs activated, or not

src.debug.saveConfigDbg (config, aStream, indent=0, path="")
    pyconf returns multilines (path expression evaluation) for debug

src.debug.saveConfigStd (config, aStream)
    returns as file .pyconf

src.debug.tofix (title, var="", force=None)
    write sys.stderr a message if _debug[-1]==True or optionally force=True use this only if no logger accessible
    for classic logger.warning(message)

src.debug.write (title, var="", force=None, fmt="\n#### DEBUG: %s:\n%s\n")
    write sys.stderr a message if _debug[-1]==True or optionally force=True
```

## src.environment module

```
class src.environment.Environ (environ=None)
    Class to manage the environment context

append (key, value, sep=':')
    Same as append_value but the value argument can be a list

    Parameters

- str (sep) – the environment variable to append
- str or list (value) – the value(s) to append to key

```



- **str** – the separator string

**append\_value** (*key*, *value*, *sep*=':')  
append value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** – the value to append to key
- **str** – the separator string

**command\_value** (*key*, *command*)

Get the value given by the system command “command” and put it in the environment variable key

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**is\_defined** (*key*)

Check if the key exists in the environment

**Parameters** **str** (*key*) – the environment variable to check

**prepend** (*key*, *value*, *sep*=':')

Same as prepend\_value but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str or list** (*value*) – the value(s) to prepend to key
- **str** – the separator string

**prepend\_value** (*key*, *value*, *sep*=':')

prepend value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str** – the value to prepend to key
- **str** – the separator string

**set** (*key*, *value*)

Set the environment variable “key” to value “value”

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** src.environment.**FileEnvWriter** (*config*, *logger*, *out\_dir*, *src\_root*, *env\_info*=None)

Class to dump the environment to a file.

**write\_cfgForPy\_file** (*filename*, *additional\_env*={}, *for\_package*=None,  
*with\_commercial*=True)

Append to current opened aFile a cfgForPy environment (SALOME python launcher).

**Parameters**

- **str** (*for\_package*) – the file path

- **dict** (*additional\_env*) – a dictionary of additional variables to add to the environment
- **str** – If not None, produce a relative environment designed for a package.

**write\_env\_file** (*filename, forBuild, shell, for\_package=None*)

Create an environment file.

**Parameters**

- **str** (*shell*) – the file path
- **bool** (*forBuild*) – if true, the build environment
- **str** – the type of file wanted (.sh, .bat)

**Returns** The path to the generated file

**Return type** str

**class** src.environment.**SalomeEnviron** (*cfg, environ, forBuild=False, for\_package=None, enable\_simple\_env\_script=True*)

Class to manage the environment of SALOME.

**add\_comment** (*comment*)

Add a commentary to the out stream (in case of file generation)

**Parameters** **str** (*comment*) – the commentary to add

**add\_line** (*nb\_line*)

Add empty lines to the out stream (in case of file generation)

**Parameters** **int** (*nb\_line*) – the number of empty lines to add

**add\_warning** (*warning*)

Add a warning to the out stream (in case of file generation)

**Parameters** **str** (*warning*) – the warning to add

**append** (*key, value, sep=':'*)

append value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** – the value to append to key
- **str** – the separator string

**dump** (*out*)

Write the environment to out

**Parameters** **file** (*out*) – the stream where to write the environment

**finish** (*required*)

Add a final instruction in the out file (in case of file generation)

**Parameters** **bool** (*required*) – Do nothing if required is False

**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**get\_names** (*lProducts*)

Get the products name to add in SALOME\_MODULES environment variable It is the name of the product, except in the case where the is a component name. And it has to be in SALOME\_MODULES variable only if the product has the property has\_salome\_hui = “yes”

**Parameters** **list** (*lProducts*) – List of products to potentially add

**is\_defined** (*key*)

Check if the key exists in the environment

**Parameters** **str** (*key*) – the environment variable to check

**load\_cfg\_environment** (*cfg\_env*)

Loads environment defined in *cfg\_env*

**Parameters** **Config** (*cfg\_env*) – A config containing an environment

**prepend** (*key, value, sep=':'*)

prepend value to key using *sep*

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str** – the value to prepend to key
- **str** – the separator string

**run\_env\_script** (*product\_info, logger=None, native=False*)

Runs an environment script.

**Parameters**

- **Config** (*product\_info*) – The product description
- **Logger** (*logger*) – The logger instance to display messages
- **Boolean** (*native*) – if True load *set\_native\_env* instead of *set\_env*

**run\_simple\_env\_script** (*script\_path, logger=None*)

Runs an environment script. Same as *run\_env\_script*, but with a script path as parameter.

**Parameters**

- **str** (*script\_path*) – a path to an environment script
- **Logger** (*logger*) – The logger instance to display messages

**set** (*key, value*)

Set the environment variable “key” to value “value”

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**set\_a\_product** (*product, logger*)

Sets the environment of a product.

**Parameters**

- **str** (*product*) – The product name
- **Logger** (*logger*) – The logger instance to display messages

**set\_application\_env** (*logger*)

Sets the environment defined in the APPLICATION file.

**Parameters** **Logger** (*logger*) – The logger instance to display messages

**set\_cpp\_env** (*product\_info*)

Sets the generic environment for a SALOME cpp product.

**Parameters** **Config** (*product\_info*) – The product description

**set\_full\_envIRON** (*logger, env\_info*)

Sets the full environment for products specified in *env\_info* dictionary.

**Parameters**

- **Logger** (*logger*) – The logger instance to display messages
- **list** (*env\_info*) – the list of products

**set\_products** (*logger, src\_root=None*)

Sets the environment for all the products.

#### Parameters

- **Logger** (*logger*) – The logger instance to display messages
- **src** (*src\_root*) – the application working directory

**set\_python\_libdirs** ()

Set some generic variables for python library paths

**set\_salome\_generic\_product\_env** (*pi*)

Sets the generic environment for a SALOME product.

**Parameters** **Config** (*pi*) – The product description

**set\_salome\_minimal\_product\_env** (*product\_info, logger*)

Sets the minimal environment for a SALOME product. `xxx_ROOT_DIR` and `xxx_SRC_DIR`

#### Parameters

- **Config** (*product\_info*) – The product description
- **Logger** (*logger*) – The logger instance to display messages

**class** `src.environment.Shell` (*name, extension*)

Definition of a Shell.

`src.environment.load_environment` (*config, build, logger*)

Loads the environment (used to run the tests, for example).

#### Parameters

- **Config** (*config*) – the global config
- **bool** (*build*) – build environment if True
- **Logger** (*logger*) – The logger instance to display messages

## src.fileEnviron module

**class** `src.fileEnviron.BashFileEnviron` (*output, environ=None*)

Bases: `src.fileEnviron.FileEnviron` (page 42)

Class for bash shell.

**command\_value** (*key, command*)

Get the value given by the system command “command” and put it in the environment variable key.

Has to be overwritten in the derived classes This can be seen as a virtual method

#### Parameters

- **str** (*command*) – the environment variable
- **str** – the command to execute

**finish** (*required=True*)

Add a final instruction in the out file (in case of file generation)

**Parameters** **bool** (*required*) – Do nothing if required is False

**set** (*key, value*)

Set the environment variable “key” to value “value”

#### Parameters

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** `src.fileEnviron.BatFileEnviron` (*output*, *environ=None*)

Bases: `src.fileEnviron.FileEnviron` (page 42)

for Windows batch shell.

**add\_comment** (*comment*)

Add a comment in the shell file

**Parameters** **str** (*comment*) – the comment to add

**command\_value** (*key*, *command*)

Get the value given by the system command “command” and put it in the environment variable key. Has to be overwritten in the derived classes This can be seen as a virtual method

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**finish** (*required=True*)

Add a final instruction in the out file (in case of file generation) In the particular windows case, do nothing

**Parameters** **bool** (*required*) – Do nothing if required is False

**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**set** (*key*, *value*)

Set the environment variable “key” to value “value”

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** `src.fileEnviron.ContextFileEnviron` (*output*, *environ=None*)

Bases: `src.fileEnviron.FileEnviron` (page 42)

Class for a salome context configuration file.

**add\_echo** (*text*)

Add a comment

**Parameters** **str** (*text*) – the comment to add

**add\_warning** (*warning*)

Add a warning

**Parameters** **str** (*text*) – the warning to add

**append\_value** (*key*, *value*, *sep=':'*)

append value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** – the value to append to key
- **str** – the separator string

**command\_value** (*key*, *command*)

Get the value given by the system command “command” and put it in the environment variable key. Has to be overwritten in the derived classes This can be seen as a virtual method

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**finish** (*required=True*)

Add a final instruction in the out file (in case of file generation)

**Parameters** **bool** (*required*) – Do nothing if required is False**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable**prepend\_value** (*key, value, sep=':'*)

prepend value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str** – the value to prepend to key
- **str** – the separator string

**set** (*key, value*)

Set the environment variable “key” to value “value”

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** `src.fileEnviron.FileEnviron` (*output, environ=None*)Bases: `object`

Base class for shell environment

**add\_comment** (*comment*)

Add a comment in the shell file

**Parameters** **str** (*comment*) – the comment to add**add\_echo** (*text*)

Add a “echo” in the shell file

**Parameters** **str** (*text*) – the text to echo**add\_line** (*number*)

Add some empty lines in the shell file

**Parameters** **int** (*number*) – the number of lines to add**add\_warning** (*warning*)

Add a warning “echo” in the shell file

**Parameters** **str** (*warning*) – the text to echo**append** (*key, value, sep=':'*)Same as `append_value` but the value argument can be a list**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** or **list** (*value*) – the value(s) to append to key
- **str** – the separator string

**append\_value** (*key, value, sep=':'*)

append value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** – the value to append to key
- **str** – the separator string

**command\_value** (*key, command*)

Get the value given by the system command “command” and put it in the environment variable key. Has to be overwritten in the derived classes This can be seen as a virtual method

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**finish** (*required=True*)

Add a final instruction in the out file (in case of file generation)

**Parameters** **bool** (*required*) – Do nothing if required is False

**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**is\_defined** (*key*)

Check if the key exists in the environment

**Parameters** **str** (*key*) – the environment variable to check

**prepend** (*key, value, sep=':'*)

Same as prepend\_value but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str or list** (*value*) – the value(s) to prepend to key
- **str** – the separator string

**prepend\_value** (*key, value, sep=':'*)

prepend value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str** – the value to prepend to key
- **str** – the separator string

**set** (*key, value*)

Set the environment variable ‘key’ to value ‘value’

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** `src.fileEnviron.LauncherFileEnviron` (*output, environ=None*)

Class to generate a launcher file script (in python syntax) SalomeContext API

**add** (*key, value*)

prepend value to key using sep

**Parameters**

- **str** (*value*) – the environment variable to prepend

- **str** – the value to prepend to key

**add\_comment** (*comment*)

**add\_echo** (*text*)

Add a comment

**Parameters** **str** (*text*) – the comment to add

**add\_line** (*number*)

Add some empty lines in the launcher file

**Parameters** **int** (*number*) – the number of lines to add

**add\_warning** (*warning*)

Add a warning

**Parameters** **str** (*text*) – the warning to add

**append** (*key, value, sep=':'*)

Same as `append_value` but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str or list** (*value*) – the value(s) to append to key
- **str** – the separator string

**append\_value** (*key, value, sep=':'*)

append value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** – the value to append to key
- **str** – the separator string

**change\_to\_launcher** (*value*)

**command\_value** (*key, command*)

Get the value given by the system command “command” and put it in the environment variable key.

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**finish** (*required=True*)

Add a final instruction in the out file (in case of file generation) In the particular launcher case, do nothing

**Parameters** **bool** (*required*) – Do nothing if required is False

**get** (*key*)

Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**is\_defined** (*key*)

Check if the key exists in the environment

**Parameters** **str** (*key*) – the environment variable to check

**prepend** (*key, value, sep=':'*)

Same as `prepend_value` but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to prepend



- **str** or **list** (*value*) – the value(s) to prepend to key
- **str** – the separator string

**prepend\_value** (*key, value, sep=','*)  
prepend value to key using sep

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str** – the value to prepend to key
- **str** – the separator string

**set** (*key, value*)  
Set the environment variable “key” to value “value”

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**class** `src.fileEnviron.ScreenEnviron` (*output, environ=None*)  
Bases: `src.fileEnviron.FileEnviron` (page 42)

**add\_comment** (*comment*)  
Add a comment in the shell file

**Parameters** **str** (*comment*) – the comment to add

**add\_echo** (*text*)  
Add a “echo” in the shell file

**Parameters** **str** (*text*) – the text to echo

**add\_line** (*number*)  
Add some empty lines in the shell file

**Parameters** **int** (*number*) – the number of lines to add

**add\_warning** (*warning*)  
Add a warning “echo” in the shell file

**Parameters** **str** (*warning*) – the text to echo

**append** (*name, value, sep=','*)  
Same as `append_value` but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to append
- **str** or **list** (*value*) – the value(s) to append to key
- **str** – the separator string

**command\_value** (*key, command*)  
Get the value given by the system command “command” and put it in the environment variable key.  
Has to be overwritten in the derived classes This can be seen as a virtual method

**Parameters**

- **str** (*command*) – the environment variable
- **str** – the command to execute

**get** (*name*)  
Get the value of the environment variable “key”

**Parameters** **str** (*key*) – the environment variable

**is\_defined** (*name*)

Check if the key exists in the environment

**Parameters** **str** (*key*) – the environment variable to check

**prepend** (*name, value, sep=':'*)

Same as `prepend_value` but the value argument can be a list

**Parameters**

- **str** (*sep*) – the environment variable to prepend
- **str or list** (*value*) – the value(s) to prepend to key
- **str** – the separator string

**run\_env\_script** (*module, script*)

**set** (*name, value*)

Set the environment variable 'key' to value 'value'

**Parameters**

- **str** (*value*) – the environment variable to set
- **str** – the value

**write** (*command, name, value, sign='='*)

`src.fileEnviron.get_file_environ` (*output, shell, environ=None*)

Instantiate correct FileEnvironment sub-class.

**Parameters**

- **file** (*output*) – the output file stream.
- **str** (*shell*) – the type of shell syntax to use.
- **dict** (*environ*) – a potential additional environment.

`src.fileEnviron.special_path_separator` (*name*)

TCLLIBPATH, TKLIBPATH, PV\_PLUGIN\_PATH environments variables need some exotic path separator. This function gives the separator regarding the name of the variable to append or prepend.

**Parameters** **str** (*name*) – The name of the variable to find the separator

## src.fork module

`src.fork.batch` (*cmd, logger, cwd, args=[], log=None, delai=20, sommeil=1*)

`src.fork.batch_salome` (*cmd, logger, cwd, args, getTmpDir, pendant='SALOME\_Session\_Server',  
fn='killSalome.py', log=None, delai=20, sommeil=1, delaiapp=0*)

`src.fork.launch_command` (*cmd, logger, cwd, args=[], log=None*)

`src.fork.show_progress` (*logger, top, delai, ss=""*)

shortcut function to display the progression

**Parameters**

- **Logger** (*logger*) – The logging instance
- **int** (*delai*) – the number to display
- **int** – the number max
- **str** (*ss*) – the string to display

`src.fork.write_back` (*logger, message, level*)

shortcut function to write at the begin of the line

**Parameters**

- **Logger** (*logger*) – The logging instance
- **str** (*message*) – the text to display
- **int** (*level*) – the level of verbosity

## src.logger module

Implements the classes and method relative to the logging

**class** `src.logger.Logger` (*config=None, silent\_sysstd=False, all\_in\_terminal=False, micro\_command=False*)

Bases: `object`

Class to handle log mechanism.

**add\_link** (*log\_file\_name, command\_name, command\_res, full\_launched\_command*)

Add a link to another log file.

### Parameters

- **str** (*command\_res*) – The file name of the link.
- **str** – The name of the command linked.
- **str** – The result of the command linked. “0” or “1”

**Parma full\_launched\_command str** The full lanch command (“sat command ...”)

**end\_write** (*attribute*)

Called just after command end: Put all fields corresponding to the command end context (time). Write the log xml file on the hard drive. And display the command to launch to get the log

**Parameters dict** (*attribute*) – the attribute to add to the node “Site”.

**error** (*message*)

Print an error.

**Parameters str** (*message*) – The message to print.

**flush** ()

Flush terminal

**put\_initial\_xml\_fields** ()

Called at class initialization: Put all fields corresponding to the command context (user, time, ...)

**write** (*message, level=None, screenOnly=False*)

function used in the commands to print in the terminal and the log file.

### Parameters

- **str** (*message*) – The message to print.
- **int** (*level*) – The output level corresponding to the message  $0 < level < 6$ .
- **boolean** (*screenOnly*) – if True, do not write in log file.

**src.logger.date\_to\_datetime** (*date*)

From a string date in format YYYYMMDD\_HHMMSS returns list year, mon, day, hour, minutes, seconds

**Parameters str** (*date*) – The date in format YYYYMMDD\_HHMMSS

**Returns** the same date and time in separate variables.

**Return type** (str,str,str,str,str,str)

**src.logger.getCurrentLogger** ()

get current logging logger, set as DefaultLogger if not set yet

**src.logger.getDefaultLogger** ()

get simple logging logger DefaultLogger, set it as current

```
src.logger.getUnittestLogger ()  
    get simple logging logger UnittestLogger, set it as current
```

```
src.logger.isCurrentLoggerUnittest ()
```

```
src.logger.list_log_file (dirPath, expression)  
    Find all files corresponding to expression in dirPath
```

**Parameters**

- **str** (*expression*) – the directory where to search the files
- **str** – the regular expression of files to find

**Returns** the list of files path and informations about it**Return type** list

```
src.logger.sendMessageToCurrentLogger (message, level)  
    assume relay from obsolescent logger.write(msg, 1/2/3...) to future logging.critical/warning/info...(msg)  
    (as logging package tips)
```

```
src.logger.setCurrentLogger (logger)  
    temporary send all in stdout as simple logging logger
```

```
src.logger.show_command_log (logFilePath, cmd, application, notShownCommands)  
    Used in updateHatXml. Determine if the log xml file logFilePath has to be shown or not in the hat log.
```

**Parameters**

- **str** (*application*) – the path to the command xml log file
- **str** – the command of the log file
- **str** – the application passed as parameter to the salomeTools command
- **list** (*notShownCommands*) – the list of commands that are not shown by default

**Returns** True if cmd is not in notShownCommands and the application in the log file corresponds to application**Return type** boolean

```
src.logger.timedelta_total_seconds (timedelta)  
    Replace total_seconds from datetime module in order to be compatible with old python versions
```

**Parameters** **datetime.timedelta** (*timedelta*) – The delta between two dates**Returns** The number of seconds corresponding to timedelta.**Return type** float

```
src.logger.update_hat_xml (logDir, application=None, notShownCommands=[])  
    Create the xml file in logDir that contain all the xml file and have a name like YYYYMM-  
    MDD_HHMMSS_namecmd.xml
```

**Parameters**

- **str** (*application*) – the directory to parse
- **str** – the name of the application if there is any

## src.options module

The Options class that manages the access to all options passed as parameters in salomeTools command lines

```
class src.options.OptResult  
    Bases: object
```

An instance of this class will be the object manipulated in code of all salomeTools commands The aim of this class is to have an elegant syntax to manipulate the options.

Example:

```
>> options, remainderArgs = command.parseArguments(args)
>> print(options.output_verbose_level)
>> 'INFO'
```

**class** `src.options.Options`

Bases: `object`

Class to manage all salomeTools options

**add\_option** (*shortName, longName, optionType, destName, helpString="", default=None*)

Add an option to a command. It gets all attributes of an option and append it in the options field

**Parameters**

- **shortName** – (str) The short name of the option (as ‘-l’ for level option).
- **longName** – (str) The long name of the option (as ‘-level’ for level option).
- **optionType** – (str) The type of the option (ex “int”).
- **destName** – (str) The name that will be used in the code.
- **helpString** – (str) The text to display when user ask for help on a command.

**Returns** None

**debug\_write** ()

**filterLevel** (*aLevel*)

filter level logging values

**filterList2** (*aStr*)

filter a list as ‘KERNEL,YACS,etc.’

**getDetailOption** (*option*)

for convenience

**Returns** (tuple) 4-elements (shortName, longName, optionType, helpString)

**get\_help** ()

Returns all options stored in self.options as help message colored string

**Returns** (str) colored string

**indent** (*text, amount, car=' '*)

indent multi lines message

**parse\_args** (*argList=None*)

Instantiates the class OptResult that gives access to all options in the code

**Parameters** **argList** – (list) the raw list of arguments that were passed

**Returns** (OptResult, list) as (optResult, args) optResult is the option instance to manipulate in the code. args is the full raw list of passed options

## src.printcolors module

In this file is stored the mechanism that manage color prints in the terminal

`src.printcolors.cleancolor` (*message*)

remove color from a colored text.

**Parameters** **str** (*message*) – The text to be cleaned.

**Returns** The cleaned text.

**Return type** str

`src.printcolors.print_color_map()`

This method prints the color map

`src.printcolors.print_color_range(start, end)`

print possible range values for colors

#### Parameters

- **int** (*end*) – The smaller value.
- **int** – The bigger value.

`src.printcolors.print_value(logger, label, value, level=1, suffix="")`

shortcut method to print a label and a value with the info color

#### Parameters

- **class logger** (*logger*) – the logger instance.
- **int** (*level*) – the label to print.
- **str** (*suffix*) – the value to print.
- **int** – the level of verboseness.
- **str** – the suffix to add at the end.

`src.printcolors.printc(txt, code="")`

print a text with colors

#### Parameters

- **str** (*code*) – The text to be printed.
- **str** – The color to use.

**Returns** The colored text.

**Return type** str

`src.printcolors.printcError(txt)`

print a text error color

**Parameters** **str** (*txt*) – The text to be printed.

**Returns** The colored text.

**Return type** str

`src.printcolors.printcHeader(txt)`

print a text header color

**Parameters** **str** (*txt*) – The text to be printed.

**Returns** The colored text.

**Return type** str

`src.printcolors.printcHighlight(txt)`

print a text highlight color

**Parameters** **str** (*txt*) – The text to be printed.

**Returns** The colored text.

**Return type** str

`src.printcolors.printcInfo(txt)`

print a text info color

**Parameters** **str** (*txt*) – The text to be printed.

**Returns** The colored text.

**Return type** str

`src.printcolors.printcLabel (txt)`  
print a text label color

**Parameters** `str (txt)` – The text to be printed.

**Returns** The colored text.

**Return type** `str`

`src.printcolors.printcSuccess (txt)`  
print a text success color

**Parameters** `str (txt)` – The text to be printed.

**Returns** The colored text.

**Return type** `str`

`src.printcolors.printcWarning (txt)`  
print a text warning color

**Parameters** `str (txt)` – The text to be printed.

**Returns** The colored text.

**Return type** `str`

## src.product module

In this file are implemented the methods relative to the product notion of salomeTools

`src.product.check_config_exists (config, prod_dir, prod_info)`

Verify that the installation directory of a product in a base exists Check all the config-<i> directory and verify the sat-config.pyconf file that is in it

**Parameters**

- **Config** (`product_info`) – The global configuration
- **str** (`prod_dir`) – The product installation directory path (without config-<i>)
- **Config** – The configuration specific to the product

**Returns** True or false is the installation is found or not and if it is found, the path of the found installation

**Return type** (`boolean, str`)

`src.product.check_installation (product_info)`

Verify if a product is well installed. Checks install directory presence and some additional files if it is defined in the config

**Parameters** **Config** (`product_info`) – The configuration specific to the product

**Returns** True if it is well installed

**Return type** `boolean`

`src.product.check_source (product_info)`

Verify if a sources of product is preset. Checks source directory presence

**Parameters** **Config** (`product_info`) – The configuration specific to the product

**Returns** True if it is well installed

**Return type** `boolean`

`src.product.get_base_install_dir (config, prod_info, version)`

Compute the installation directory of a product in base

**Parameters**

- **Config** (*product\_info*) – The global configuration
- **Config** – The configuration specific to the product
- **str** (*version*) – The version of the product

**Returns** The path of the product installation

**Return type** str

`src.product.get_install_dir (config, base, version, prod_info)`

Compute the installation directory of a given product

**Parameters**

- **Config** (*product\_info*) – The global configuration
- **str** (*version*) – This corresponds to the value given by user in its application.pyconf for the specific product. If “yes”, the user wants the product to be in base. If “no”, he wants the product to be in the application workdir
- **str** – The version of the product
- **Config** – The configuration specific to the product

**Returns** The path of the product installation

**Return type** str

`src.product.get_product_components (product_info)`

Get the component list to generate with the product

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** The list of names of the components

**Return type** List

`src.product.get_product_config (config, product_name, with_install_dir=True)`

Get the specific configuration of a product from the global configuration

**Parameters**

- **Config** (*config*) – The global configuration
- **str** (*product\_name*) – The name of the product
- **boolean** (*with\_install\_dir*) – If false, do not provide an install directory (at false only for internal use of the function `check_config_exists`)

**Returns** the specific configuration of the product

**Return type** *Config* (page 56)

`src.product.get_product_dependencies (config, product_info)`

Get recursively the list of products that are in the product\_info dependencies

**Parameters**

- **Config** (*product\_info*) – The global configuration
- **Config** – The configuration specific to the product

**Returns** the list of products in dependence

**Return type** list

`src.product.get_product_section (config, product_name, version, section=None)`

Get the product description from the configuration

**Parameters**

- **Config** (*config*) – The global configuration
- **str** (*section*) – The product name



- **str** – The version of the product
- **str** – The searched section (if not None, the section is explicitly given)

**Returns** The product description

**Return type** *Config* (page 56)

`src.product.get_products_infos` (*lproducts*, *config*)

Get the specific configuration of a list of products

**Parameters**

- **List** (*lproducts*) – The list of product names
- **Config** (*config*) – The global configuration

**Returns** the list of tuples (product name, specific configuration of the product)

**Return type** [(str, *Config* (page 56))]

`src.product.product_compiles` (*product\_info*)

Know if a product compiles or not (some products do not have a compilation procedure)

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product compiles, else False

**Return type** boolean

`src.product.product_has_env_script` (*product\_info*)

Know if a product has an environment script

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product it has an environment script, else False

**Return type** boolean

`src.product.product_has_logo` (*product\_info*)

Know if a product has a logo (YACSGEN generate)

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** The path of the logo if the product has a logo, else False

**Return type** Str

`src.product.product_has_patches` (*product\_info*)

Know if a product has one or more patches

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product has one or more patches

**Return type** boolean

`src.product.product_has_salome_gui` (*product\_info*)

Know if a product has a SALOME gui

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product has a SALOME gui, else False

**Return type** Boolean

`src.product.product_has_script` (*product\_info*)

Know if a product has a compilation script

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product it has a compilation script, else False

**Return type** boolean

`src.product.product_is_autotools` (*product\_info*)

Know if a product is compiled using the autotools

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is autotools, else False

**Return type** boolean

`src.product.product_is_cmake` (*product\_info*)

Know if a product is compiled using the cmake

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is cmake, else False

**Return type** boolean

`src.product.product_is_cpp` (*product\_info*)

Know if a product is cpp

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is a cpp, else False

**Return type** boolean

`src.product.product_is_debug` (*product\_info*)

Know if a product is in debug mode

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is in debug mode, else False

**Return type** boolean

`src.product.product_is_dev` (*product\_info*)

Know if a product is in dev mode

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is in dev mode, else False

**Return type** boolean

`src.product.product_is_fixed` (*product\_info*)

Know if a product is fixed

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is fixed, else False

**Return type** boolean

`src.product.product_is_generated` (*product\_info*)

Know if a product is generated (YACSGEN)

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is generated

**Return type** boolean

`src.product.product_is_mpi` (*product\_info*)

Know if a product has openmpi in its dependencies

**Parameters** **Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product has openmpi inits dependencies

**Return type** boolean

`src.product.product_is_native` (*product\_info*)

Know if a product is native

**Parameters Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is native, else False

**Return type** boolean

`src.product.product_is_salome` (*product\_info*)

Know if a product is a SALOME module

**Parameters Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is a SALOME module, else False

**Return type** boolean

`src.product.product_is_smesh_plugin` (*product\_info*)

Know if a product is a SMESH plugin

**Parameters Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is a SMESH plugin, else False

**Return type** boolean

`src.product.product_is_vcs` (*product\_info*)

Know if a product is download using git, svn or cvs (not archive)

**Parameters Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is vcs, else False

**Return type** boolean

`src.product.product_is_verbose` (*product\_info*)

Know if a product is in verbose mode

**Parameters Config** (*product\_info*) – The configuration specific to the product

**Returns** True if the product is in verbose mode, else False

**Return type** boolean

## src.pyconf module

This is a configuration module for Python.

This module should work under Python versions  $\geq 2.2$ , and cannot be used with earlier versions since it uses new-style classes.

Development and testing has only been carried out (so far) on Python 2.3.4 and Python 2.4.2. See the test module (`test_config.py`) included in the U{distribution<[http://www.red-dove.com/python\\_config.html#blank](http://www.red-dove.com/python_config.html#blank)>} (follow the download link).

A simple example - with the example configuration file:

```
messages:
[
  {
    stream : `sys.stderr`
    message: 'Welcome'
    name: 'Harry'
  }
  {
    stream : `sys.stdout`
    message: 'Welkom'
    name: 'Ruud'
  }
  {
```

(continues on next page)

(continued from previous page)

```

    stream : $messages[0].stream
    message: 'Bienvenue'
    name: Yves
  }
]

```

a program to read the configuration would be:

```

from config import Config

f = file('simple.cfg')
cfg = Config(f)
for m in cfg.messages:
    s = '%s, %s' % (m.message, m.name)
    try:
        print >> m.stream, s
    except IOError, e:
        print e

```

which, when run, would yield the console output:

```

Welcome, Harry
Welkom, Ruud
Bienvenue, Yves

```

See U{this tutorial<[http://www.red-dove.com/python\\_config.html](http://www.red-dove.com/python_config.html)>} for more information.

#modified for salomeTools @version: 0.3.7.1

@author: Vinay Sajip

@copyright: Copyright (C) 2004-2007 Vinay Sajip. All Rights Reserved.

@var streamOpener: The default stream opener. This is a factory function which takes a string (e.g. filename) and returns a stream suitable for reading. If unable to open the stream, an IOError exception should be thrown.

The default value of this variable is L{defaultStreamOpener}. For an example of how it's used, see test\_config.py (search for streamOpener).

**class** `src.pyconf.Config` (*streamOrFile=None, parent=None, PWD=None*)  
 Bases: `src.pyconf.Mapping` (page 60)

This class represents a configuration, and is the only one which clients need to interface to, under normal circumstances.

**class** `Namespace`

Bases: `object`

This internal class is used for implementing default namespaces.

An instance acts as a namespace.

**addNamespace** (*ns, name=None*)

Add a namespace to this configuration which can be used to evaluate (resolve) dotted-identifier expressions. @param ns: The namespace to be added. @type ns: A module or other namespace suitable for passing as an argument to vars(). @param name: A name for the namespace, which, if specified, provides an additional level of indirection. @type name: str

**getByPath** (*path*)

Obtain a value in the configuration via its path. @param path: The path of the required value @type path: str @return the value at the specified path. @rtype: any @raise ConfigError: If the path is invalid

**load** (*stream*)

Load the configuration from the specified stream. Multiple streams can be used to populate the same instance, as long as there are no clashing keys. The stream is closed. @param stream: A stream

from which the configuration is read. @type stream: A read-only stream (file-like object). @raise ConfigError: if keys in the loaded configuration clash with existing keys. @raise ConfigFormatError: if there is a syntax error in the stream.

**removeNamespace** (*ns, name=None*)

Remove a namespace added with L{addNamespace}. @param ns: The namespace to be removed. @param name: The name which was specified when L{addNamespace} was called. @type name: str

**exception** `src.pyconf.ConfigError`

Bases: `exceptions.Exception`

This is the base class of exceptions raised by this module.

**exception** `src.pyconf.ConfigFormatError`

Bases: `src.pyconf.ConfigError` (page 57)

This is the base class of exceptions raised due to syntax errors in configurations.

**class** `src.pyconf.ConfigInputStream` (*stream*)

Bases: `object`

An input stream which can read either ANSI files with default encoding or Unicode files with BOMs.

Handles UTF-8, UTF-16LE, UTF-16BE. Could handle UTF-32 if Python had built-in support.

**close** ()

**read** (*size*)

**readline** ()

**class** `src.pyconf.ConfigList`

Bases: `list`

This class implements an ordered list of configurations and allows you to try getting the configuration from each entry in turn, returning the first successfully obtained value.

**getByPath** (*path*)

Obtain a value from the first configuration in the list which defines it.

@param path: The path of the value to retrieve. @type path: str @return: The value from the earliest configuration in the list which defines it. @rtype: any @raise ConfigError: If no configuration in the list has an entry with the specified path.

**class** `src.pyconf.ConfigMerger` (*resolver=<function defaultMergeResolve>*)

Bases: `object`

This class is used for merging two configurations. If a key exists in the merge operand but not the merge target, then the entry is copied from the merge operand to the merge target. If a key exists in both configurations, then a resolver (a callable) is called to decide how to handle the conflict.

**handleMismatch** (*obj1, obj2*)

Handle a mismatch between two objects.

@param obj1: The object to merge into. @type obj1: any @param obj2: The object to merge. @type obj2: any

**merge** (*merged, mergee*)

Merge two configurations. The second configuration is unchanged, and the first is changed to reflect the results of the merge.

@param merged: The configuration to merge into. @type merged: L{Config}. @param mergee: The configuration to merge. @type mergee: L{Config}.

**mergeMapping** (*map1, map2*)

Merge two mappings recursively. The second mapping is unchanged, and the first is changed to reflect the results of the merge.

@param map1: The mapping to merge into. @type map1: L{Mapping}. @param map2: The mapping to merge. @type map2: L{Mapping}.

**mergeSequence** (*seq1, seq2*)

Merge two sequences. The second sequence is unchanged, and the first is changed to have the elements of the second appended to it.

@param seq1: The sequence to merge into. @type seq1: L{Sequence}. @param seq2: The sequence to merge. @type seq2: L{Sequence}.

**overwriteKeys** (*map1, seq2*)

Renint variables. The second mapping is unchanged, and the first is changed depending the keys of the second mapping. @param map1: The mapping to reinit keys into. @type map1: L{Mapping}. @param map2: The mapping container reinit information. @type map2: L{Mapping}.

**class** `src.pyconf.ConfigOutputStream` (*stream, encoding=None*)

Bases: object

An output stream which can write either ANSI files with default encoding or Unicode files with BOMs.

Handles UTF-8, UTF-16LE, UTF-16BE. Could handle UTF-32 if Python had built-in support.

**close** ()

**flush** ()

**write** (*data*)

**class** `src.pyconf.ConfigReader` (*config*)

Bases: object

This internal class implements a parser for configurations.

**getChar** ()

Get the next char from the stream. Update line and column numbers appropriately.

@return: The next character from the stream. @rtype: str

**getToken** ()

Get a token from the stream. String values are returned in a form where you need to eval() the returned value to get the actual string. The return value is (token\_type, token\_value).

Multiline string tokenizing is thanks to David Janes (BlogMatrix)

@return: The next token. @rtype: A token tuple.

**load** (*stream, parent=None, suffix=None*)

Load the configuration from the specified stream.

@param stream: A stream from which to load the configuration. @type stream: A stream (file-like object). @param parent: The parent of the configuration (to which this reader belongs) in the hierarchy. Specified when the configuration is included in another one. @type parent: A L{Container} instance. @param suffix: The suffix of this configuration in the parent configuration. Should be specified whenever the parent is not None. @raise ConfigError: If parent is specified but suffix is not. @raise ConfigFormatError: If there are syntax errors in the stream.

**location** ()

Return the current location (filename, line, column) in the stream as a string.

Used when printing error messages,

@return: A string representing a location in the stream being read. @rtype: str

**match** (*t*)

Ensure that the current token type matches the specified value, and advance to the next token.

@param t: The token type to match. @type t: A valid token type. @return: The token which was last read from the stream before this function is called. @rtype: a token tuple - see L{getToken}. @raise ConfigFormatError: If the token does not match what's expected.

**parseFactor** ()

Parse a factor in an multiplicative expression (a \* b, a / b, a % b)

@return: the parsed factor @rtype: any scalar @raise ConfigFormatError: if a syntax error is found.

**parseKeyValuePair** (*parent*)

Parse a key-value pair, and add it to the provided L{Mapping}.

@param parent: The mapping to add entries to. @type parent: A L{Mapping} instance. @raise ConfigFormatError: if a syntax error is found.

**parseMapping** (*parent, suffix*)

Parse a mapping.

@param parent: The container to which the mapping will be added. @type parent: A L{Container} instance. @param suffix: The suffix for the value. @type suffix: str @return: a L{Mapping} instance representing the mapping. @rtype: L{Mapping} @raise ConfigFormatError: if a syntax error is found.

**parseMappingBody** (*parent*)

Parse the internals of a mapping, and add entries to the provided L{Mapping}.

@param parent: The mapping to add entries to. @type parent: A L{Mapping} instance.

**parseReference** (*type*)

Parse a reference.

@return: the parsed reference @rtype: L{Reference} @raise ConfigFormatError: if a syntax error is found.

**parseScalar** ()

Parse a scalar - a terminal value such as a string or number, or an L{Expression} or L{Reference}.

@return: the parsed scalar @rtype: any scalar @raise ConfigFormatError: if a syntax error is found.

**parseSequence** (*parent, suffix*)

Parse a sequence.

@param parent: The container to which the sequence will be added. @type parent: A L{Container} instance. @param suffix: The suffix for the value. @type suffix: str @return: a L{Sequence} instance representing the sequence. @rtype: L{Sequence} @raise ConfigFormatError: if a syntax error is found.

**parseSuffix** (*ref*)

Parse a reference suffix.

@param ref: The reference of which this suffix is a part. @type ref: L{Reference}. @raise ConfigFormatError: if a syntax error is found.

**parseTerm** ()

Parse a term in an additive expression (a + b, a - b)

@return: the parsed term @rtype: any scalar @raise ConfigFormatError: if a syntax error is found.

**parseValue** (*parent, suffix*)

Parse a value.

@param parent: The container to which the value will be added. @type parent: A L{Container} instance. @param suffix: The suffix for the value. @type suffix: str @return: The value @rtype: any @raise ConfigFormatError: if a syntax error is found.

**setStream** (*stream*)

Set the stream to the specified value, and prepare to read from it.

@param stream: A stream from which to load the configuration. @type stream: A stream (file-like object).

**exception** `src.pyconf.ConfigResolutionError`

Bases: `src.pyconf.ConfigError` (page 57)

This is the base class of exceptions raised due to semantic errors in configurations.

**class** `src.pyconf.Container` (*parent*)

Bases: `object`

This internal class is the base class for mappings and sequences.

@ivar `path`: A string which describes how to get to this instance from the root of the hierarchy.

Example:

```
a.list.of[1].or['more'].elements
```

**evaluate** (*item*)

Evaluate items which are instances of `L{Reference}` or `L{Expression}`.

`L{Reference}` instances are evaluated using `L{Reference.resolve}`, and `L{Expression}` instances are evaluated using `L{Expression.evaluate}`.

@param `item`: The item to be evaluated. @type `item`: any @return: If the item is an instance of `L{Reference}` or `L{Expression}`, the evaluated value is returned, otherwise the item is returned unchanged.

**setPath** (*path*)

Set the path for this instance. @param `path`: The path - a string which describes how to get to this instance from the root of the hierarchy. @type `path`: `str`

**writeToStream** (*stream, indent, container*)

Write this instance to a stream at the specified indentation level.

Should be redefined in subclasses.

@param `stream`: The stream to write to @type `stream`: A writable stream (file-like object) @param `indent`: The indentation level @type `indent`: `int` @param `container`: The container of this instance @type `container`: `L{Container}` @raise `NotImplementedError`: If a subclass does not override this

**writeValue** (*value, stream, indent*)

**class** `src.pyconf.Expression` (*op, lhs, rhs*)

Bases: `object`

This internal class implements a value which is obtained by evaluating an expression.

**evaluate** (*container*)

Evaluate this instance in the context of a container.

@param `container`: The container to evaluate in from. @type `container`: `L{Container}` @return: The evaluated value. @rtype: any @raise `ConfigResolutionError`: If evaluation fails. @raise `ZeroDivideError`: If division by zero occurs. @raise `TypeError`: If the operation is invalid, e.g. subtracting one string from another.

**class** `src.pyconf.Mapping` (*parent=None*)

Bases: `src.pyconf.Container` (page 59)

This internal class implements key-value mappings in configurations.

**addMapping** (*key, value, comment, setting=False*)

Add a key-value mapping with a comment.

@param `key`: The key for the mapping. @type `key`: `str` @param `value`: The value for the mapping. @type `value`: any @param `comment`: The comment for the key (can be `None`). @type `comment`: `str` @param `setting`: If `True`, ignore clashes. This is set to `true` when called from `L{__setattr__}`. @raise `ConfigFormatError`: If an existing key is seen again and `setting` is `False`.

**get** (*key, default=None*)

Allows a dictionary-style get operation.

**iteritems** ()

**iterkeys** ()



**keys** ()

Return the keys in a similar way to a dictionary.

**writeToStream** (*stream, indent, container*)

Write this instance to a stream at the specified indentation level.

Should be redefined in subclasses.

@param stream: The stream to write to @type stream: A writable stream (file-like object) @param indent: The indentation level @type indent: int @param container: The container of this instance @type container: L{Container}

**class** `src.pyconf.Reference` (*config, type, ident*)

Bases: object

This internal class implements a value which is a reference to another value.

**addElement** (*type, ident*)

Add an element to the reference.

@param type: The type of reference. @type type: BACKTICK or DOLLAR @param ident: The identifier which continues the reference. @type ident: str

**findConfig** (*container*)

Find the closest enclosing configuration to the specified container.

@param container: The container to start from. @type container: L{Container} @return: The closest enclosing configuration, or None. @rtype: L{Config}

**resolve** (*container*)

Resolve this instance in the context of a container.

@param container: The container to resolve from. @type container: L{Container} @return: The resolved value. @rtype: any @raise ConfigResolutionError: If resolution fails.

**class** `src.pyconf.Sequence` (*parent=None*)

Bases: `src.pyconf.Container` (page 59)

This internal class implements a value which is a sequence of other values.

**class** `SeqIter` (*seq*)

Bases: object

This internal class implements an iterator for a L{Sequence} instance.

**next** ()

**append** (*item, comment*)

Add an item to the sequence.

@param item: The item to add. @type item: any @param comment: A comment for the item. @type comment: str

**writeToStream** (*stream, indent, container*)

Write this instance to a stream at the specified indentation level.

Should be redefined in subclasses.

@param stream: The stream to write to @type stream: A writable stream (file-like object) @param indent: The indentation level @type indent: int @param container: The container of this instance @type container: L{Container}

`src.pyconf.deepCopyMapping` (*inMapping*)

`src.pyconf.defaultMergeResolve` (*map1, map2, key*)

A default resolver for merge conflicts. Returns a string indicating what action to take to resolve the conflict.

@param map1: The map being merged into. @type map1: L{Mapping}. @param map2: The map being used as the merge operand. @type map2: L{Mapping}. @param key: The key in map2 (which also exists in map1). @type key: str

**@return:** One of “merge”, “append”, “mismatch” or “overwrite” indicating what action should be taken. This should be appropriate to the objects being merged - e.g. there is no point returning “merge” if the two objects are instances of L{Sequence}.

@rtype: str

`src.pyconf.defaultStreamOpener` (*name*)

This function returns a read-only stream, given its name. The name passed in should correspond to an existing stream, otherwise an exception will be raised.

This is the default value of L{streamOpener}; assign your own callable to streamOpener to return streams based on names. For example, you could use urllib2.urlopen().

@param name: The name of a stream, most commonly a file name. @type name: str @return: A stream with the specified name. @rtype: A read-only stream (file-like object)

`src.pyconf.isWord` (*s*)

See if a passed-in value is an identifier. If the value passed in is not a string, False is returned. An identifier consists of alphanumeric or underscore characters.

Examples:

```
isWord('a word') ->False
isWord('award') -> True
isWord(9) -> False
isWord('a_b_c_') ->True
```

@note: isWord(‘9abc’) will return True - not exactly correct, but adequate for the way it’s used here.

@param s: The name to be tested @type s: any @return: True if a word, else False @rtype: bool

`src.pyconf.makePath` (*prefix, suffix*)

Make a path from a prefix and suffix.

Examples: makePath(‘’, ‘suffix’) -> ‘suffix’ makePath(‘prefix’, ‘suffix’) -> ‘prefix.suffix’  
makePath(‘prefix’, ‘[1]’) -> ‘prefix[1]’

@param prefix: The prefix to use. If it evaluates as false, the suffix is returned. @type prefix: str @param suffix: The suffix to use. It is either an identifier or an index in brackets. @type suffix: str @return: The path concatenation of prefix and suffix, with adot if the suffix is not a bracketed index. @rtype: str

`src.pyconf.overwriteMergeResolve` (*map1, map2, key*)

An overwriting resolver for merge conflicts. Calls L{defaultMergeResolve}, but where a “mismatch” is detected, returns “overwrite” instead.

@param map1: The map being merged into. @type map1: L{Mapping}. @param map2: The map being used as the merge operand. @type map2: L{Mapping}. @param key: The key in map2 (which also exists in map1). @type key: str

## src.system module

In this file : all functions that do a system call, like open a browser or an editor, or call a git command

`src.system.archive_extract` (*from\_what, where, logger*)

Extracts sources from an archive.

### Parameters

- **str** (*where*) – The path to the archive.
- **str** – The path where to extract.
- **Logger** (*logger*) – The logger instance to use.

**Returns** True if the extraction is successful

**Return type** boolean

`src.system.cvs_extract` (*protocol, user, server, base, tag, product, where, logger, checkout=False, environment=None*)

Extracts sources from a cvs repository.

**Parameters**

- **str** (*where*) – The cvs protocol.
- **str** – The user to be used.
- **str** – The remote cvs server.
- **str** – .
- **str** – The tag.
- **str** – The product.
- **str** – The path where to extract.
- **Logger** (*logger*) – The logger instance to use.
- **boolean** (*checkout*) – If true use checkout cvs.
- **src.environment.Environment** (*environment*) – The environment to source when extracting.

**Returns** True if the extraction is successful

**Return type** boolean

`src.system.git_extract` (*from\_what, tag, where, logger, environment=None*)

Extracts sources from a git repository.

**Parameters**

- **str** (*where*) – The remote git repository.
- **str** – The tag.
- **str** – The path where to extract.
- **Logger** (*logger*) – The logger instance to use.
- **src.environment.Environment** (*environment*) – The environment to source when extracting.

**Returns** True if the extraction is successful

**Return type** boolean

`src.system.show_in_editor` (*editor, filePath, logger*)

open filePath using editor.

**Parameters**

- **str** (*filePath*) – The editor to use.
- **str** – The path to the file to open.

`src.system.svn_extract` (*user, from\_what, tag, where, logger, checkout=False, environment=None*)

Extracts sources from a svn repository.

**Parameters**

- **str** (*where*) – The user to be used.
- **str** – The remote git repository.
- **str** – The tag.
- **str** – The path where to extract.
- **Logger** (*logger*) – The logger instance to use.

- **boolean** (*checkout*) – If true use checkout svn.
- **src.environment. Environ** (*environment*) – The environment to source when extracting.

**Returns** True if the extraction is successful

**Return type** boolean

### src.template module

```
class src.template.MyTemplate (template)
    Bases: string.Template

    delimiter = '\xc2\xa4'

    pattern = <_sre.SRE_Pattern object>

src.template.substitute (template_file, subst_dic)
```

### src.test\_module module

```
class src.test_module.Test (config, logger, tmp_working_dir, testbase="", grids=None, sessions=None, launcher="", show_desktop=True)

    generate_launching_commands ()
    generate_script (listTest, script_path, ignoreList)
    get_test_timeout (test_name, default_value)
    get_tmp_dir ()
    prepare_testbase (test_base_name)
    prepare_testbase_from_dir (testbase_name, testbase_dir)
    prepare_testbase_from_git (testbase_name, testbase_base, testbase_tag)
    prepare_testbase_from_svn (user, testbase_name, testbase_base)
    read_results (listTest, has_timed_out)
    run_all_tests ()
    run_grid_tests ()
    run_script (script_name)
    run_session_tests ()
    run_testbase_tests ()
    run_tests (listTest, ignoreList)
    search_known_errors (status, test_grid, test_session, test)
    write_test_margin (tab)

src.test_module.getTmpDirDEFAULT ()
```

### src.xmlManager module

```
class src.xmlManager.ReadXmlFile (filePath)
    Bases: object

    Class to manage reading of an xml log file
```

**getRootAttrib ()**

Get the attributes of the self.xmlroot

**Returns** The attributes of the root node

**Return type** dict

**get\_attrib (node\_name)**

Get the attributes of the node node\_name in self.xmlroot

**Parameters** **str** (node\_name) – the name of the node

**Returns** the attributes of the node node\_name in self.xmlroot

**Return type** dict

**get\_node\_text (node)**

Get the text of the first node that has name that corresponds to the parameter node

**Parameters** **str** (node) – the name of the node from which get the text

**Returns** the text of the first node that has name that corresponds to the parameter node

**Return type** str

**class** src.xmlManager.XmlLogFile (filePath, rootname, attrib={})

Bases: object

Class to manage writing in salomeTools xml log file

**add\_simple\_node (node\_name, text=None, attrib={})**

Add a node with some attributes and text to the root node.

**Parameters**

- **str** (text) – the name of the node to add
- **str** – the text of the node
- **dict** (attrib) – the dictionary containing the attribute of the new node

**append\_node\_attrib (node\_name, attrib)**

Append a new attributes to the node that has node\_name as name

**Parameters**

- **str** (node\_name) – The name of the node on which append text
- **dict** (attrib) – The attrib to append

**append\_node\_text (node\_name, text)**

Append a new text to the node that has node\_name as name

**Parameters**

- **str** (text) – The name of the node on which append text
- **str** – The text to append

**write\_tree (stylesheet=None, file\_path=None)**

Write the xml tree in the log file path. Add the stylesheet if asked.

**Parameters** **str** (stylesheet) – The stylesheet to apply to the xml file

src.xmlManager.add\_simple\_node (root\_node, node\_name, text=None, attrib={})

Add a node with some attributes and text to the root node.

**Parameters**

- **etree.Element** (root\_node) – the Etree element where to add the new node
- **str** (text) – the name of the node to add

- **str** – the text of the node
- **dict** (*attrib*) – the dictionary containing the attribute of the new node

`src.xmlManager.append_node_attrib` (*root\_node*, *attrib*)

Append a new attributes to the node that has *node\_name* as name

#### Parameters

- **etree.Element** (*root\_node*) – the Etree element where to append the new attributes
- **dict** (*attrib*) – The attrib to append

`src.xmlManager.find_node_by_attrib` (*xmlroot*, *name\_node*, *key*, *value*)

Find the first node from *xmlroot* that has name *name\_node* and that has in its attributes {key : value}. Return the node

#### Parameters

- **etree.Element** (*xmlroot*) – the Etree element where to search
- **str** (*value*) – the name of node to search
- **str** – the key to search
- **str** – the value to search

**Returns** the found node

**Return type** *xmlroot* `etree.Element`

`src.xmlManager.write_report` (*filename*, *xmlroot*, *stylesheet*)

Writes a report file from a XML tree.

#### Parameters

- **str** (*stylesheet*) – The path to the file to create
- **etree.Element** (*xmlroot*) – the Etree element to write to the file
- **str** – The stylesheet to add to the begin of the file

## Module contents

initial imports and utilities methods for salomeTools

**class** `src.Path` (*path*)

**base** ()

**chmod** (*mode*)

**copy** (*path*, *smart=False*)

**copydir** (*dst*, *smart=False*)

**copyfile** (*path*)

**copylink** (*path*)

**dir** ()

**exists** ()

**isdir** ()

**isfile** ()

**islink** ()

**list** ()

**make** (*mode=None*)

**readlink** ()

**rm** ()

**smartcopy** (*path*)

**symlink** (*path*)

**exception** `src.SatException`  
 Bases: `exceptions.Exception`  
 rename Exception Class

`src.activate_mesa_property` (*config*)  
 Add mesa property into application properties

**Parameters** **Config** (*config*) – The global configuration. It must have an application!

`src.check_config_has_application` (*config, details=None*)  
 check that the config has the key APPLICATION. Else raise an exception.

**Parameters** **class** 'common.pyconf.Config' (*config*) – The config.

`src.check_config_has_profile` (*config, details=None*)  
 check that the config has the key APPLICATION.profile. else, raise an exception.

**Parameters** **class** 'common.pyconf.Config' (*config*) – The config.

`src.config_has_application` (*config*)

`src.deepcopy_list` (*input\_list*)  
 Do a deep copy of a list

**Parameters** **List** (*input\_list*) – The list to copy

**Returns** The copy of the list

**Return type** List

`src.ensure_path_exists` (*p*)  
 Create a path if not existing

**Parameters** **str** (*p*) – The path.

`src.find_file_in_lpath` (*file\_name, lpath, additional\_dir=""*)  
 Find in all the directories in lpath list the file that has the same name as file\_name. If it is found then return the full path of the file else return False.

The additional\_dir (optional) is the name of the directory to add to all paths in lpath.

**Parameters**

- **str** (*additional\_dir*) – The file name to search
- **List** (*lpath*) – The list of directories where to search
- **str** – The name of the additional directory

**Returns** the full path of the file or False if not found

**Return type** str

`src.get_base_path` (*config*)  
 Returns the path of the products base.

**Parameters** **Config** (*config*) – The global Config instance.

**Returns** The path of the products base.

**Return type** str

`src.get_cfg_param` (*config*, *param\_name*, *default*)  
Search for *param\_name* value in *config*. If *param\_name* is not in *config* then return *default*, else return the found value

**Parameters**

- **class** `'common.pyconf.Config'` (*config*) – The config.
- **str** (*default*) – the name of the parameter to get the value
- **str** – The value to return if *param\_name* is not in *config*

**Returns** see initial description of the function

**Return type** str

`src.get_launcher_name` (*config*)  
Returns the name of salome launcher.

**Parameters** **Config** (*config*) – The global Config instance.

**Returns** The name of salome launcher.

**Return type** str

`src.get_log_path` (*config*)  
Returns the path of the logs.

**Parameters** **Config** (*config*) – The global Config instance.

**Returns** The path of the logs.

**Return type** str

`src.get_property_in_product_cfg` (*product\_cfg*, *ppty*)

`src.get_salome_version` (*config*)

`src.get_tmp_filename` (*cfg*, *name*)

`src.handleRemoveReadOnly` (*func*, *path*, *exc*)

`src.merge_dicts` (*\*dict\_args*)

Given any number of dicts, shallow copy and merge into a new dict, precedence goes to key value pairs in latter dicts.

`src.only_numbers` (*str\_num*)

`src.parse_date` (*date*)

Transform YYYYMMDD\_hhmmss into YYYY-MM-DD hh:mm:ss.

**Parameters** **str** (*date*) – The date to transform

**Returns** The date in the new format

**Return type** str

`src.print_info` (*logger*, *info*)

Prints the tuples that are in *info* variable in a formatted way.

**Parameters**

- **Logger** (*logger*) – The logging instance to use for the prints.
- **list** (*info*) – The list of tuples to display

`src.read_config_from_a_file` (*filePath*)

`src.remove_item_from_list` (*input\_list*, *item*)

Remove all occurrences of *item* from *input\_list*

**Parameters** **List** (*input\_list*) – The list to modify

**Returns** The without any item



**Return type** List

src.**replace\_in\_file** (*filein*, *strin*, *strout*)  
Replace <strin> by <strout> in file <filein>



## RELEASE NOTES

### 5.1 Release notes

In construction.



## PYTHON MODULE INDEX

### S

- src, 66
- src.architecture, 34
- src.colorama, 33
- src.colorama.ansi, 29
- src.colorama.ansitowin32, 31
- src.colorama.initialise, 31
- src.colorama.win32, 32
- src.colorama.winterm, 32
- src.compilation, 34
- src.debug, 35
- src.ElementTree, 33
- src.environment, 36
- src.fileEnviron, 40
- src.fork, 46
- src.logger, 47
- src.options, 48
- src.printcolors, 49
- src.product, 51
- src.pyconf, 55
- src.system, 62
- src.template, 64
- src.test\_module, 64
- src.xmlManager, 64



## A

activate\_mesa\_property() (in module src), 67  
 add() (src.fileEnviron.LauncherFileEnviron method), 43  
 add\_comment() (src.environment.SalomeEnviron method), 38  
 add\_comment() (src.fileEnviron.BatFileEnviron method), 41  
 add\_comment() (src.fileEnviron.FileEnviron method), 42  
 add\_comment() (src.fileEnviron.LauncherFileEnviron method), 44  
 add\_comment() (src.fileEnviron.ScreenEnviron method), 45  
 add\_echo() (src.fileEnviron.ContextFileEnviron method), 41  
 add\_echo() (src.fileEnviron.FileEnviron method), 42  
 add\_echo() (src.fileEnviron.LauncherFileEnviron method), 44  
 add\_echo() (src.fileEnviron.ScreenEnviron method), 45  
 add\_line() (src.environment.SalomeEnviron method), 38  
 add\_line() (src.fileEnviron.FileEnviron method), 42  
 add\_line() (src.fileEnviron.LauncherFileEnviron method), 44  
 add\_line() (src.fileEnviron.ScreenEnviron method), 45  
 add\_link() (src.logger.Logger method), 47  
 add\_option() (src.options.Options method), 49  
 add\_simple\_node() (in module src.xmlManager), 65  
 add\_simple\_node() (src.xmlManager.XmlLogFile method), 65  
 add\_warning() (src.environment.SalomeEnviron method), 38  
 add\_warning() (src.fileEnviron.ContextFileEnviron method), 41  
 add\_warning() (src.fileEnviron.FileEnviron method), 42  
 add\_warning() (src.fileEnviron.LauncherFileEnviron method), 44  
 add\_warning() (src.fileEnviron.ScreenEnviron method), 45  
 addElement() (src.pyconf.Reference method), 61  
 addMapping() (src.pyconf.Mapping method), 60  
 addNamespace() (src.pyconf.Config method), 56  
 ANSI\_CSI\_RE (src.colorama.ansitowin32.AnsiToWin32 attribute), 31

ANSI\_OSC\_RE (src.colorama.ansitowin32.AnsiToWin32 attribute), 31  
 AnsiBack (class in src.colorama.ansi), 29  
 AnsiCodes (class in src.colorama.ansi), 29  
 AnsiCursor (class in src.colorama.ansi), 30  
 AnsiFore (class in src.colorama.ansi), 30  
 AnsiStyle (class in src.colorama.ansi), 30  
 AnsiToWin32 (class in src.colorama.ansitowin32), 31  
 append() (src.environment.Environ method), 36  
 append() (src.environment.SalomeEnviron method), 38  
 append() (src.fileEnviron.FileEnviron method), 42  
 append() (src.fileEnviron.LauncherFileEnviron method), 44  
 append() (src.fileEnviron.ScreenEnviron method), 45  
 append() (src.pyconf.Sequence method), 61  
 append\_node\_attrib() (in module src.xmlManager), 66  
 append\_node\_attrib() (src.xmlManager.XmlLogFile method), 65  
 append\_node\_text() (src.xmlManager.XmlLogFile method), 65  
 append\_value() (src.environment.Environ method), 37  
 append\_value() (src.fileEnviron.ContextFileEnviron method), 41  
 append\_value() (src.fileEnviron.FileEnviron method), 42  
 append\_value() (src.fileEnviron.LauncherFileEnviron method), 44  
 archive\_extract() (in module src.system), 62

## B

BACK() (src.colorama.ansi.AnsiCursor method), 30  
 back() (src.colorama.winterm.WinTerm method), 32  
 base() (src.Path method), 66  
 BashFileEnviron (class in src.fileEnviron), 40  
 batch() (in module src.fork), 46  
 batch\_salome() (in module src.fork), 46  
 BatFileEnviron (class in src.fileEnviron), 41  
 BLACK (src.colorama.ansi.AnsiBack attribute), 29  
 BLACK (src.colorama.ansi.AnsiFore attribute), 30  
 BLACK (src.colorama.winterm.WinColor attribute), 32  
 BLUE (src.colorama.ansi.AnsiBack attribute), 29  
 BLUE (src.colorama.ansi.AnsiFore attribute), 30  
 BLUE (src.colorama.winterm.WinColor attribute), 32  
 BRIGHT (src.colorama.ansi.AnsiStyle attribute), 30  
 BRIGHT (src.colorama.winterm.WinStyle attribute), 32

- BRIGHT\_BACKGROUND  
(src.colorama.winterm.WinStyle attribute), 32
- build\_configure() (src.compilation.Builder method), 34
- Builder (class in src.compilation), 34
- ## C
- call\_win32() (src.colorama.ansitowin32.AnsiToWin32 method), 31
- change\_to\_launcher() (src.fileEnviron.LauncherFileEnviron method), 44
- check() (src.compilation.Builder method), 34
- check\_config\_exists() (in module src.product), 51
- check\_config\_has\_application() (in module src), 67
- check\_config\_has\_profile() (in module src), 67
- check\_installation() (in module src.product), 51
- check\_source() (in module src.product), 51
- chmod() (src.Path method), 66
- cleancolor() (in module src.printcolors), 49
- clear\_line() (in module src.colorama.ansi), 30
- clear\_screen() (in module src.colorama.ansi), 30
- close() (src.debug.OutStream method), 36
- close() (src.ElementTree.TreeBuilder method), 33
- close() (src.ElementTree.XMLTreeBuilder method), 33
- close() (src.pyconf.ConfigInputStream method), 57
- close() (src.pyconf.ConfigOutputStream method), 58
- cmake() (src.compilation.Builder method), 34
- code\_to\_chars() (in module src.colorama.ansi), 30
- colorama\_text() (in module src.colorama.initialise), 31
- command\_value() (src.environment.Environ method), 37
- command\_value() (src.fileEnviron.BashFileEnviron method), 40
- command\_value() (src.fileEnviron.BatFileEnviron method), 41
- command\_value() (src.fileEnviron.ContextFileEnviron method), 41
- command\_value() (src.fileEnviron.FileEnviron method), 43
- command\_value() (src.fileEnviron.LauncherFileEnviron method), 44
- command\_value() (src.fileEnviron.ScreenEnviron method), 45
- Comment() (in module src.ElementTree), 33
- complete\_environment() (src.compilation.Builder method), 34
- Config (class in src.pyconf), 56
- Config.Namespace (class in src.pyconf), 56
- config\_has\_application() (in module src), 67
- ConfigError, 57
- ConfigFormatError, 57
- ConfigInputStream (class in src.pyconf), 57
- ConfigList (class in src.pyconf), 57
- ConfigMerger (class in src.pyconf), 57
- ConfigOutputStream (class in src.pyconf), 58
- ConfigReader (class in src.pyconf), 58
- ConfigResolutionError, 59
- configure() (src.compilation.Builder method), 34
- Container (class in src.pyconf), 59
- ContextFileEnviron (class in src.fileEnviron), 41
- convert\_ansi() (src.colorama.ansitowin32.AnsiToWin32 method), 31
- convert\_osc() (src.colorama.ansitowin32.AnsiToWin32 method), 31
- copy() (src.Path method), 66
- copydir() (src.Path method), 66
- copyfile() (src.Path method), 66
- copylink() (src.Path method), 66
- cursor\_adjust() (src.colorama.winterm.WinTerm method), 32
- cvs\_extract() (in module src.system), 62
- CYAN (src.colorama.ansi.AnsiBack attribute), 29
- CYAN (src.colorama.ansi.AnsiFore attribute), 30
- CYAN (src.colorama.winterm.WinColor attribute), 32
- ## D
- data() (src.ElementTree.TreeBuilder method), 33
- date\_to\_datetime() (in module src.logger), 47
- debug\_write() (src.options.Options method), 49
- deepcopy\_list() (in module src), 67
- deepCopyMapping() (in module src.pyconf), 61
- defaultMergeResolve() (in module src.pyconf), 61
- defaultStreamOpener() (in module src.pyconf), 62
- deinit() (in module src.colorama.initialise), 31
- delimiter (src.template.MyTemplate attribute), 64
- DIM (src.colorama.ansi.AnsiStyle attribute), 30
- dir() (src.Path method), 66
- do\_batch\_script\_build() (src.compilation.Builder method), 34
- do\_default\_build() (src.compilation.Builder method), 34
- do\_python\_script\_build() (src.compilation.Builder method), 35
- do\_script\_build() (src.compilation.Builder method), 35
- doctype() (src.ElementTree.XMLTreeBuilder method), 33
- DOWN() (src.colorama.ansi.AnsiCursor method), 30
- dump() (in module src.ElementTree), 33
- dump() (src.environment.SalomeEnviron method), 38
- ## E
- Element() (in module src.ElementTree), 33
- ElementTree (class in src.ElementTree), 33
- end() (src.ElementTree.TreeBuilder method), 33
- end\_write() (src.logger.Logger method), 47
- ensure\_path\_exists() (in module src), 67
- Environ (class in src.environment), 36
- erase\_line() (src.colorama.winterm.WinTerm method), 32
- erase\_screen() (src.colorama.winterm.WinTerm method), 32
- error() (src.logger.Logger method), 47
- evaluate() (src.pyconf.Container method), 60
- evaluate() (src.pyconf.Expression method), 60
- exists() (src.Path method), 66
- Expression (class in src.pyconf), 60



- extract\_params() (src.colorama.ansitowin32.AnsiToWin32 method), 31
- ## F
- feed() (src.ElementTree.XMLTreeBuilder method), 33
- FileEnviron (class in src.fileEnviron), 42
- FileEnvWriter (class in src.environment), 37
- filterLevel() (src.options.Options method), 49
- filterList2() (src.options.Options method), 49
- find() (src.ElementTree.ElementTree method), 33
- find\_file\_in\_lpath() (in module src), 67
- find\_node\_by\_attrib() (in module src.xmlManager), 66
- findall() (src.ElementTree.ElementTree method), 33
- findConfig() (src.pyconf.Reference method), 61
- findtext() (src.ElementTree.ElementTree method), 33
- finish() (src.environment.SalomeEnviron method), 38
- finish() (src.fileEnviron.BashFileEnviron method), 40
- finish() (src.fileEnviron.BatFileEnviron method), 41
- finish() (src.fileEnviron.ContextFileEnviron method), 42
- finish() (src.fileEnviron.FileEnviron method), 43
- finish() (src.fileEnviron.LauncherFileEnviron method), 44
- flush() (src.logger.Logger method), 47
- flush() (src.pyconf.ConfigOutputStream method), 58
- fore() (src.colorama.winterm.WinTerm method), 32
- format\_color\_exception() (in module src.debug), 36
- format\_exception() (in module src.debug), 36
- FORWARD() (src.colorama.ansi.AnsiCursor method), 30
- fromstring() (in module src.ElementTree), 33
- ## G
- generate\_launching\_commands() (src.test\_module.Test method), 64
- generate\_script() (src.test\_module.Test method), 64
- get() (src.environment.Environ method), 37
- get() (src.environment.SalomeEnviron method), 38
- get() (src.fileEnviron.BatFileEnviron method), 41
- get() (src.fileEnviron.ContextFileEnviron method), 42
- get() (src.fileEnviron.FileEnviron method), 43
- get() (src.fileEnviron.LauncherFileEnviron method), 44
- get() (src.fileEnviron.ScreenEnviron method), 45
- get() (src.pyconf.Mapping method), 60
- get\_attrib() (src.xmlManager.ReadXmlFile method), 65
- get\_attrs() (src.colorama.winterm.WinTerm method), 32
- get\_base\_install\_dir() (in module src.product), 51
- get\_base\_path() (in module src), 67
- get\_cfg\_param() (in module src), 67
- get\_distrib\_version() (in module src.architecture), 34
- get\_distribution() (in module src.architecture), 34
- get\_file\_environ() (in module src.fileEnviron), 46
- get\_help() (src.options.Options method), 49
- get\_install\_dir() (in module src.product), 52
- get\_launcher\_name() (in module src), 68
- get\_log\_path() (in module src), 68
- get\_names() (src.environment.SalomeEnviron method), 38
- get\_nb\_proc() (in module src.architecture), 34
- get\_node\_text() (src.xmlManager.ReadXmlFile method), 65
- get\_position() (src.colorama.winterm.WinTerm method), 32
- get\_product\_components() (in module src.product), 52
- get\_product\_config() (in module src.product), 52
- get\_product\_dependencies() (in module src.product), 52
- get\_product\_section() (in module src.product), 52
- get\_products\_infos() (in module src.product), 53
- get\_property\_in\_product\_cfg() (in module src), 68
- get\_python\_version() (in module src.architecture), 34
- get\_salome\_version() (in module src), 68
- get\_test\_timeout() (src.test\_module.Test method), 64
- get\_tmp\_dir() (src.test\_module.Test method), 64
- get\_tmp\_filename() (in module src), 68
- get\_user() (in module src.architecture), 34
- get\_win32\_calls() (src.colorama.ansitowin32.AnsiToWin32 method), 31
- getByPath() (src.pyconf.Config method), 56
- getByPath() (src.pyconf.ConfigList method), 57
- getChar() (src.pyconf.ConfigReader method), 58
- getCurrentLogger() (in module src.logger), 47
- getDefaultLogger() (in module src.logger), 47
- getDetailOption() (src.options.Options method), 49
- getiterator() (src.ElementTree.ElementTree method), 33
- getLocalEnv() (in module src.debug), 36
- getroot() (src.ElementTree.ElementTree method), 33
- getRootAttrib() (src.xmlManager.ReadXmlFile method), 64
- getStrConfigDbg() (in module src.debug), 36
- getStrConfigStd() (in module src.debug), 36
- getTmpDirDEFAULT() (in module src.test\_module), 64
- getToken() (src.pyconf.ConfigReader method), 58
- getUnittestLogger() (in module src.logger), 47
- git\_extract() (in module src.system), 63
- GREEN (src.colorama.ansi.AnsiBack attribute), 29
- GREEN (src.colorama.ansi.AnsiFore attribute), 30
- GREEN (src.colorama.winterm.WinColor attribute), 32
- GREY (src.colorama.winterm.WinColor attribute), 32
- ## H
- hack\_libtool() (src.compilation.Builder method), 35
- handleMismatch() (src.pyconf.ConfigMerger method), 57
- handleRemoveReadOnly() (in module src), 68
- ## I
- indent() (in module src.debug), 36
- indent() (src.options.Options method), 49
- init() (in module src.colorama.initialise), 31
- install() (src.compilation.Builder method), 35
- InStream (class in src.debug), 36

is\_a\_tty() (in module src.colorama.ansitowin32), 31  
 is\_defined() (src.environment. Environ method), 37  
 is\_defined() (src.environment.SalomeEnviron method), 38  
 is\_defined() (src.fileEnviron.FileEnviron method), 43  
 is\_defined() (src.fileEnviron.LauncherFileEnviron method), 44  
 is\_defined() (src.fileEnviron.ScreenEnviron method), 45  
 is\_stream\_closed() (in module src.colorama.ansitowin32), 31  
 is\_windows() (in module src.architecture), 34  
 isCurrentLoggerUnittest() (in module src.logger), 48  
 isdir() (src.Path method), 66  
 iselement() (in module src.ElementTree), 33  
 isfile() (src.Path method), 66  
 islink() (src.Path method), 66  
 isTypeConfig() (in module src.debug), 36  
 isWord() (in module src.pyconf), 62  
 iteritems() (src.pyconf.Mapping method), 60  
 iterkeys() (src.pyconf.Mapping method), 60  
 iterparse (class in src.ElementTree), 33

## K

keys() (src.pyconf.Mapping method), 60

## L

launch\_command() (in module src.fork), 46  
 LauncherFileEnviron (class in src.fileEnviron), 43  
 LIGHTBLACK\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTBLACK\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTBLUE\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTBLUE\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTCYAN\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTCYAN\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTGREEN\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTGREEN\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTMAGENTA\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTMAGENTA\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTRED\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTRED\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 LIGHTWHITE\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTWHITE\_EX (src.colorama.ansi.AnsiFore attribute), 30

LIGHTYELLOW\_EX (src.colorama.ansi.AnsiBack attribute), 29  
 LIGHTYELLOW\_EX (src.colorama.ansi.AnsiFore attribute), 30  
 list() (src.Path method), 66  
 list\_log\_file() (in module src.logger), 48  
 load() (src.pyconf.Config method), 56  
 load() (src.pyconf.ConfigReader method), 58  
 load\_cfg\_environment() (src.environment.SalomeEnviron method), 39  
 load\_environment() (in module src.environment), 40  
 location() (src.pyconf.ConfigReader method), 58  
 log() (src.compilation.Builder method), 35  
 log\_command() (src.compilation.Builder method), 35  
 Logger (class in src.logger), 47

## M

MAGENTA (src.colorama.ansi.AnsiBack attribute), 29  
 MAGENTA (src.colorama.ansi.AnsiFore attribute), 30  
 MAGENTA (src.colorama.winterm.WinColor attribute), 32  
 make() (src.compilation.Builder method), 35  
 make() (src.Path method), 67  
 makePath() (in module src.pyconf), 62  
 Mapping (class in src.pyconf), 60  
 match() (src.pyconf.ConfigReader method), 58  
 merge() (src.pyconf.ConfigMerger method), 57  
 merge\_dicts() (in module src), 68  
 mergeMapping() (src.pyconf.ConfigMerger method), 57  
 mergeSequence() (src.pyconf.ConfigMerger method), 58  
 MyTemplate (class in src.template), 64

## N

next() (src.ElementTree.iterparse method), 33  
 next() (src.pyconf.Sequence.SeqIter method), 61  
 NORMAL (src.colorama.ansi.AnsiStyle attribute), 30  
 NORMAL (src.colorama.winterm.WinStyle attribute), 32

## O

only\_numbers() (in module src), 68  
 Options (class in src.options), 49  
 OptResult (class in src.options), 48  
 OutputStream (class in src.debug), 36  
 overwriteKeys() (src.pyconf.ConfigMerger method), 58  
 overwriteMergeResolve() (in module src.pyconf), 62

## P

parse() (in module src.ElementTree), 33  
 parse() (src.ElementTree.ElementTree method), 33  
 parse\_args() (src.options.Options method), 49  
 parse\_date() (in module src), 68  
 parseFactor() (src.pyconf.ConfigReader method), 58  
 parseKeyValuePair() (src.pyconf.ConfigReader method), 59

- parseMapping() (src.pyconf.ConfigReader method), 59  
 parseMappingBody() (src.pyconf.ConfigReader method), 59  
 parseReference() (src.pyconf.ConfigReader method), 59  
 parseScalar() (src.pyconf.ConfigReader method), 59  
 parseSequence() (src.pyconf.ConfigReader method), 59  
 parseSuffix() (src.pyconf.ConfigReader method), 59  
 parseTerm() (src.pyconf.ConfigReader method), 59  
 parseValue() (src.pyconf.ConfigReader method), 59  
 Path (class in src), 66  
 pattern (src.template.MyTemplate attribute), 64  
 PI() (in module src.ElementTree), 33  
 pop\_debug() (in module src.debug), 36  
 POS() (src.colorama.ansi.AnsiCursor method), 30  
 prepare() (src.compilation.Builder method), 35  
 prepare\_testbase() (src.test\_module.Test method), 64  
 prepare\_testbase\_from\_dir() (src.test\_module.Test method), 64  
 prepare\_testbase\_from\_git() (src.test\_module.Test method), 64  
 prepare\_testbase\_from\_svn() (src.test\_module.Test method), 64  
 prepend() (src.environment. Environ method), 37  
 prepend() (src.environment.SalomeEnviron method), 39  
 prepend() (src.fileEnviron.FileEnviron method), 43  
 prepend() (src.fileEnviron.LauncherFileEnviron method), 44  
 prepend() (src.fileEnviron.ScreenEnviron method), 46  
 prepend\_value() (src.environment. Environ method), 37  
 prepend\_value() (src.fileEnviron.ContextFileEnviron method), 42  
 prepend\_value() (src.fileEnviron.FileEnviron method), 43  
 prepend\_value() (src.fileEnviron.LauncherFileEnviron method), 45  
 print\_color\_map() (in module src.printcolors), 49  
 print\_color\_range() (in module src.printcolors), 50  
 print\_info() (in module src), 68  
 print\_value() (in module src.printcolors), 50  
 printc() (in module src.printcolors), 50  
 printcError() (in module src.printcolors), 50  
 printcHeader() (in module src.printcolors), 50  
 printcHighlight() (in module src.printcolors), 50  
 printcInfo() (in module src.printcolors), 50  
 printcLabel() (in module src.printcolors), 51  
 printcSuccess() (in module src.printcolors), 51  
 printcWarning() (in module src.printcolors), 51  
 ProcessingInstruction() (in module src.ElementTree), 33  
 product\_compiles() (in module src.product), 53  
 product\_has\_env\_script() (in module src.product), 53  
 product\_has\_logo() (in module src.product), 53  
 product\_has\_patches() (in module src.product), 53  
 product\_has\_salome\_gui() (in module src.product), 53  
 product\_has\_script() (in module src.product), 53  
 product\_is\_autotools() (in module src.product), 53  
 product\_is\_cmake() (in module src.product), 54  
 product\_is\_cpp() (in module src.product), 54  
 product\_is\_debug() (in module src.product), 54  
 product\_is\_dev() (in module src.product), 54  
 product\_is\_fixed() (in module src.product), 54  
 product\_is\_generated() (in module src.product), 54  
 product\_is\_mpi() (in module src.product), 54  
 product\_is\_native() (in module src.product), 54  
 product\_is\_salome() (in module src.product), 55  
 product\_is\_smesh\_plugin() (in module src.product), 55  
 product\_is\_vcs() (in module src.product), 55  
 product\_is\_verbose() (in module src.product), 55  
 push\_debug() (in module src.debug), 36  
 put\_initial\_xml\_fields() (src.logger.Logger method), 47  
 put\_txt\_log\_in\_appli\_log\_dir() (src.compilation.Builder method), 35
- ## Q
- QName (class in src.ElementTree), 33
- ## R
- read() (src.pyconf.ConfigInputStream method), 57  
 read\_config\_from\_a\_file() (in module src), 68  
 read\_results() (src.test\_module.Test method), 64  
 readline() (src.pyconf.ConfigInputStream method), 57  
 readlink() (src.Path method), 67  
 ReadXmlFile (class in src.xmlManager), 64  
 RED (src.colorama.ansi.AnsiBack attribute), 29  
 RED (src.colorama.ansi.AnsiFore attribute), 30  
 RED (src.colorama.winterm.WinColor attribute), 32  
 Reference (class in src.pyconf), 61  
 reinit() (in module src.colorama.initialise), 31  
 remove\_item\_from\_list() (in module src), 68  
 removeNamespace() (src.pyconf.Config method), 57  
 replace\_in\_file() (in module src), 69  
 RESET (src.colorama.ansi.AnsiBack attribute), 29  
 RESET (src.colorama.ansi.AnsiFore attribute), 30  
 RESET\_ALL (src.colorama.ansi.AnsiStyle attribute), 30  
 reset\_all() (in module src.colorama.initialise), 31  
 reset\_all() (src.colorama.ansitowin32.AnsiToWin32 method), 31  
 reset\_all() (src.colorama.winterm.WinTerm method), 32  
 resolve() (src.pyconf.Reference method), 61  
 rm() (src.Path method), 67  
 run\_all\_tests() (src.test\_module.Test method), 64  
 run\_env\_script() (src.environment.SalomeEnviron method), 39  
 run\_env\_script() (src.fileEnviron.ScreenEnviron method), 46  
 run\_grid\_tests() (src.test\_module.Test method), 64  
 run\_script() (src.test\_module.Test method), 64  
 run\_session\_tests() (src.test\_module.Test method), 64  
 run\_simple\_env\_script() (src.environment.SalomeEnviron method), 39  
 run\_testbase\_tests() (src.test\_module.Test method), 64

run\_tests() (src.test\_module.Test method), 64

## S

SalomeEnviron (class in src.environment), 38

SatException, 67

saveConfigDbg() (in module src.debug), 36

saveConfigStd() (in module src.debug), 36

ScreenEnviron (class in src.fileEnviron), 45

search\_known\_errors() (src.test\_module.Test method), 64

sendMessageToCurrentLogger() (in module src.logger), 48

Sequence (class in src.pyconf), 61

Sequence.SeqIter (class in src.pyconf), 61

set() (src.environment.Environ method), 37

set() (src.environment.SalomeEnviron method), 39

set() (src.fileEnviron.BashFileEnviron method), 40

set() (src.fileEnviron.BatFileEnviron method), 41

set() (src.fileEnviron.ContextFileEnviron method), 42

set() (src.fileEnviron.FileEnviron method), 43

set() (src.fileEnviron.LauncherFileEnviron method), 45

set() (src.fileEnviron.ScreenEnviron method), 46

set\_a\_product() (src.environment.SalomeEnviron method), 39

set\_application\_env() (src.environment.SalomeEnviron method), 39

set\_attrs() (src.colorama.winterm.WinTerm method), 32

set\_console() (src.colorama.winterm.WinTerm method), 32

set\_cpp\_env() (src.environment.SalomeEnviron method), 39

set\_cursor\_position() (src.colorama.winterm.WinTerm method), 32

set\_full\_environ() (src.environment.SalomeEnviron method), 39

set\_products() (src.environment.SalomeEnviron method), 40

set\_python\_libdirs() (src.environment.SalomeEnviron method), 40

set\_salome\_generic\_product\_env() (src.environment.SalomeEnviron method), 40

set\_salome\_minimal\_product\_env() (src.environment.SalomeEnviron method), 40

set\_title() (in module src.colorama.ansi), 30

set\_title() (src.colorama.winterm.WinTerm method), 32

SetConsoleTextAttribute() (in module src.colorama.win32), 32

setCurrentLogger() (in module src.logger), 48

setPath() (src.pyconf.Container method), 60

setStream() (src.pyconf.ConfigReader method), 59

Shell (class in src.environment), 40

should\_wrap() (src.colorama.ansitowin32.AnsiToWin32 method), 31

show\_command\_log() (in module src.logger), 48

show\_in\_editor() (in module src.system), 63

show\_progress() (in module src.fork), 46

smartcopy() (src.Path method), 67

special\_path\_separator() (in module src.fileEnviron), 46

src (module), 66

src.architecture (module), 34

src.colorama (module), 33

src.colorama.ansi (module), 29

src.colorama.ansitowin32 (module), 31

src.colorama.initialise (module), 31

src.colorama.win32 (module), 32

src.colorama.winterm (module), 32

src.compilation (module), 34

src.debug (module), 35

src.ElementTree (module), 33

src.environment (module), 36

src.fileEnviron (module), 40

src.fork (module), 46

src.logger (module), 47

src.options (module), 48

src.printcolors (module), 49

src.product (module), 51

src.pyconf (module), 55

src.system (module), 62

src.template (module), 64

src.test\_module (module), 64

src.xmlManager (module), 64

start() (src.ElementTree.TreeBuilder method), 33

StreamWrapper (class in src.colorama.ansitowin32), 31

style() (src.colorama.winterm.WinTerm method), 32

SubElement() (in module src.ElementTree), 33

substitute() (in module src.template), 64

svn\_extract() (in module src.system), 63

symlink() (src.Path method), 67

## T

Test (class in src.test\_module), 64

timedelta\_total\_seconds() (in module src.logger), 48

tofix() (in module src.debug), 36

tostring() (in module src.ElementTree), 33

TreeBuilder (class in src.ElementTree), 33

## U

UP() (src.colorama.ansi.AnsiCursor method), 30

update\_hat\_xml() (in module src.logger), 48

## W

WHITE (src.colorama.ansi.AnsiBack attribute), 29

WHITE (src.colorama.ansi.AnsiFore attribute), 30

winapi\_test() (in module src.colorama.win32), 32

WinColor (class in src.colorama.winterm), 32

WinStyle (class in src.colorama.winterm), 32

WinTerm (class in src.colorama.winterm), 32

wmake() (src.compilation.Builder method), 35

wrap\_stream() (in module src.colorama.initialise), 31

write() (in module src.debug), 36

write() (src.colorama.ansitowin32.AnsiToWin32 method), 31

`write()` (`src.colorama.ansitowin32.StreamWrapper` method), 31

`write()` (`src.ElementTree.ElementTree` method), 33

`write()` (`src.fileEnviron.ScreenEnviron` method), 46

`write()` (`src.logger.Logger` method), 47

`write()` (`src.pyconf.ConfigOutputStream` method), 58

`write_and_convert()` (`src.colorama.ansitowin32.AnsiToWin32` method), 31

`write_back()` (in module `src.fork`), 46

`write_cfgForPy_file()` (`src.environment.FileEnvWriter` method), 37

`write_env_file()` (`src.environment.FileEnvWriter` method), 38

`write_plain_text()` (`src.colorama.ansitowin32.AnsiToWin32` method), 31

`write_report()` (in module `src.xmlManager`), 66

`write_test_margin()` (`src.test_module.Test` method), 64

`write_tree()` (`src.xmlManager.XmlLogFile` method), 65

`writeToStream()` (`src.pyconf.Container` method), 60

`writeToStream()` (`src.pyconf.Mapping` method), 61

`writeToStream()` (`src.pyconf.Sequence` method), 61

`writeValue()` (`src.pyconf.Container` method), 60

## X

`XML()` (in module `src.ElementTree`), 33

`XmlLogFile` (class in `src.xmlManager`), 65

`XMLTreeBuilder` (class in `src.ElementTree`), 33

## Y

`YELLOW` (`src.colorama.ansi.AnsiBack` attribute), 29

`YELLOW` (`src.colorama.ansi.AnsiFore` attribute), 30

`YELLOW` (`src.colorama.winterm.WinColor` attribute), 32