SWIG

Information

What is SWIG?
Compatibility
Features
Tutorial
Documentation
News
The Bleeding Edge
History
Guilty Parties
Projects
Legal Department
Links
Download
SwigWiki
Donate

Affiliations

Software Freedom Conservancy

Our Generous Host

sourceforge

Exits

AllegroCL
C# - Mono
C# - MS .NET
CFFI
CHICKEN
CLISP
D
Go language
Guile
Java
Lua
MzScheme/Racket
Ocaml
Octave
Perl
PHP
Python
R
Ruby
Tcl/Tk

Home Development Mailing Lists Bugs and Patches

TUTORIAL

So you want to get going in a hurry? To illustrate the use of SWIG, suppose you have some C functions you want added to Tcl, Perl, Python, Java and C#. Specifically, let's say you have them in a file 'example.c'

```
/* File : example.c */

#include <time.h>
double My_variable = 3.0;

int fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}

int my_mod(int x, int y) {
    return (x%y);
}

char *get_time()
{
    time_t ltime;
    time(&ltime);
    return ctime(&ltime);
}
```

Interface file

Now, in order to add these files to your favorite language, you need to write an "interface file" which is the input to SWIG. An interface file for these C functions might look like this :

```
/* example.i */
%module example
%{
/* Put header files here or function declarations like below */
extern double My_variable;
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
%}

extern double My_variable;
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
```

Building a Tcl module

At the UNIX prompt, type the following (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -tcl example.i
unix % gcc -fpic -c example.c example_wrap.c \
       -I/usr/local/include
unix % gcc -shared example.o example_wrap.o -o example.so
unix % tclsh
% load ./example.so example
% puts $My_variable
3.0
% fact 5
120
% my_mod 7 3
1
% get_time
Sun Feb 11 23:01:07 1996

%
```

The `swig` command produces a file `example_wrap.c` that should be compiled and linked with the rest of the program. In this case, we have built a dynamically loadable extension that can be loaded into the Tcl interpreter using the 'load' command.

Building a Python module

Turning C code into a Python module is also easy. Simply do the following (shown for Irix, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -python example.i
unix % gcc -c example.c example_wrap.c \
       -I/usr/local/include/python2.1
unix % ld -shared example.o example_wrap.o -o _example.so
```

We can now use the Python module as follows :

```
>>> import example
>>> example.fact(5)
120
>>> example.my_mod(7,3)
1
>>> example.get_time()
'Sun Feb 11 23:01:07 1996'
>>>
```

Building a Perl module

You can also build a Perl5 module as follows (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -perl5 example.i
unix % gcc -c `perl -MConfig -e 'print join(" ", @Config{qw(ccflags optimize cccdlflags)}, \
       "-I$Config{archlib}/CORE")'` example.c example_wrap.c
unix % gcc `perl -MConfig -e 'print $Config{lddlflags}'` example.o example_wrap.o -o example.so
unix % perl
use example;
print $example::My_variable,"\n";
print example::fact(5),"\n";
print example::get_time(),"\n";
<ctrl-d>
3.0
120
Sun Feb 11 23:01:07 1996
unix %
```

Building a Java module

SWIG will also generate JNI code for accessing C/C++ code from Java. Here is an example building a Java module (shown for Cygwin, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
$ swig -java example.i
$ gcc -c example.c example_wrap.c -I/c/jdk1.3.1/include -I/c/jdk1.3.1/include/win32
$ gcc -shared example.o  example_wrap.o -mno-cygwin -Wl,--add-stdcall-alias  -o example.dll
$ cat main.java
public class main {
  public static void main(String argv[]) {
    System.loadLibrary("example");
    System.out.println(example.getMy_variable());
    System.out.println(example.fact(5));
    System.out.println(example.get_time());
  }
}
$ javac main.java
$ java main
3.0
120
Mon Mar  4 18:20:31  2002
$
```

Building a C# module

SWIG will also generate code for accessing C/C++ code from C# using PInvoke. Here is an example building a C# module (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems). It uses the open source DotGNU Portable.NET C# compiler which runs on most Unix systems, but works equally well using other C# compilers:

```
$ swig -csharp  example.i
$ gcc -c -fpic  example.c example_wrap.c
$ gcc -shared example.o  example_wrap.o   -o libexample.so
$ cscc -o runme *.cs
$ cat runme.cs
using System;
public class runme {
    static void Main() {
        Console.WriteLine(example.My_variable);
        Console.WriteLine(example.fact(5));
        Console.WriteLine(example.get_time());
    }
}
$ ilrun runme
3
120
Tue May 13 10:45:45 2003

$
```

SWIG for the truly lazy

As it turns out, it is not always necessary to write a special interface file. If you have a header file, you can often just include it directly in the SWIG interface. For example:

```
%module example
%{
/* Includes the header in the wrapper code */
#include "header.h"
%}

/* Parse the header file to generate wrappers */
%include "header.h"
```

Alternatively, some people might just include SWIG directives in a header file with conditional compilation. For example:

```
#ifdef SWIG
%module example
%{
#include "header.h"
%}
#endif

extern int fact(int n);
...
```

Running SWIG under Microsoft Windows

SWIG also works perfectly well under all known 32 bit versions of Windows including 95/98/NT/2000/XP. SWIG is typically invoked from the command prompt and can be used with NMAKE. Modules are typically compiled in the form of a DLL that can be dynamically loaded into Tcl, Python, or whatever language you are using. With a little work, SWIG can also be used as a custom build option within MS Developer Studio.

That's it (well, more or less)

That's about everything you need to know to get started. Here's the short checklist :

- Make sure you specify a module name.
- Use ANSI C/C++ syntax
- Figure out how to compile a shared library module / dynamic link library (may require reading a few man pages for

- Figure out how to compile a shared library module / dynamic link library (may require reading a few man pages for your compiler).
- Relax.

Surely there's more to it...

The above example is intentionally simple, but the general idea extends to more complicated C/C++ programming tasks. In fact, it is important to know that SWIG is a fairly complete C++ compiler with support for nearly every language feature. This includes preprocessing, pointers, classes, inheritance, and even C++ templates. SWIG can also be used to package structures and classes into proxy classes in the target language---exposing the underlying functionality in a very natural manner.

To illustrate, suppose you wanted to wrap the following C++ data structure:

```
// pair.h.  A pair like the STL
namespace std {
   template<class T1, class T2> struct pair {
       T1 first;
       T2 second;
       pair() : first(T1()), second(T2()) { };
       pair(const T1 &f, const T2 &s) : first(f), second(s) { }
   };
}
```

To wrap with SWIG, you might specify the following interface:

```
// pair.i - SWIG interface
%module pair
%{
#include "pair.h"
%}

// Ignore the default constructor
%ignore std::pair::pair();

// Parse the original header file
%include "pair.h"

// Instantiate some templates

%template(pairii) std::pair<int,int>;
%template(pairdi) std::pair<double,int>;
```

Now, compiling (Python):

```
$ swig -python -c++ pair.i
$ c++ -c pair_wrap.c -I/usr/local/include/python2.1
$ c++ -shared pair_wrap.o -o _pair.so
$ python
Python 2.1 (#3, Aug 20 2001, 15:41:42)
[GCC 2.95.2 19991024 (release)] on sunos5
Type "copyright", "credits" or "license" for more information.
>>> import pair
>>> a = pair.pairii(3,4)
>>> a.first
3
>>> a.second
4
>>> a.second = 16
>>> a.second
16
>>> b = pair.pairdi(3.5,8)
>>> b.first
3.5
>>> b.second
8
```

Feedback and questions concerning this site should be posted to the swig-devel mailing list.

Last modified Sat Mar 26 09:55:45 2011

Last modified : Sat Mar 26 09:55:45 2011