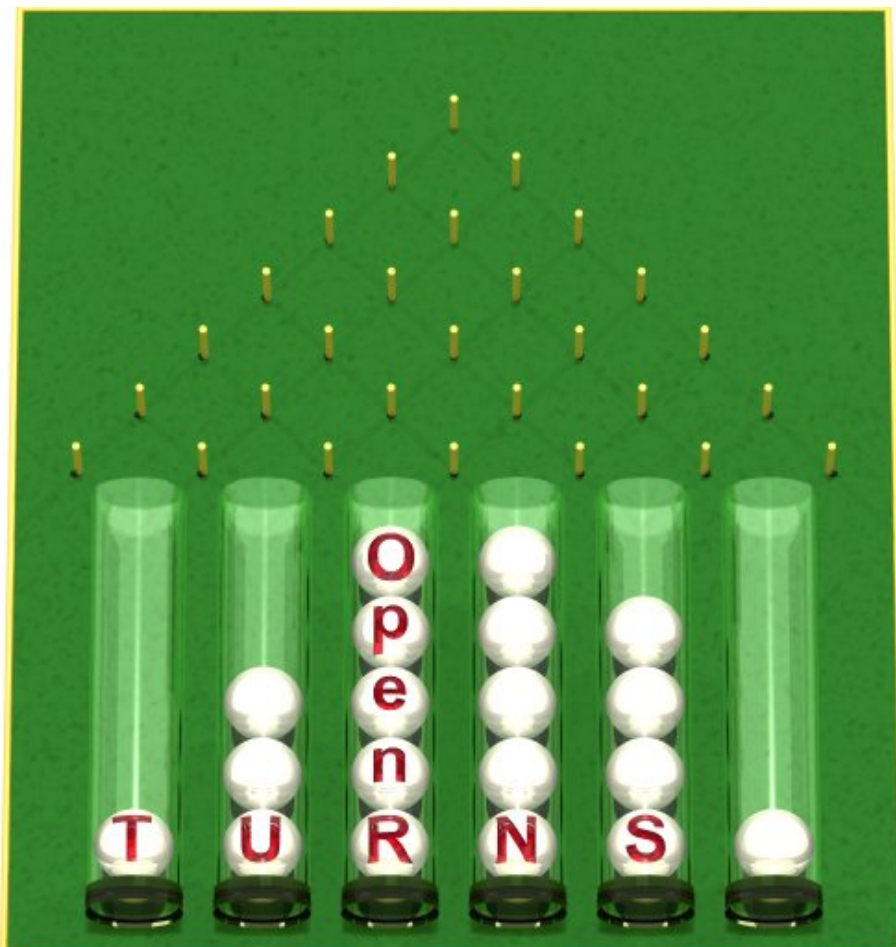


Windows Port Guide

Open TURNS version 0.14.0

Documentation built from package openturns-doc-June2011

June 30, 2011



Contents

1	Introduction	2
2	Linux side	3
2.1	Set cross compilation environment	3
2.1.1	MinGW	3
2.1.2	BLAS / LAPACK	3
2.1.3	Pthreads	3
2.1.4	dlfcn	3
2.1.5	libxml2	3
2.1.6	regex	4
2.2	Linux test Environment	4
2.2.1	WINE	4
2.2.2	Windows shared libraries	4
2.2.3	Advice	4
2.2.4	R	4
2.2.5	Python	5
2.3	Open TURNS compilation	5
2.3.1	Compilation	5
2.4	How to create the installer	6
3	Windows side	7
3.1	Install Open TURNS	7
3.2	Install Open TURNS with a non admin account	7
3.3	Open TURNS validation	8
3.4	Open TURNS compilation examples	9
3.4.1	Simple program	9
3.4.2	Wrapper	9
3.4.3	Dev-C++	10
3.4.4	Visual C++	10
3.4.5	Benchmark	10
4	Unresolved problems	10
4.1	Compilation of python modules	10
4.2	Compilation of Open TURNS wrappers	10
5	Resolved problems	11

1 Introduction

This documentation aims at guiding the developer with Open TURNS cross compilation for Windows target.

This documentation is separated into two main parts :

- compile Open TURNS under Linux for Windows target,
- validation and use of Open TURNS on Windows.

2 Linux side

2.1 Set cross compilation environment

2.1.1 MinGW

To install MinGW, it is recommended to follow the web page <http://www.mingw.org/wiki/LinuxCrossMinGW>

. It is the official supported procedure for MinGW compilation, and will install GCC version 3.4.5. It is recommended to use the installation scripts from the CVS repository.

2.1.2 BLAS / LAPACK

The BLAS / LAPACK library given by Open TURNS does not work with the MinGW compiler. So it is required to use an external library.

- download from a RedHAT server the mingw-lapack package, (e.g. <ftp://ftp.pbone.net/mirror/ftp.sourceforge.net/pub/sourceforge/m/project/mi/mingw-cross/OldFiles/mingw-lapack-3.1.1-5.fc9.i386.rpm>).
- transform the rpm to a tgz with the program alien.
- copy the library file libblas.dll.a and liblapack.dll.a to the lib directory of MinGW.

2.1.3 Pthreads

- download the binary of pthread for Windows here <http://sourceware.org/pthreads-win32>
- copy the library libpthreadGC2.a in the lib directory of MinGW.
- copy the headers in the include directory of MinGW, create an empty config.h file in the MinGW include directory.

2.1.4 dlfcn

- download the binary of dlfcn for Windows here :
<http://dlfcn-win32.googlecode.com/files/dlfcn-win32-shared-r11.tar.bz2>
- copy the library libdl.a and libdl.dll.a in the lib directory of MinGW.
- copy dlfcn.h in the include directory of MinGW.

2.1.5 libxml2

- download the precompiled zip files of iconv zlib and libxml2 from <http://sourceforge.net/projects/gnuwin32/files/> (at this time mingw32-iconv-1.12-7.zip, mingw32-zlib-1.2.3-11.zip and mingw32-libxml2-2.7.2-4.zip).
- install the content of include/ and lib/ directories of iconv and zlib into the respective directory of MinGW
- decompress libxml2 in a separate directory (e.g. : /opt/mingw32-sharedlib/libxml2).
- modify libxml2.la file so that the iconv dependancy becomes correct, e.g. :

```
replace the line :
dependency_libs=' -lz /usr/i686-pc-mingw32/sys-root/mingw/lib/libiconv.la -lws2_32'
by :
dependency_libs=' -lz /opt/mingw-3.4.5/lib/libiconv.la -lws2_32'
```

```
and the line :
libdir='/usr/i686-pc-mingw32/sys-root/mingw/lib'
by :
libdir='/opt/mingw32-sharedlib/libxml2/lib'
```

2.1.6 regex

- download mingw-libgnurx-2.5.1-bin.tar.gz and mingw-libgnurx-2.5.1-dev.tar.gz from the MinGW official website.
- decompress this two files in a same directory (e.g. /opt/mingw32-sharedlib/regex).

2.2 Linux test Environment

2.2.1 WINE

Install any WINE version (<http://www.winehq.org/>), for example, the one given by your Linux distribution.

2.2.2 Windows shared libraries

- put the shared libraries of pthreads (pthreadGC2.dll), BLAS/LAPACK (lapack.dll and blas.dll), dlfcn (libdl.dll), libxml2 (libcharset-1.dll libiconv-2.dll libxml2-2.dll zlib1.dll), and regex (libgnurx-0.dll) in a directory where the PATH environment variable of WINE points to.
- put the shared library given by MinGW (mingwm10.dll) in this directory too.

Note : WINE's PATH can be modified in the file `~/.wine/system.reg`.

2.2.3 Advice

It is better to install Open TURNS dependencies in directory without spaces (e.g. not in `C:\Program Files`). The space between *Program* and *Files* can cause cumbersome problems (mostly with autotools).

2.2.4 R

Ghostscript

- download and install ghostscript into WINE environment. The installer (e.g. gs864w32.exe) can be found here <http://sourceforge.net/projects/ghostscript/>. Launch the command like this : `wine gs864w32.exe`.
- add the path to gswin32c.exe to the PATH environment variable of WINE.

R

- install R into WINE environment by using the standard Windows installer from the official site <http://cran.r-project.org>.
- add the path to R.exe to the PATH environment variable of WINE.

R packages

- install the sensitivity and rotR zip files with Rgui.exe (menu packages => install R packages fom zip files).

The packages `sensitivity_1.3-1.tar.gz` and `rotRPackage_1.4.4.tar.gz` have been transformed to Windows packages with the website <http://win-builder.r-project.org>.

2.2.5 Python

- download the 2.6 version of Windows python installer from the official site <http://www.python.org>. Install python using this command : `wine msiexec /i python-2.6.2.msi` .
- add the path to Python.exe to the PATH environment variable of WINE.

PyQT module Install a PyQT version compatible with Python2.6 and QT4 (e.g. : `PyQt-Py2.6-gpl-4.5.1-1.exe`) from the following web site <http://www.riverbankcomputing.co.uk/software/pyqt/download>.

RPy module It is not necessary to install RPy into WINE because no ckecktest use it and furthermore RPy does not install properly into WINE. This section will only be for RPy on Windows.

- download RPy from the website <http://rpy.sourceforge.net/download.html>. At this time, RPy is compiled on Windows with python version 2.6.
- As described on the download page, install the RPy dependancies : `Pywin32 (pywin32-213.win32-py2.6.exe)` and `NumPy (numpy-1.3.0-win32-superpack-python2.6.exe)`.
- Install the following version : `rpy2-2.0.3.win32-py2.6.exe`.
The python version is fixed by the this RPy dependancy.

2.3 Open TURNS compilation

2.3.1 Compilation

In order to cross-compile Open TURNS :

First you need to adapt the source code for Windows :

```
cd ot-src
cd utils
./adapt_check_tests.sh win32
```

This script use the command `sed` in order to adapt the library prefix for Windows (e.g. : `.so` to `.dll`). If you want to restore the intial state of the source code to compile for Linux, you only have to launch :

```
./adapt_check_tests.sh unix
```

Then, get the type of your computer in order to set the `--build` configure settings :

```
export BUILD_MACHINE='gcc -dumpmachine'
```

The configuration step (you must have already bootstrap) :

```

# adapt these following lines to your configuration :
PYTHON_VERSION=26
PYTHON_PREFIX=$HOME/.wine/drive_c/Python$PYTHON_VERSION
R_PATH=$HOME/.wine/drive_c/R/R-2.9.0
REGEX_PREFIX=/opt/mingw32-sharedlib/regex
LIBXML2_PREFIX=/opt/mingw32-sharedlib/libxml2

# do not modify :
export PYTHON=$PYTHON_PREFIX/python.exe
export PYTHON_NOVERSIONCHECK="1"
export PYTHON_CPPFLAGS=-I$PYTHON_PREFIX/include
export PYTHON_LDFLAGS="-L$PYTHON_PREFIX/libs -lpython$PYTHON_VERSION"
export PYTHON_EXTRA_LIBS=" "
export PYTHON_SITE_PKG=$PYTHON_PREFIX/Lib/site-packages

# launch as is :
./configure --with-swig --with-R=$R_PATH --enable-R-renaming=R.exe --with-regex=$REGEX_PREFIX \
--with-libxml2=$LIBXML2_PREFIX --enable-debug=none CXXFLAGS=-O2 CFLAGS=-O2 FFLAGS=-O2 \
--build=$BUILD_MACHINE --target=i386-mingw32 --host=i386-mingw32 --prefix=$PWD/install

```

Debug symbols are deactivated so that binaries are 3 times smaller.

In the same shell, start the compilation :

```

# openturns compilation and installation
make; make install

```

The validation : launch the following command :

```

# set the PATH to python.exe
PATH=$PATH:$PYTHON_PREFIX

# Checktests needs a valid temp dir
mkdir -p $HOME/.wine/drive_c/OpenTURNS/tmp

make check && make installcheck

```

Verification (this part is optional, it is not currently working but could work) : in order to check that the sources are distributable, set the variable like described in the above paragraph and launch the following command :

```

make distcheck DISTCHECK_CONFIGURE_FLAGS="--with-R=$R_PATH --enable-R-renaming=R.exe \
--with-regex=$REGEX_PREFIX --with-libxml2=$LIBXML2_PREFIX \
--enable-debug=none CXXFLAGS=-O2 CFLAGS=-O2 FFLAGS=-O2 \
--build=$BUILD_MACHINE --target=i386-mingw32 --host=i386-mingw32"

```

2.4 How to create the installer

Two installers are created.

- setup.exe installs Open TURNS DLL and headers, and its dependancies. It is mainly for users that interact with Open TURNS through Python.

- `setup_developer.exe` helps compiling Open TURNS program and wrapper on Windows. Also, it permits to launch Open TURNS checktests.

NSIS is used to create Windows auto-installers. The command that creates the installers can be launched from Linux or Windows.

Before using NSIS, in directory `distro/windows/`, modify the line of `create_installer.sh` containing `OT_PREFIX` and, if needed, the line containing `WINOT_PATH` in the file `openturns_script.nsi`.

- `OPENTURNS_PREFIX` must point to the installation directory of Open TURNS.
- `WINOT_PATH` must point to the directory containing every Windows Open TURNS dependencies. An archive of these files will be available on Open TURNS' sourceforge website (`openturns_windeps.tgz`).

Finally, to create the `setup.exe` file, launch the command :

```
./create_installer.sh
```

3 Windows side

3.1 Install Open TURNS

The following steps are done automatically by the installer (`setup.exe`).

To install Open TURNS without installer :

- Copy the *install* directory (created by the command `make install`) from Linux to Windows into the directory `C:\openturns`.
- Like with WINE, every DLL must be reachable (`mingwm`, `pthread`, `BLAS/LAPACK`, `dlfcn`, `libxml2`, `regex` and `OpenTurns`), and the programs must be installed : R with its packages, `ghostscript`, Python with the required modules.

On Windows, DLLs are searched in directories listed in the `PATH` environment variable. To set the `PATH` variable temporarily, hit on a DOS console :

```
set PATH=%PATH%;C:\openturns\bin;C:\openturns\lib\bin
echo %PATH%
```

To set permanently the `PATH` variable : configuration panel -> system -> tab "advanced" -> button "environment variable" -> list "system variable" -> modify `PATH` variable.

3.2 Install Open TURNS with a non admin account

Use Open TURNS installer as usual and try to start python interpreter from a DOS command. If it does not start correctly, you probably need to install Visual C++ 2008 redistributable package from an administrator account.

Open TURNS developer installer can be used too if you installed Open TURNS in default directory. But MinGW and MSYS installation will need an administrator account too.

3.3 Open TURNS validation

To test Open TURNS on Windows,

- if you have the OpenTURNS developer installer (setup_developer.exe):

- Open TURNS should have been installed in default directory C:\OpenTURNS
- install Open TURNS developer with every checkboxes enabled.
- click on shortcuts : Start Menu -> OpenTurns -> Start-checktests.

- if you do not have the Open TURNS installer :

- install MinGW and MSYS
- install like with WINE : R with its packages, ghostscript, Python with the required modules.
- copy the *install* directory (created by the command make install) from Linux to Windows into the directory C:\openturns.
- suppress the empty file openturns\share\openturns\examples\libOT-0.dll (dead unix link).
- finally, from an msys shell, go to the examples directory

```
cd /c/OpenTURNS/share/openturns/examples/
```

and launch the checktests :

```
export PRINTF_EXPONENT_DIGITS=2
```

```
./check_testsuite AUTOTEST_PATH="$PWD" OPENTURNS_CONFIG_PATH="$PWD/../../../../etc/openturns"
```

```
export abs_srcdir="$PWD"
```

```
./installcheck_testsuite AUTOTEST_PATH="$PWD" \  
OPENTURNS_NUMERICALSAMPLE_PATH="$PWD" \  
OPENTURNS_WRAPPER_PATH="$PWD/../../wrappers" \  
OPENTURNS_CONFIG_PATH="$PWD/../../../../etc/openturns"
```

```
PYTHON_VERSION=26
```

```
export examplesdir="$PWD"
```

```
./python_installcheck_testsuite AUTOTEST_PATH="$PWD" \  
OPENTURNS_NUMERICALSAMPLE_PATH="$PWD" \  
OPENTURNS_WRAPPER_PATH="$PWD/../../wrappers" \  
OPENTURNS_CONFIG_PATH="$PWD/../../../../etc/openturns" \  
PYTHONPATH="$PWD/../../../../lib/python$PYTHON_VERSION/site-packages"
```

3.4 Open TURNS compilation examples

3.4.1 Simple program

Install MinGW from the official installer (provided by Open TURNS developers installer). During the installation, choose the compiler g++.

In order to compile, g++ needs Open TURNS headers and libraries. If Open TURNS is installed like this :

```
c:
'--openturns
|-- include
|   '-- openturns
|     '-- ...
|-- lib
|   |-- bin
|   |   |-- libOT.dll.a
|   |   '-- ...
'-- src
'-- mon_prog.cxx
```

From a DOS console, compile with this command :

```
cd src
g++.exe mon_prog.cxx -I..\include\openturns -L..\lib\bin -lOT -o mon_prog.exe
```

An example is given in the directory `openturns/share/openturns/examples/simple_example`.

3.4.2 Wrapper

To compile a wrapper on Windows, Open TURNS developers installer must be installed. An example of a wrapper for Windows can be found in the directory `openturns/share/openturns/WrapperTemplates/mingw_wrapper_linked_` (this example is installed by the developers installer).

In this directory, launch the compilation from an MSYS shell :

```
PATH=/c/MinGW/bin:$PATH
./bootstrap
mkdir build; cd build
../configure --with-openturns=/c/openturns --prefix="$PWD/install"
make && make install
```

The test using this wrapper can be started :

```
./start-test.sh
```

The test can be started too by using the `test.py` file.

The following paragraphs are automatically done by the Open TURNS developers installer :

- To compile a wrapper using Autotools, MinGW, MSYS and `msysDTK` must be installed. These installers can be found on MinGW site.
- Then, the Pthread library must be installed in MinGW directory (like in paragraph 2.1.3)

- Autotools wrappers scripts use the `openturn-config` command. These one is configured to be installed in `c:\openturns\bin` directory. If Open TURNS is not installed in this directory, the prefix variable of `openturns-config` must be modified.
- If during the compilation of the wrapper, `libtool` cannot produce a dynamic library because it can not found shared library, check that the library is existing and that the corresponding `.la` file is correct.

3.4.3 Dev-C++

Dev-C++ is an integrated development environment like Visual Studio.

Download the last Dev-C++ version. The compilations options are the same with those of paragraph 3.4.1.

Configure it so that it uses MinGW `g++ 3.4.5`. At this time, the linker fails with the Dev-C++ compiler (`g++ 3.4.2`).

3.4.4 Visual C++

Link a program compiled with Visual C++ with Open TURNS DLL is not possible. The C++ binaries produced by Visual C++ and `g++` are not compatible (C binaries are compatible). Further informations can be found here : <http://chadaustin.me/cppinterface.html>.

If you need to use only a small subset of the OpenTURNS C++ interface, it is possible to make an had-hoc MinGW wrapper that wrap Open TURNS C++ symbols to C symbols (C binaries are compatible between `gcc` and Visual C). The application compiled with Visual Studio will be able to interact with Open TURNS through the C symbols of the wrapper.

3.4.5 Benchmark

No benchmark of Open TURNS on Windows has been done.

4 Unresolved problems

4.1 Compilation of python modules

In order to compile the python wrappers (Python -> Open TURNS), the static library of python (`libpython.a`) must be included. `Libtool` does not allow to create DLL with static dependancies. Python wrappers are also created directly with the compiler without using `libtool`.

The symbols of the static library `libpython.a` are also in each python DLL. At Open TURNS execution, until now, no problems have been reported.

Furthermore, if `libtool` succeeds to create a DLL, it does it by appending a "-0" to the filename. This is a problem, because python searches for a filename without -0 (e.g. `_stat.pyd` and not `_stat-0.pyd`).

4.2 Compilation of Open TURNS wrappers

Open TURNS wrappers must be compiled on Windows with the `-LOT` flag.

On Linux, this could cause problems (at execution, the wrappers symbols could be loaded twice).

On Windows, the compilation of dynamics libraries is different : when a DLL is compiled, every symbols must be present (through a DLL skeleton).

I do not know how to avoid this flag.

This remains a point to take care of. The problem seen on Linux should be tested on Windows.

5 Resolved problems

- if *NumericalMathFunction* (*exec external*) check test fails :

The temporary-dir element of `openturns.conf` could be misconfigured.

If Open TURNS examples has been installed in a directory containing spaces, copy the files `poutre_external_infile*` to a directory without spaces and set the environment variable `abs_srcdir` to this directory.

- if DLLs or programs are not found :
check your MSYS or Windows PATH environment variable.
- if Open TURNS does not start from python interpreter and if the PYTHONPATH is correctly set :
check that the version of the python interpreter is the same as the version Open TURNS has been compiled for.
- if a program is installed in `C:\Program Files` and if it is not well detected,
reinstall it in directory without spaces in the name. The space between *Program* and *Files* can cause cumbersome problems (mostly with autotools).
- to modify the PATH variable of `.wine/system.reg`, no WINE processus must be started. When a WINE processus stops, it overwrites this files.
- libtool wrappers do not work on Ubuntu Intrepid. They work correctly on Mandriva 2009 32bits and Ubuntu Jaunty. If libtool wrappers do not work, modify `ot-src/lib/config/test.am` as described in this file.
- if the static libraries `libgc2.a` and `libfirtbegin.a` are included during the compilation, libtool produces only a static Open TURNS library. These libraries are also deactivated during cross-compilation.