SWIG

Nick Thompson

July 21, 2015

Installation

```
sudo apt-get -y install libpcre3-dev byacc yodl
git clone --depth 1 https://github.com/swig/swig.git
cd swig
./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
```

Steps to Wrap C code

- Compile the C code into a shared object.
- Use ctypes to load the dll (.so)
- Tell the Python interpreter how to interpret the output

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

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

Steps:

- Compile the C++ into a shared object
- Determine the mangled symbol name using nm (which is not guaranteed to be consistent across compilers!)
- Use ctypes to interpret the return type (which might not be a primitive C type!)
- Call the function (which might segfault as C++ does not have a well-defined application binary interface)

But you can call private methods using this technique . . .

Using Mangled Symbols in ctypes is Untenable!

But note the `extern ''C''` trick which puts demangled symbols in the symbol table of the shared object.
Use this method if you can, but note the extern "C" rules (can't be in a namespace. . . infinitude of others that will cause cryptic linker errors.)

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 can 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 treat 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 . . .)

An example `PyObject`: `Py_True`

- On my machine, found definition found in: /usr/include/python3.4/boolobject.h
- Demonstrates use of garbage collection by reference counting: `Py_INCREF(Py_True)`
- Following through to the `PyObject`, we see that it is a struct which acts essentially as an abstract base class for all Python types (see object.h)

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
```

How does it work?

Please remain calm.

```
#ifdef __cplusplus
extern ``C'' {
  #endif
  SWIGINTERN PyObject *_wrap_is_even(PyObject *
      SWIGUNUSEDPARM(self), PyObject *args) {
    PyObject *resultobj = 0;
    int arg1 ;
    int val1 ;
    int ecode1 = 0 ;
    PyObject * obj0 = 0 ;
    bool result;

    if (!PyArg_ParseTuple(args,(char *)''O:is_even
        '',&obj0)) SWIG_fail;
    ecode1 = SWIG_AsVal_int(obj0, &val1);
    if (!SWIG_IsOK(ecode1)) {
      SWIG_exception_fail(SWIG_ArgError(ecode1),
          ``in method ''' ``is_even'' ``',
          argument `` ``1" of type ''' ``int"''');
        }
        arg1 = static_cast< int >(val1);
```

Steps:

- Compile the source into a shared object.
- Write a `foo.i` file, which tells swig what to wrap.
- Use swig to generate a *4000 line C++/C wrapper file* which demangles all symbols and casts everything into the appropriate inheritor of `PyObject`, as well as a Python script to load the module
- Compile the C++/C wrapper into a shared object (same name as the source .so, with leading underscore)

Obvious in retrospect . . .

You cannot compile your extension module linking against PythonX.Y and run it in PythonU.V! You code will work with one and only one Python interpreter, for each build.

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
```

Extern C and namespaces

- Since C doesn't have namespaces, we must tell swig how to call the C++ code, within a namespace, by appropriate editing of foo.i.
- Extern C can cause namespace clashes! (There was a reason for introducing this annoyance . . .)
- Namespace clashes caused by swig's use of extern C must be resolved by the programmer.

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)
```

SWIG and C++ namespace clashes

- Resolve namespace clashes in your shared object using the "rename" functionality in SWIG (see swig_namespace_clash/is_even.i).

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
```