July 12, 1996

beazley@cs.utah.edu

Salt Lake City, Utah 84112
University of Utah
Department of Computer Science

David M. Beazley

Scoping Languages with C and C++

SWIG: An Easy to Use Tool for Integrating
Topics

- What is SWIG?
- Background
- Applications
- Limitations
- Future directions
- A quick tour
SWIG (Simplified Wrapper and Interface Generator) compiles ANSI C/C++ declarations into bindings to interpreted languages and currently supports Tcl, Python, Perl5, Perl4, and Guile3. It supports most C/C++ datatypes, simple C++ classes, multiple files and modules, run-time type checking, and automatically generates documentation. SWIG also supports most C/C++ datatypes and interfaces to interpreted languages, automatically generates documentation, and currently supports Tcl, Python, Perl5, Perl4, and Guile3.
Where am I coming from?

50-100 Gbytes of Data
Lousy Tools
3 years of frustration

Dave's Attitude Problem

Perl
Python

Tcl/Tk

SWIG Prototype

Dave's Scripting Language

Guile

Ease of Use

Big FLOPS

50-100 Gbytes of Data

Ease of Use

SWIG

Where am I coming from?
The Two Language Model

- Two languages better than one
- C/C++ (performance, number crunching, etc...)
- Tcl (control, debugging, modules, user interface)
- C++ (performance, number crunching, etc...)

Tcl Wrapper Function

Unfortunately, need to write "wrapper" functions

C Function

\[
\text{int wrap_fact(ClientData clientData, Tcl_Interp *interp, int argc, char *argv[])}
\]

\[
\begin{aligned}
\text{if (argc != 2) } & \text{ return TCL_ERROR;} \\
\text{arg0 = atoi(argv[1]);} & \\
\text{result = fact(arg0);} & \\
\text{sprintf(interp->result, "%d", result);} & \\
\text{else return TCL_OK;} & \\
\end{aligned}
\]
**Automatic Wrapper Generation**

- Most languages have tools to generate wrapper code
- Usually only support a single target language
- Often use non-C syntax
- Usually only provide a parser and primitives. Allow everything else to be redefined.

**SWIG Design Goals:**

- Provide a parser and primitives. Allow everything else to be redefined.
- Use ANSI C/C++ syntax
- Ease of use and flexibility
- Language independence
- Special purpose

**Most languages have tools to generate wrapper code**
SWIG Overview

- Interface file with ANSI C/C++
- Generic parser
- Target languages implemented as C++ classes
- Easy to extend
- Produces C++ source file as output
- Generic parser
- ANSI C/C++
A Simple Example

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

```swig
%module fact
{ /* put header files here */
  %module fact
}
```

```unx>
unix> swig -tcl fact.i
unix> gcc -c fact.c fact_wrap.c -I/usr/local/include
unix> ld -shared fact.o fact_wrap.o -o fact.so
unix> tclsh7.5
    % load ./fact.so
    % fact 6
    720
}
```

```c
extern int fact(int);
```
Datatypes

• All C/C++ built-in datatypes
  int, short, long, char, float, double, void

• C/C++ pointers (used for everything else)
  int, short, long, char, float, double, void

• All C/C++ built-in datatypes

Pointers are type-checked at run-time.

Pointers are represented as character strings

Vectors: new Vector(doubte x, doubte y, doubte z)

---

SWIG 1.996 Tcl/Tk Workshop
Most "typical" C declarations can be handled.

- Wrapping a C/C++ function
  \[
  \text{double } \text{foo}(\text{double } a, \text{int } b, \text{void }* \text{ptr});
  \]

- Linking with a global variable
  \[
  \text{extern int Status;}
  \]

- Creating constants
  \[
  \text{#define MY_CONST } 5
  \]

\[
\text{typedef MY_CONST 5}
\]

- Most "typical" C declarations can be handled.

- Wrapping a C/C++ function
  \[
  \text{extern } \text{Vector }* \text{transform}(\text{Matrix }* \text{m}, \text{Vector }* \text{v});
  \]

- Linking with a global variable
  \[
  \text{int MyClass::bar(}\text{double }\text{);}
  \]

- Creating a constants
  \[
  \text{extern int Status;}
  \]

- Wrapping a C/C++ function

- Functions, Variables, and Constants
SWIG and C++

- Single public inheritance
- Constructors, destructors, and virtual functions
- Simple C++ classes and structs
- C++ politely turned into C and wrapped (for now)
Controlling and Debugging C/C++

SWIG provides direct access to C/C++ variables, functions, constants, and classes.

SWIG works particularly well in research applications.

- Requires minimal or no code modifications.
- Can control existing C/C++ applications at a function level.
- Provides direct access to C/C++ variables.
- Functions, constants, and classes.

Tcl makes a great debugger.

- Easily add Tcl/Tk to programs without worrying about messy details.

Controlling and Debugging C/C++
Rapid Prototyping

• Use SWIG for prototyping and experimentation

Sample code:

```c
set light_ambient [newfv4 0.0 0.0 0.0 1.0]
gLightfv $GL_LIGHT0 $GL_AMBIENT $light_ambient
...glClear $GL_COLOR_BUFFER_BIT
...auxSolidTorus 0.275 0.85
```

Example: OpenGL module

Total development time: > 20 minutes
< 8000 lines of wrapper code
708 constants
426 functions
Developed from OpenGL header files (gl.h, glu.h, aux.h)

USE SWIG FOR PROTOTYPING AND EXPERIMENTATION
• SWIG can be used to build highly modular and programmable applications.

Don’t build a huge monolithic system---build components and reuse them.

• Tcl/Tk

• Data Analysis

• MATLAB

• SPaSM Molecular Dynamics Code

Building Modular Applications

SWIG can be used to build highly modular and programmable applications.
Language Independence

- All languages have strengths and weaknesses
- SWIG interface files are language independent
- All languages have strengths and weaknesses
- Language independent code re-use!

Example:

Perl script to generate web statistics

Uses MATLAB module developed for Tcl/Tk
Current Limitations

- C++ support is incomplete (well, I think so).
- No exception model
- Pointer model is extremely flexible, yet incompatible with most other Tcl extensions (i.e., file handles).
- No variable or optional arguments
- Numerical representation problems (particularly unsigned integers and longs)
- No exception model
- C++ support is incomplete (and will probably always be so)
Conclusions

Much work remains!

SWIG has proven to be remarkably reliable and powerful in a variety of applications.

Different applications have different needs. There is no "best" scripting language. Language independence is essential!

Provides a direct mapping onto C/C++ (and a reasonable subset of C++)

Particularly well-suited for research applications.
Future Work

- Using SWIG to do cool stuff
- Simplified extension mechanism
- More complete C/C++ parsing
- An exception model
- Java?
- ILU
- Object Tcl
- HCL
- Support for more target languages

SWIG 2.0
Acknowledgments

DOE, NSF, NIH

The Advanced Computing Laboratory (LANL)

The Scientific Computing and Imaging Group

John Schmidl, Kurtis Bleeker, Patrick Fullmann

John Schmidt, Kurtis Bleeker, Patrick Fullmann

German, Niels Jensen (LANL)

Peter Lombardi, Shujia Zhou, Brad Holian, Tim

John Buckman (non-unix platforms)

and provided feedback

Users who have braved the first few releases

Accknowledgements
SWIG is free and fully documented.

Source code and user manual are available via


The SWIG homepage:
http://www.cs.utah.edu/~beazley/SWIG

The SWIG mailing list:
swig@cs.utah.edu