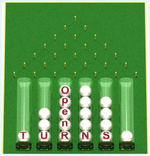
OpenTURNS modules development

Trainer : Sofiane Haddad
Airbus
sofiane.s.haddad@airbus.com

Developers training



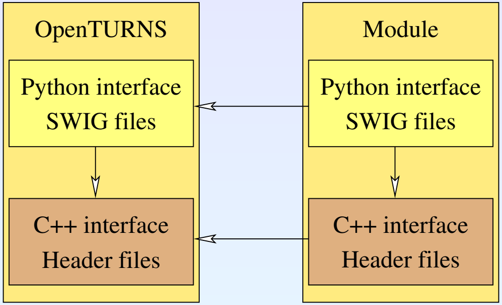
OpenTURNS modules development

OpenTURNS modules

Objectives

OpenTURNS is a growing system developed by a small team. A constant problem is to assess the stability of the whole product, and in order to achieve this goal we introduced a notion of module, in order to insulate a core library dedicated to the definition of the abstract data model and to propose all the specific algorithms as optional modules. The core would evolve quite slowly, insuring its robustness, whereas the modules would have a more dynamic development model.
Another key objective is to provide a way to extend the existing platform with functionalities developed by teams that are reluctant to adopt the OpenTURNS development process. Within the module, the development team can adopt any coding rule or programming language he want, as long as the OpenTURNS interface is respected as well as the objects lifecycle.

OpenTURNS modules



The principles

An OpenTURNS module is typically made of two parts:

- a C++ part that uses the OpenTURNS C++ interface in order to provide new specialization or to produce instances of the data model using new algorithms with no OpenTURNS counterpart.
- a Python part, the Python interface of the C++ part, often obtained using SWIG. In this case, this SWIG interface must use the OpenTURNS interface the same way its C++ interface uses the OpenTURNS C++ interface.

OpenTURNS modules

And for a Python module?

For now, there is (almost) no way to use a Python object within the OpenTURNS C++ library, excepting:

- A Distribution;
- a Function;
- An experiment
- ...

and use these one as "OpenTURNS classes".
The concept of "OpenTURNS Python module" is not very specific: we get some ‘full python‘ packages such as `oticp`, `otwrap` , `otsklearn`. But these examples are in a way independent as they rely on the "OpenTURNS API" and the API is not able to ‘integrate‘ them. In other words, any set of Python classes or Python functions that use the OpenTURNS Python interface can be called an OpenTURNS Python module. In this case, the notion of OpenTURNS module is mainly a packaging notion, and the use of the Python setup tools is probably more mature and more adapted!

Module development

Step 1: copy and adapt an existing template

- Copy and rename the source tree of an example module (for example the Strange module) from the OpenTURNS source tree. The examples modules are located under the module subdirectory of OpenTURNS source tree:
  ```
  git clone https://github.com/openturns/ottemplate.git MyModule
  ```
- Adapt the template to your module:
  ```
  ./customize MyModule
  ```
  This command change the module name into all the scripts, and adapt the example class to this new name.

Module development

Step 2: develop the module

- Implement your module. You are free to use the rules you want, but if the final objective of the module is to be integrated in the official release of OpenTURNS, it is wise to adopt the OpenTURNS development process and rules.
- Build your module as usual:
  ```
  mkdir build
  cd build
  cmake .. -DCMAKE_INSTALL_PREFIX=INSTALLDIR
  -DOpenTURNS_DIR=OPENTURNS_INSTALLDIR/lib/cmake/openturns
  make
  ```
- Create a source package of your module:
  ```
  make dist
  ```
  It will create a tarball named mymodule-X.Y.Z.tar.gz (and mymodule-X.Y.Z.tar.bz2), where X.Y.Z is the version number of the module.

Module development

Step 3: documentation

Module documentation is very close to API one:

- Developer guide : Architecture, validation
- SWIG documentation is to be completed (docstrings);
- Examples and API documentation;
- Theory (if needed)

Module development

Step 4: install and test the module

- Check that you have a working OpenTURNS installation, for example by trying to load the OpenTURNS module within an interactive python session:
  ```
  python
  >>> import openturns as ot
  ```
  and python should not complain about a non existing openturns module. The installation script has many more capabilities, you can access to its embedded documentation by invoking it without argument.
- Test your module within python:
  ```
  python
  >>> import openturns as ot
  >>> import mymodule
  ```
  and python should not complain about a non existing mymodule module.

Module development

Step 5: Maintenance

Each developer is responsible of his package should maintain it!

- We can rely on continuous integration tool accounting API changes
- He is in charge of bugs (except if underlying bug is in the API)

Exception : modules maintained by the consortium!

Step 6: Packaging

No specific rule concerning the packaging!

- Tarball generation
- Git
- conda...

Exception : modules maintained by the consortium!