SWIG

Nick Thompson

June 16, 2015

Installation

```
sudo apt-get -y install libpcre3-dev byacc yodl
git clone --depth 1 https://github.com/swig/swig.git
./autogen.sh
./configure --prefix=/place/for/swig # or default: /
    usr/local/share
make -j8 && make install
```

Clone the examples

```
git clone https://github.com/NAThompson/
    SWIGExamples.git
```

Wrapping C is Easy:

```
cd wrap_pure_c;
make
./say_hello.py
```

Let's try to wrap some C++ without SWIG

```
cd wrap_cpp_wo_swip;
make
./say_hello.py
```

Name mangling is a pain!
Also, note how much the Python interpreter cares about your private methods!

Using Mangled Symbols in ctypes is Untenable!

Because the C++ type system is richer than ctypes can handle . . .

How would *you* design a wrapper script for interfacing Python and C++?

- C variables are passed by pointer or value.
- C++ variables can be pass by pointer, value, reference, and rvalue reference, and const qualified to your heart's delight.
- Python variables are passed by assignment (creation of references)

Design of SWIG: Hammer the C++ into C code the Python interpreter cam use

- Everything in Python is treated by the Python interpreter as a PyObject.
- A PyObject "is a type which contains the information Python needs to a pointer to an object as an object".
- So if we want to wrap a `bool foo()`, we create a new function returning `PyObject` of type `Py_True` or `Py_False` . . . which are themselves PyObject*.
- Generate code that we can use `extern ''C''` on; build a Python extension module.
- Will it work with Jython? (methinks no . . .)

SWIG Minimal Working Example

```
cd swig_mwe;
swig_mwe$ make
swig_mwe$ python3
>>> import is_even
>>> dir(is_even)
['__builtins__', '__cached__', '__doc__', '
    __file__', '__loader__', '__name__', '
    __package__', '__spec__', '_is_even', '
    _newclass', '_object', '_swig_getattr', '
    _swig_getattr_nondynamic', '_swig_property', '
    _swig_repr', '_swig_setattr', '
    _swig_setattr_nondynamic', 'is_even']
>>> is_even.is_even(5)
False
```

Deep Mystery

Why can we only call globally defined symbols from a shared object, and not locally defined symbols?

```
swig_mwe$ nm _is_even.so | grep ``_wrap_is_even \|
    _init_''
00000000000020c0 T _init
000000000000629c t _wrap_is_even
swig_mwe$ python3 -q
>>> import os
>>> from ctypes import cdll
>>> so = cdll.LoadLibrary(os.path.join(os.getcwd()
    , ``_is_even.so'')))
>>> is_even = so._wrap_is_even
Symbol not found
>>> foo = so._init
```

SWIG And C++ namespaces

```
cd swig_namespace;
swig_namespace$ make
swig_namespace$ python3
>>> import is_even
>>> dir(is_even)
['__builtins__', '__cached__', '__doc__', '
    __file__', '__loader__', '__name__', '
    __package__', '__spec__', '_is_even', '
    _newclass', '_object', '_swig      _getattr',
    '_swig_getattr_nondynamic', '_swig_property',
    '_swig_repr', '_swig_setattr', '
    _swig_setattr_nondynamic', 'is_even']
>>> is_even.is_even(5)
False
```

SWIG and a C++ namespace clash

“If your program utilizes thousands of small deeply nested namespaces each with identical symbol names, well, then you get what you deserve.“ –Swig documentation

```
cd swig_namespace_clash
swig_namespace_clash$ make
swig_namespace_clash$ python3 -q
>>> import is_even
>>> dir(is_even)
['__builtins__', '__cached__', '__doc__', '
    __file__', '__loader__', '__name__', '
    __package__', '__spec__', '_is_even', '
    _newclass', '_object', '_swig_getattr', '
    _swig_getattr_nondynamic', '_swig_property', '
    _swig_repr', '_swig_setattr', '
    _swig_setattr_nondynamic', 'bar_is_even', '
    foo_is_even']
>>> is_even.bar_is_even(4)
True
>>> is_even.foo_is_even(4)
True
```

How to wrap C++ templates

Python doesn't use templates; nor does C. C++ templates are turned into assembly code at compile time, iff there exists a template instantiation.
So you must instantiate the templates to interface C++ and Python.

```
cd swig_vector;
make
```