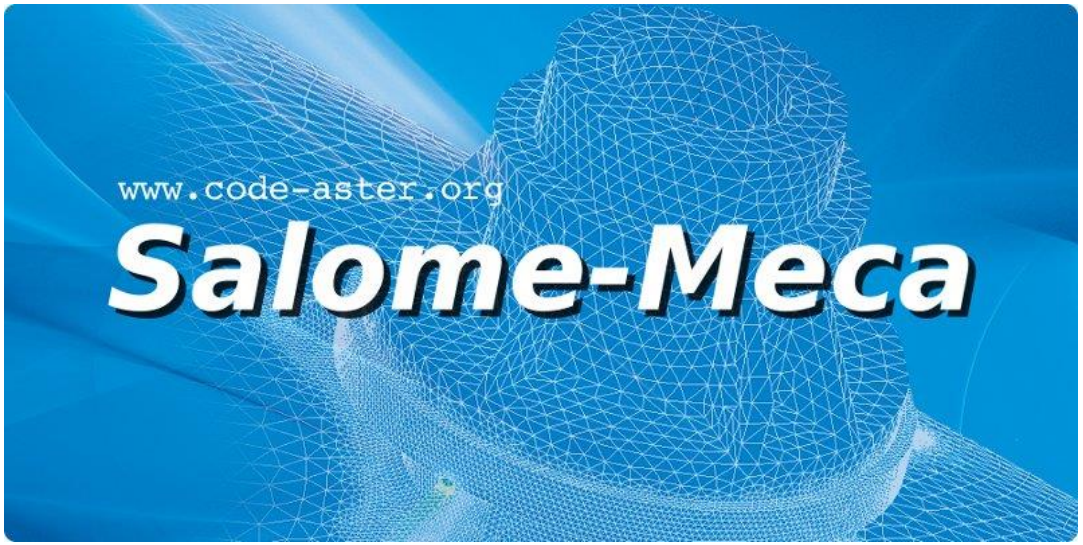
Development in code_aster
Building code_aster using waf



Code_Aster, Salome-Meca course material

EDF

code_aster prerequisites

code_aster is written in Fortran90 (1M.lines) , C (10 k.lines), Python (200 k.lines). We need:

- Fortran & C compilers
- Python + numpy
- Mathematics (blas, lapack) librairies
- libhdf5, libmed (for input/output with SALOME)
- mfront (behaviours integration)
- libmetis (renumbering for solvers)
- scotch (renumbering and partitionning tool for solvers)
- mumps, petsc solvers
- mpi...

These requirements must first be found in order to build code_aster (this is called the *configuration* step).

→ This often requires a specific environment.

Getting the source files

code_aster repositories:

- src: source files (C, fortran, python), verification testcases and build scripts (~350MB)
- validation: testcases containing confidential data and validation testcases (~1GB)
- data: files with non-public data
- devtools: environment scripts, developer tools

Main branches:

- v12: maintenance branch, only for bugfixes

  Major release (ex. 12.4.0) is tagged ***stable***,
  current minor release (ex. 12.4.7) is called ***stable-updates***.

  Previous branch is v11 and its last release is tagged ***oldstable***.

- default: development branch for bugfixes and new features

  Major release (ex. 13.1.0) is tagged ***testing***,
  current minor release (ex. 13.1.13) is called ***unstable***.

The tag *unstable* moves every week, the tags *stable*, *testing* move with Salome-Meca releases, the tag *oldstable* moves every two years.

What is waf?

waf is a Python-based framework for configuring, compiling and installing applications.

- *waf configure*
  - Search for the compilers (C, Fortran, C++), prerequisites header files and librairies…
  - Check the consistency of the prerequisites with the expected ones in Code_Aster.
  - Store some variables to make Code_Aster build easier (current tag).
- *waf build*
  - Automatically executed by *install*
  - Compile source files, create librairies and executable
- *waf install*
  - Copy all files in the installation directory
  - Build the elements database
- waf can execute testcases and may help for debugging.

See *waf --help* for all available options.

Build code_aster step by step (1)

http://aster-services.der.edf.fr/mercurial/waf.html#construction-de-code-aster

- Clone some repo (src, devtools, validation):

```
mkdir -p $HOME/dev/codeaster && cd $HOME/dev/codeaster
hg clone http://aster-repo.der.edf.fr/scm/hg/codeaster/devtools
hg clone http://aster-repo.der.edf.fr/scm/hg/codeaster/src
```

- Set the environment for compilers:

```
. $HOME/dev/codeaster/devtools/etc/env_unstable.sh
```

- Finalize repositories configuration (add clones addresses, aliases…)

```
install_env [--internet]
```

- Configure code_aster:

```
cd $HOME/dev/codeaster/src
./waf configure --use-config=calibre9 --prefix=../install/std
```

Or just:

```
./waf configure
```

The configuration used is automatically defined by *env_unstable.sh* (through *DEVTOOLS_COMPUTER_ID* var)

To build code_aster from Bitbucket repo, first install an aster-full package and use the provided configuration file.

Build code_aster step by step (2)

- Build and install code_aster:

  ```
  ./waf install
  ```

  The build directory is *build/release*.

- Run tests with Waf:

  ```
  ./waf test -n sslp114a -n ssnv15a
  ```

- Run a list of tests:

  ```
  run_testcases [--testlist=list_of_tests] [--resutest=../resutest] [...]
  ```

- Run a test over several revisions:

  ```
  run_testcase_bisect --good (good rev) --bad (bad rev) -n test_name
  ```

Build code_aster - variants

« Native » variant:

```
waf install_debug
waf test_debug -n ssnp15a –exectool=debugger
```

Build a version with debug symbols (by passing the option '*-g*' to the compilers), build directory is *build/debug*.

waf_variant script:

- Allows to build the same source code with different options (compilers, librairies, options...).
- Uses a separated build directory: *build/'variant'*.
- Just create a symlink named waf_'name':

```
ln -s waf_variant waf_mpi
```

Usual variant: for the MPI version

- use *env_unstable_mpi.sh* on official platforms (will use *calibre9_mpi* or *athosdev_mpi* configuration for example).

End of presentation

Is something missing or unclear in this document?

Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about Code_Aster training materials.

Do not hesitate to share with us your comments on the Code_Aster forum dedicated thread.