Universal Interface Generator

Vadim Zeitlin

ITO33 SA, 39 rue Lhomond, 75005 Paris, France, eMail: Vadim@ito33.com

Introduction

Writing quality high performance libraries is a demanding endeavor and dictates a number of choices at the engineering level. One of them is the choice of the programming language and to achieve the best compromise between ease-of-development and the performance, C++ is often used. However, C++ may not always be the best choice for rapid applications development or integration with the existing back-end infrastructure.

This explains why it may be interesting to provide the library clients with the possibility to use the libraries from any language of their choice. Some popular ones are Java or C# for rapid development of the GUI applications or C for squeezing the last ounce of performance. To achieve this goal and to simplify creating wrappers for the new languages which may become useful in the future (such as various .NET languages or scripting languages such as Perl or Python) we have created the *universal interface generator* called *C*++2Any.

This tool can be used to create, fully automatically, the wrappers allowing an existing C++ library to be used from almost any other language. It is capable of reading the existing C++ source files and produces the necessary glue to allow the library functionality to be used across the language boundary. The interface generator is not tied to a particular library nor a particular target language and so provides a very flexible solution for the library users who wish to work with it from their favorite language or development environment.

In this article, we are going to briefly describe how *C*++2Any works and how it can be used and customized. Please see [1] for more details about *C*++2Any.

How Does It Work?

C++2Any has two categories of input files: first of all, it reads the standard C++ header files, possibly with some special comments, containing the classes and functions to generate the wrappers for. But to make the tool as flexible as possible, its output format is not fixed but is described by the *template files*. The formal documentation of the template files can be found in [2] but, briefly, they are just simple text files with the text which should appear on output and special directives which instruct the generator to place here the declarations of all the exported classes, there the declarations of their methods and so on. It is also possible to specify how exactly all of those declarations are going to look like and so on.

The template files for several target languages are included with *C*++2Any so there is no need to write, or even understand, them to use the tool. However they do provide a very simple way to fine tune its output to any specific needs. All appearance aspects of the generated code may be changed simply by editing the templates.

The other aspects—i.e. what code is generated on output—cannot be changed at the templates level as they are governed by the generator back-end used. For example, the generator knows that `vector<int>` C++ type must be represented as an array of integers in the output file and the templates cannot change this. However due to a modular *C*++2Any architecture it is very easy to write new back-end plug-ins. This simplifies not only adding support for new languages to the generator but can also be used to inject into the generator knowledge about some specific types and classes used in a particular library. To return to the example of `vector<int>`, it is quite possible in practice that the library being wrapped uses some other, non standard, containers such as `Matrix<double>`. With a custom type handler plug-in it is possible to teach generator to treat such matrices as well as vectors and generate proper wrappers for all functions taking matrices as parameters or returning them.

To summarize, *C*++2Any aims to be as flexible as possible and provides several ways to change its standard behavior without compromising its ease of use. This is one of its main advantages in comparison to other similar tools such as [3].

Using *C*++2Any

In order to use *C*++2Any you must first prepare the C++ header files. Although *C*++2Any should usually be capable of parsing most of the existing declarations, it is impossible for the tool to realize which classes and methods should be exported and which should be not. So, at a minimum, the classes which don't have to be exported must be marked with a special comment. This must be done only once, before the first *C*++2Any run.

Once the header files are properly annotated, you can run the tool. It is a command line application which has quite a few options but in the simplest case it accepts just the input C++ file names and the output format. Currently the possible choices for the format are C, Java, COM and Excel. The latter two are slightly special and so, maybe, merit an explanation.

COM output means that the tool generates the necessary glue code to wrap the C++ library as a COM[1] object. After compiling the tool output (and linked with the support files) you obtain a COM DLL which can be used from practically any programming language under Windows platform. Thus, the advantage of COM wrappers is their universality—but only under one operating system. For cross platform libraries using C or Java API is probably a better solution.

Excel back-end is an even more narrowly specialized format but one which may be extremely useful, especially for the financial libraries. The output in this case are the declarations of the Visual Basic for Applications functions which can be called directly from the spreadsheet by simply choosing the appropriate function from the "Add In" Excel menu.

It is perhaps difficult to appreciate how much work would be normally involved in making a C++ class work as a COM component, Excel add in, Java class and C library after reading this simple description, but in practice the time savings are enormous. And among the features of *C*++2Any we haven't mentioned are the automatic help generation (using Doxygen comments in the C++ code), support for exceptions and error reporting, automatic parameter validation in unsafe languages such as C or VBA, and much more.

Summary

We have presented *C*++2Any, a tool for effortless creation of wrappers for C++ libraries, which may be used to increase the appeal of C++ libraries by allowing to use them from any language and not just C++. We emphasized the tools flexibility and ease of use and briefly described its usage. Please visit the *C*++2Any home page for additional information about it!

FOOTNOTE & REFERENCES

1. also known as ActiveX.

[1] C++2Any Home Page, `http://www.tt-solutions.com/products/cpp2any/`
[2] "C++2Any Templates Manual", `http://www.tt-solutions.com/products/cpp2any/templates.pdf`
[3] Simplified Wrapper Interface Generator, `http://www.swig.org/`