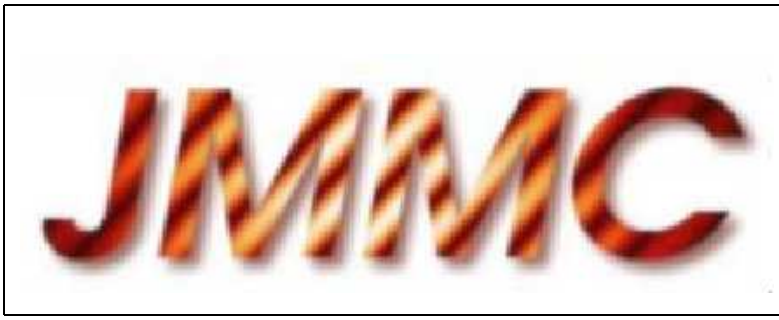

JMMC-MEM-2910-0001

Revision 1.0

Date: 28/05/2004

JMMC

Génération d'interface C/C++ avec des langages interprétés

Authors:
Thierry Stein <Thierry.Stein@obs.ujf-grenoble.fr> — LAOG/JMMC

Author: Thierry Stein Institute: LAOG/JMMC	Signature: Date: 28/05/2004
Approved by: Gérard Zins Institute: LAOG/JMMC	Signature: Date: 28/05/2004
Released by: Guillaume Mella Institute: LAOG/JMMC	Signature: Date: 17/05/2004

Table of contents

1 Introduction

L'objet de ce document est de présenter le logiciel SWIG[1]. SWIG est un logiciel libre développé depuis 1995 par des informaticiens venus de tous les horizons. Il permet d'interfacer des programmes C et C++ avec des langages interpretés (Tcl, Python...), ou compilés (Java). Toutes les informations relatives à SWIG sont disponibles sur le site http://www.swig.org. Quand au logiciel, il est téléchargeable (bien évidemment gratuitement) sur le site http://sourceforge.net.

Pour utiliser SWIG, il faut lui fournir en entrée un fichier interface (.i ou .swg) (voir section 2), et lui préciser le langage dans lequel on souhaite qu'il génère l'interface. Il crée alors un fichier *_wrap.c ou .cxx, compilable. Puis, après la phase de compilation et d'édition de liens, il crée un fichier .so, qui permet d'utiliser les fonctions du fichier source dans le langage que l'on a préalablement choisi.

Supposons que l'on souhaite interfacer des fonctions écrites dans un fichier source nommé example.c. Les commandes unix pour créer un module Tcl avec SWIG sont les suivantes :
unix > swig -tcl example.i
unix > gcc -c -shared example.c example_wrap.c -I./
unix > gcc -shared example_wrap.o -o example.so
unix > tclsh
% load ./example.so

Une fois la librairie shared chargée, vous pouvez utiliser les fonctions situées dans le fichier source example.c.

Le logiciel s'accompagne d'un grand nombre d'exemples d'applications pour l'ensemble des langages qu'il connait. Je me suis pour ma part intéressé aux exemples d'interfaçage avec Tcl (ceci est juste une question de commodité, le principe de SWIG restant le même quelque soit le langage de sortie choisi). La majorité de ces exemple se trouvent dans le module *SWIG-1.3.2*, dans le répertoire */Examples/tcl*. Ils seront détaillés dans la section 3.

[1] Simplified Wrapper and Interface Generator

2 Le fichier interface

Le fichier interface est la base de travail du logiciel. SWIG peut interfacer la grande majorité des fonctionnalités du C++, à la seule condition que le fichier interface contienne les bonnes informations.

La forme de base du fichier est la suivante :

```
%module nom_module
%{
#include "fichier.h"
}%
// Déclarations de variables et méthodes C/C++
int dumbo;
int voler(int dumbo);
...
```

Ce fichier peut contenir le nom du (ou des) fichier source C ou header à interfacer, mais ce n'est pas obligatoire. Il peut contenir de plus :

- Des définitions de types ou structures
  Elles sont nécessaires car sinon, SWIG ne saura pas traiter les structures.
- Des entêtes de fonctions (définies dans le fichier source)
  SWIG n'interfacera que les fonctions dont les entêtes se trouvent dans le fichier interface. Ceci permet de réduire le nombre de méthodes que l'on souhaite interfacer. Dans le cas des classes C++, SWIG interface l'ensemble des méthodes publiques de la classe.
- Le code source de fonctions supplémentaires
  L'utilisateur peut choisir de rajouter une ou plusieurs fonctions, comme dans l'exemple de la section 3.4
- Des macros similaires au préprocesseur C
  Voir à ce sujet la section 3.3.

3 Exemples d'applications

SWIG fourni des exemples d'applications en Tcl pour les fonctionnalités C et C++ suivantes :

1. Simple
2. Variables
3. Constants
4. Value
5. Pointer
6. Funcptr
7. Multimap
8. Class
9. Operator
10. Enum
11. Reference

On trouvera à la suite quelques exemples complémentaires :

- Template
- La librairie liblog

Nous allons brièvement détailler tous ces exemples dans la suite de ce document.

3.1 Simple

Ce premier exemple présente l'interfaçage et l'utilisation d'une variable globale (Foo), ainsi que de la fonction simple pgcd.
Le fichier "example.i" ne comprend pas de #include. Il contient juste la déclaration de la variable et l'entête de la fonction.
L'interface crée en Tcl permet de déclarer des variables x et y, d'appeler la fonction pgcd avec ces deux variables en paramètres, et d'afficher le résulat. On peut accéder au contenu de la variable global Foo, modifier sa valeur et l'afficher.

3.2 Variables

Cet exemple contient :

- Un fichier header contenant la définition de la structure point
- Un fichier C contenant :
  - Un certain nombre de variables globales couvrant l'ensemble des types prédéfinis du C
  - Deux variables globales de type point
  - Une variable déclarée en lecture seule dans le fichier interface
  - Une fonction affichant le contenu de l'ensemble de variables
  - Une fonction créant un entier
  - Une fonction créant un point
  - Une fonction affichant le contenu d'un point
- Le fichier interface inclut le fichier header, et contient les déclarations de toutes les variables globales. Certaines de ces variables sont déclarées en lecture seule grace à la macro spécifique à SWIG **%immutable** :

  %immutable;
  extern int status;
  extern char path[256];
  %mutable;

Cet exemple montre que l'on peut effectuer une affectation à une variable globale de la manière que l'on souhaite, c'est à dire soit dans le code C, soit avec l'interface Tcl. De plus, on vérifie bien que les variables déclarées en lecture seule ne peuvent pas être modifiées dans l'interface tcl.

3.3 Constants

Cet exemple contient simplement un fichier interface, ce qui montre que SWIG peut se passer d'un fichier header ou source C pour travailler. Le fichier interface contient des déclarations de constantes similaires à celle du préprocesseur C. On constate que ces déclarations acceptent des entiers, des réels, des chaînes de caractères, un retour chariot :

#define FCONST 2.1828
#define CCONST 'x'
#define CCONST2 '\n'
#define SCONST "Hello World"

Elles n'acceptent pas des mots clés, ni de constante non définis :

#define EXTERN extern

3.4 Value

Cet exemple défini une structure vecteur, ainsi que deux fonctions : addition de deux vecteurs, et produit scalaire. L'intérêt de cet exemple est double:

- Le fichier interface contient les déclarations de fonctions supplémentaires: initialisation directe d'un vecteur, et écriture d'un vecteur, comme l'exemple ci dessous :

  void vector_print(Vector *v) {
  printf("Vector %x = (%g, %g, %g)\n", v, v->x, v->y, v->z);
  }

- Le passage des paramètres dans les fonctions se fait pas valeur :

  Vector vector_add(Vector a, Vector b)

3.5 Pointer

Cet exemple montre l'utilisation de variables de type pointeur. Ces variables peuvent être des variables d'entrée, de sortie, ou bien les deux.

Cet exemple montre de plus l'utilisation de la librairie typemap de SWIG, principalement utilisé pour les pointeurs. Celle ci permet de déclarer des variables de type pointeur comme étant des variables d'entrée (*INPUT) ou de sortie (*OUTPUT) :

%include typemaps.i
extern void sub(int *INPUT, int *INPUT, int *OUTPUT);

La librairie typemaps permet de plus de passer des valeurs en paramètres lorsque ce sont des pointeurs qui sont attendus.

3.6 Funcptr

Cet exemple montre l'utilisation de fonctions dont l'entête est passé en paramètre. Concrètement, on dispose d'une fonction nommé "do_op", contenant en paramètre une variable "*op", qui pointe sur le nom de la fonction que l'on souhaite utiliser. Les référence aux fonctions sont définis dans le fichier interface.

Seule restriction apparente : les fonctions que l'on souhaite appeler de cette manière doivent avoir les mêmes types d'entrées et de sortie.

3.7 Multimap

Cet exemple met en évidence l'interfaçage de fonctions traitant des chaînes de caractères. Elle montre l'utilisation d'une fonction convertissant les lettres minuscules en majuscules, ainsi qu'une fonction comptant le nombre d'apparitions d'une lettre dans une chaîne de caractères. Elle montre aussi l'utilisation de la fonction pgcd en lui passant en paramètre une chaîne de caractères. Ceci se fait en utilisant la librairie typemap de SWIG, qui dispose de nombreuses fonctions pour traiter les chaînes de caractères.

3.8 Class

Cet exemple montre l'interfaçage par SWIG des classes C++.

Concrètement, il montre la création d'un objet, l'affectation de valeurs à des variables membres, l'utilisation de méthodes, et l'appel du destructeur. Le fichier interface néccessite juste l'inclusion du fichier header, dans lequel se trouve la définition de la classe et l'entête des méthodes.

3.9 Operator

On dispose d'une classe "Complexe", défini dans un fichier header. On a redéfini dans cette classe les opérateurs classiques d'addition, soustraction, multiplication, et décrémentation. On a de plus défini plusieurs opérateurs de création. SWIG n'a aucun problème pour gérer ces opérateurs.

On constate de plus que si un des opérateurs est déclaré "private", il n'est pas accessible dans l'interface Tcl.

3.10 Enum

Cet exemple met en évidence les types énumérés. Le type couleur {RED, BLUE, GREEN } est déclaré en global. Le type speed {IMPULSE, WARP, LUDICROUS } est déclaré dans une classe Foo. On peut les utiliser avec des fonctions ou avec des méthodes définies dans une classe.

3.11 Reference

Cet exemple illustre l'utilisation des références C++ en Tcl. Ondispose d'une classe Vector, dans laquelle est redéfini l'opérateur addition. Celui ci est paramètré avec deux vecteurs dont le passage se fait par réferénce.

On a rajouté dans le fichier interface la fonction suivante, pour simplifier l'utilisation de l'opérateur :

```
%inline %{
Vector addv(Vector &a, Vector &b) {
   return a+b;
   }
%}
```

3.12 Template

Cet exemple montre un interfaçage simples de méthodes et de classes templates. J'ai effectué ce test à partir du fichier situé dans le répertoire */Examples/java/template*.

On dispose de la définition suivante :

```
template<class T> T max(T a, T b) { return a>b ? a : b; }
```

Pour interfacer cette classe, il faut écrire dans le fichier interface les lignes suivantes :

```
%template(maxint) max<int>;
%template(maxdouble) max<double>;
```

Ces fonctions deviennent ensuite directement utilisable en tcl avec la commande :

```
% maxint 2 3
```

3.13 La librairie liblog

On peut sans difficulter interfacer les fonctions de la librairie liblog[2]. Pour cela, il suffit d'écrire le fichier interface suivant :

```
%module myliblog
%{
#include "log.h"
%}
typedef char mcsPROCNAME[mcsPROCNAME_LEN+1];
extern mcsCOMPL_STAT logIdentify(const mcsPROCNAME processName);
```

On peut ainsi créer un fichier "myliblog.so", permettant d'utiliser via Tcl la fonction logIdentify. Par contre, il est nécessaire de définir le type mcsPROCNAME, sinon SWIG ne saura pas comment le traiter.

[2] développée par Guillaume Mella

4 Conclusion

L'ensemble de ces exemples permet d'avoir un aperçu du potentiel de SWIG. La version actuel permet d'interfacer la quasi-totalité des fonctionnalités du C++ . Il ne subsiste que quelques restrictions, répertoriées sur la page web officielle du logiciel. Par exemple, la déclaration suivante est supportée par SWIG :

extern const int Nombre;

Celle ci n'est par contre pas supportée :

const int extern Nombre;

Globalement, un programme écrit de manière propre, selon la syntaxe standard ANSI C, ne posera aucun problème à SWIG.