

# Experiences of SWIG for cmtz library

*Martyn Winn*

July 2001 (using SWIG1.1p5)

Updated December 2001 (using SWIG-1.3.9)

Updated again December 2005

## Introduction

To quote the author: "SWIG is a code development tool that makes it possible to quickly build powerful scripting language interfaces to C, C++, or Objective-C programs". In this case, I wanted to build quickly (no split infinitives in my documentation!) a python interface to the C library for reading/writing/manipulating MTZ files. This is a record of my experiences.

This is a selective and subjective account which it is hoped will show the reader how to use SWIG in a real-life situation. It is no substitute for the SWIG User Guide. And it may well change as I discover my mistakes.

SWIG can be found at <http://www.swig.org/>. One should also look at the Boost Python Library (<http://www.boost.org/libs/python/doc/>) but I haven't. See this comparison (<http://www.boost.org/libs/python/doc/comparisons.html>).

## How to use it

1. Create an "interface file", which will be used to generate a C-language wrapper with functions that can be called from Python. Here is the simplest one for CMTZ:

```
/* File: cmtzlib.i */
```

```
%module cmtz
%{
#include "mtzdata.h"
%}
#include "cmtzlib.h"
```

%module specifies the name of the Python module to be created. %{ and %} bracket code that is to be included as is, in this case an include preprocessor directive that is necessary for the wrapper functions. %include includes a header file which lists the functions for which we want wrappers.

%inline can be used to define additional functions for the Python API which don't exist in cmtzlib.h, for example:

```
%inline %{
float refdata_get(MTZCOL *col, int i) {
    return col->ref[i];
}
void refdata_set(MTZCOL *col, int i, float f) {
    col->ref[i] = f;
}
%}
```

2. Run SWIG on the interface file:

```
swig -python cmtzlib.i
```

This generates a file `cmtzlib_wrap.c` which contains python-callable wrapper functions, based on the ones declared in `cmtzlib.h`. If you have a non-standard installation, you may need to set the `SWIG_LIB` environment variable.

3. Compile and include in shared library. On Linux:

```
gcc -c -I/usr/include/python2.4 -o cmtzlib_wrap.o cmtzlib_wrap.c
ld -shared --whole-archive libccp4c.a cmtzlib_wrap.o -o cmtzmodule.so
```

On SGI:

```
cc -n32 -c -I/usr/local/include/python1.5 -o cmtzlib_wrap.o cmtzlib_wrap.c
ld -n32 -shared -all libccp4c.a cmtzlib_wrap.o -o cmtzmodule.so
```

4. Tell python where to find the shared library:

```
PYTHONPATH=/blah/blah/cmtz
```

5. And you can now write your python script:

```
# File: mtzapp.py

from cmtz import *

mtz = MtzGet('tox.d.mtz')
ccp4_lhprt(mtz, 1)
MtzFree(mtz)
```

## So what now?

This is now all included in my Makefile so I should be able to forget about it. The wrapper `cmtzlib_wrap.c` should be updated as I update `cmtzlib.h` for the main library. The Makefile is specific to my version of SWIG and my version of python - I haven't thought about portability issues at all.

So now I do the same for tcl ....

## SWIG and C++

See <http://www.y.sbl.york.ac.uk/~mcnicholas/ccp4lib> for experiences of SWIG'ing MMDB. Here are some of my observations from trying to interface parts of Clipper:

1. The shared library needs to include all relevant system libraries, and some don't seem to be included by the linker by default (this is obviously system specific). For example, it took a while to find that `-lCio` is necessary. Even more obscurely, one needs

"/usr/lib32/c++init.o" to satisfy the symbol `_record_needed_destruction`. These details were revealed by comparison with "CC -v" on an equivalent program. So one ends up with:

```
ld -n32 -shared $(CLIPPYOBJS) -o CLIPPYc.so $(LDFLAGS)
/usr/lib32/c++init.o -lm -lC -lCio
```

If/when I find a neater way, I'll let you know ....

2. SWIG objects to "using" when read in via a %include directive.

## Some SWIG gotchas

1. I had some functions declared with arguments (char logname[], ...) which generated a TypeError with SWIG 1.3. Changing the arguments to (char \*logname, ...) seem to satisfy it.
2. Preprocessor directives:

```
#include"ccp4_lib.h"
```

with no space after the #include confused the SWIG preprocessor. Add a space!

3. If you have two functions:

```
CMMFile *ccp4_cmap_open( const char *, int);
void ccp4_cmap_header_print(const CMMFile *);
```

you should be able to call them in python as:

```
map = ccp4_cmap_open("toxd_aupatt_x.map", 0)
.
.
ccp4_cmap_header_print(map)
```

"map" is a pointer to a struct, but python doesn't need to know anything about this struct. EXCEPT: SWIG treats "CMMFile \*" and "const CMMFile \*" as two different data types, and one gets a TypeError.

My solution is to insert a "#define const " in the .i file, so that "const" is stripped from the wrapper function, although it still exists in the original function.

---

*m.d.winn@dl.ac.uk*

Last modified: Fri Jul 13 16:00:25 BST 2001