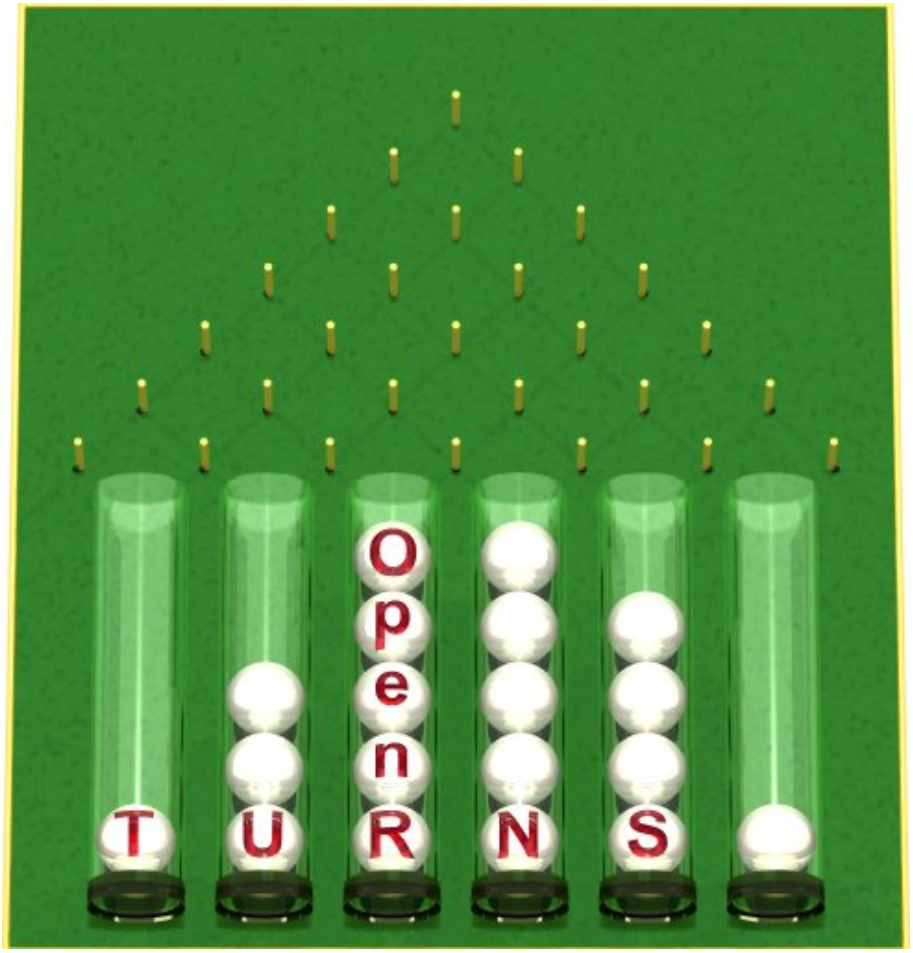
Contribution Guide

Open TURNS version 0.12.1

November 8, 2008



Contents

1 Introduction

This documentation aims at guiding the developpers in their contributions to the OpenTURNS software. This contribution can be done in (at least) the three following contexts:

- a contribution to the C++ library,
- a contribution to the TUI written in python,
- a contribution in the R language.

2 How to add a class MyClass in an existing directory of OpenTURNS sources?

This how-to explains the process that must be followed to fully integrate a new class that provides an end-user facility (e.g. a new distribution). We suppose that this class will take place in an existing directory of the sources directories, to avoid the burden of the autotools infrastructure creation.

2.1 First, add the class to the OpenTURNS C++ library

1. Create MyClass.hxx and MyClass.cxx in the same directory. The files must have the standard header comment, with a brief description of the class in Doxygen form and the standard reference to the LGPL license.

   For the header file MyClass.hxx, the interface must be embraced between the preprocessing clauses:

   ```
   #ifndef OPENTURNS_MYCLASS_HXX
   #define OPENTURNS_MYCLASS_HXX
   ...
   your interface
   ...
   #endif OPENTURNS_MYCLASS_HXX
   ```

   to prevent from multiple inclusions.

   See any pair of .hxx/.cxx files in the current directory and the PGQL document for the OpenTURNS coding rules: use of namespaces, case convention for the static methods, the methods and the attributes, trailing underscore for the attribute names for namming a few.

2. Modify the Makefile.am file in the directory containing MyClass.hxx and MyClass.cxx:

   - add MyClass.hxx to the otinclude_HEADERS variable
   - add MyClass.cxx to the libOTXXXXXX_la__SOURCES variable, where XXXXXX is the name of the current directory.

3. Create a test file t_MyClass_std.cxx in the directory lib/test. This test file must use the standard functionalities of the class MyClass.

4. Create an autotest file t_MyClass_std.at in the directory lib/test. This file describes the test, how to run it and what is the expected output (copy-paste the *validated* output of the test in the proper section of t_MyClass_std.at)

5. Modify the Makefile.am file in lib/test:
   - add t_MyClass_std (which is the name of the test executable) to the variable HECK_PROGS or INSTALLCHECK_PROGS depending on the fact the test checks the correct behaviour of OpenTURNS independently of its installation or not. The several executables are organized following the library organization, you must follow this rule.
   - add t_MyClass_std.at to the variable CHECK_TESTS or INSTALLCHECK_TESTS and in the correct set of autotest files, following the same rules than for the executable.
   - Create a variable called t_MyClass_std_SOURCES and set its value to t_MyClass.cxx in the relevant set of sources.
6. Add t_MyClass_std.at to the file check_testsuite.at or installcheck_testsuite.at using the same rule than for the Makefile.am modification.
7. If the validation of your class involved advanced mathematics, or was a significant work using other tools, you can add this validation in the validation/src directory.
   - copy all of your files in the validation/src directory.
   - modify the Makefile.am file by appending the list of your files to the dist_validation_DATA variable.

That's it! Your class is integrated to the library and will be checked for non-regression in all the subsequent versions of OpenTURNS, assuming that your contribution has been incorporated in the "official" OpenTURNS release. But nobody can use it!

2.2 Second, document your contribution (in english, using LaTeX)

8. Add an entry in the document doc/src/ArchitectureGuide/OpenTURNS_ArchitectureGuide.tex if your class has a significant impact on the library architecture.
9. Add an entry in the document doc/src/WrappersGuide/OpenTURNS_WrappersGuide.tex if your class has a significant impact on the way OpenTURNS interfaces external codes.
10. Add an entry in the document doc/src/ReferenceGuide/OpenTURNS_ReferenceGuide.tex if your class add a new concept not already described in the reference guide. Your entry must take the form of a specific description using the same template than the other descriptions.

Ok, your contribution can be used by a programmer who uses the library. But for the other users, some work remains.

2.3 Third: make your contibution usable from the Textual User Interface

11. Create MyClass.i in the python/src directory. In most situations, it should be:

```
// SWIG file MyClass.i
// Author : $LastChangedBy: dutka $
// Date : $LastChangedDate: 2008-06-26 13:50:17 +0200 (jeu, 26 jun 2008) $
// Id : $Id: OpenTURNS_ContributionGuide.tex 862 2008-06-26 11:50:17Z dutka $

%{
#include "MyClass.hxx"
```

```
%}

%include MyClass.hxx
namespace OpenTURNS { namespace NameSpace1 { namespace NameSpace2 { %extend MyClass {
MyClass(const MyClass & other) { return new OpenTURNS::NameSpace1::NameSpace2::MyClass(
other); } } }}}
```

if your class is in the namespace OpenTURNS::NameSpace1::NameSpace2.

12. Modify the Makefile.am file in python/src: add MyClass.i to the variable OPENTURNS_SWIG_SRC
13. Modify the file openturns.i to include MyClass.i in the correct set of .i files (see the comments in openturns.i file)
14. Create a test file t_MyClass_std.py in the directory python/test. This test implements the same tests than t_MyClass_std.cxx, but using python.
15. Create an autotest file t_MyClass_std.atpy that has the same role than t_MyClass_std.at, but for the python test.
16. Modify the Makefile.am file in python/test:
    - add t_MyClass_std.py to the variable PYTHONINSTALLCHECK_PROGS. The several executables are organized following the library organization, you must follow this rule.
    - add t_MyClass_std.atpy to the variable PYTHONINSTALLCHECK_TESTS.

2.4 Allmost finished. Document your contribution to the TUI

17. Comment your python test as a new use-case in the document doc/src/OpenTURNS_UseCasesGuide/UseCasesGuide.tex following the generic format of this document:
    - describe the inputs of your use-case.
    - extract code snippets that show the user interaction with your class.
    - add the relevant keywords to the index.
18. Gives a description of your class in the document doc/src/UserManual/OpenTURNS_UserManual.tex
    - following the general form of this document, fill-in the sections but only describe the methods the user is intended to use (forget the most computer programming inclined methods).
    - don't hesitate to give some reminders of theoretical aspects if needed, in the form of an equation or a short (1 or 2 sentences) mathematical explanation. Give a pointer to the relevant reference guide section.

That's all, folks!
Some timings from an OpenTURNS Guru: 2 days of work for the most trivial contribution (a copy-paste of a class with 5 methods, no mathematical or algorithmic tricks). For a well-trained OpenTURNS contributor, a user-visible class with a dozen of methods and well-understood algorithms, a new class should not be less than a week of work...

3 How to add a whole set of classes in a new subdirectory of OpenTURNS sources?

This how-to explains the process that must be followed to fully integrate a set of classes that provides an end-user facility (e.g. a new simulation algorithm) developped in a new subdirectory of the existing sources. The task is very similar to the steps described in the how-to (2), only the new steps will be described. We suppose that the subdirectory has already been created, as well as the several source files. There are three new steps in additon to those of the how-to (2): the creation of the autotool infrastructure in the new subdirectory, the modification of the autotool infrastructure in the parent directory and the modification of the autotool infrastructure in the root directory.

3.1 Autotool infrastructure in the new subdirectory

3.2 Autotool infrastructure in the parent sudirectory

3.3 Autotool infrastructure in the root directory