Dynamic libraries: maintaining a single source for Unix & Windows

Valery Fine (Faine) [1],

Joint Institute for Nuclear Research, 141980 Dubna, Russia

The main problems are caused by the fact that Windows Dynamic-Linked Libraries (DLL s) assume some specific information to be introduced in the source code directly, while the UNIX Dynamic-Linked share Libraries don t need any "special information" in the source.
To provide a minimal effort to maintain a common source for both types of operating systems a set of the utilities was created.
These utilities, Makefile and CMZ macros are used to build the Windows NT versions of ROOT system, the CERNLIB Program libraries, and some application for the Atlas experiment and freely available via `http://root.cern.ch` Web site.

Key words: UNIX; Windows; DLL; ROOT; library;

1 Introduction

Our 4 years experience (since 1993,Aug) with porting of CERN Program library (CERNLIB) to Windows NT (for Alpha and Intel chips)/ Windows 95 platforms has shown clearly that the new generation of the personal computers and cheap Workstations with Windows NT/Windows-95 offers a powerful combination to the HEP user and/or programmer and represent a significant advance in the use of PCs in HEP.

At present all of these were done by creating the so-called static object modules libraries. But to enjoy the advantages the modern Object-Oriented and Client/Server technologies offer us, the next step "Creating the Dynamic-Linked Libraries (DLLs)" must be done.

[1] Thanks ROOT development team, Masahura Goto, Andres Sandoval

It allows to enhance reusing of the software tools and libraries. Therefore it is essential to make them available on systems running as UNIX as Windows NT and/or Windows 95 operating system. It is assumed that our users do not want to support multiple versions of these libraries, and do want to minimise the amount of conditionally compiled code.

David G.Korn[1] found six basic strategies been employed to port existing UNIX based applications to Windows NT. In his article he investigates those allow us to run programs on both UNIX and Windows NT based systems.

That proves that creating a common program source base for two different kinds of platforms is not trivial but still possible.

2 WIN32 features

Having started to use the Microsoft Software Development Kit (SDK) for Windows/NT system, a HEP (UNIX) user would rapidly encounter a significant hurdle. That's because of the Microsoft's extension of the base languages (and byy many other reasons too). For instance the "standard" C/C++ define the term "scope" as "File scope", "Class scope", "Local scope", "Prototype scope" etc. Miscorsoft extends this term and implemented let's say the "DLL scope" via the "attribute" mechanisme. The Visual C/C++ documentation says: "The *thread, nake, dllimport* and *dllexport* attributes storage-class are property only the declaration of the data or function to which they are applied. The *thread* attribute affects **data** only. The *nake* attribute affects **function** only. The *dllimport* and *dllexport* attributes affect **function** and **data**". This means DLL entry points and DLL global data **are the elements** of the Microsoft extension of the `C/C++/Fortran` **languages** and must be defined on the source level! To exploit the features of the WIN32 DLLs, the programmer must introduce a special keywords in his source code and create a module definition file to minimise the source code corrections. It must be mentioned that SDK has no tool to build DEF files and even has no full description of its format[2].

After looking at all the alternatives, we decided to write our own utility to create DLLs as with Microsoft SDK for Windows NT and Windows 95 as for UNIX from the single source code.

In general the problem looks as follows.

Even for Windows OS only one needs 3 versions of one and the same include files `(*.h)`:

(i) Export - to be used by calling routines
(ii) Import - to be used by called routines
(iii) "Normal" - to be used to build static libraries

2 kinds of source files `(*.c, *.cpp, *.for)`

(i) To build DLL
(ii) To build a "static" executable

And even for a single platform like Windows one needs a solution because Microsoft (and other vendors of PC tools) doesn't supply it.

The programmer must choose what he has to do "by hand"

- Apply the control macro-names
- Generate so-called "module-definition" files
- Use both approaches

In real life this entails the only way is adopted. Namely developers support either DLL approach or create a static libraries/executable. Only few companies are rich enough to supply just both things as DLLs as static libraries. In fact there are few reasons to support both type of libraries for Windows only. If it is possible the DLLs are created otherwise static ones.

To optimize the amount of conditionally compiled code we decided to use the both approaches. This needed to solve two things:

- no tool is supplied to create that "module definition file"[2] `(DEF)`.
- The format of `DEF` file can not define the IMPORT references. This means there is no way to decide whether some module needs a global pointer from another DLL but the source of this module using the Microsoft extension of C/C++.

The first goal is to create the DEF file form the object files in COFF[3] format directly.

It is possible to avoid the second problem with the external variables (and make maintaining of code easy). For that one should modify the source C/C++ code to exclude referencing the global constants and pointers across DLLs. It is always possible by implementing some very simple access functions. It brings in the extra advantages since it permits to synchronise the access to those pointers for multi-thread applications for both platforms in the question as UNIX as Windows.

3 Creating WIN32 DLL's

Creating the application DLLs needs two passes (see picture). With first one the programmer uses the special utility BINDEXPLIB to created a `DEF` and the Microsoft librarian `LIB` to create the export and import libraries. One needs to fulfil the first step for each DLL one wants to make up. Once all import/export libraries done it is possible to go to the second step to build the DLLs and the executable files. This step needs NO extra tool but LINK supplied with Microsoft SDK.

4 Single program source for UNIX and Window

Putting all things together we can summarise that to build and to maintain the common program base for two kinds of the operating systems one has to perform several steps as follows (it is assumed that the original programs were written with C++ for UNIX):

– Check whether any of your desired DLLs intents to access the global variables of another one. All definitions of these external variables must be changed. We advice to apply a pre-processor macro as follows:

```
/*
  This include file defines DllImport/DllExport macro
  to build DLLs under Windows OS.

  They are defined as dummy for UNIX's
*/
#ifndef ROOT_DllImport
#define ROOT_DllImport

#if defined(WIN32) && defined(_DLL)
#  define DllImport     __declspec( dllimport )
#  define DllExport     __declspec( dllexport )
#else
#  define DllImport
#  define DllExport
#endif
#  define EXTERN       DllImport extern


#endif
```

and replace all `extern` keywords with the uppercase `EXTERN`. If one wants to access the global variable of `C` code from the `C++` code one has to define it with `extern "C"` additionally. Namely, the piece of the `C` code

```
  extern int SomeCplusVar;
```

will look as

```
#include "DllImport.h"
extern "C" {
  EXTERN int SomeCplusVar;
}
```

- Provide the access method for the static data-members. Access from one DLL the the static data-members of the C++ classes of another DLL can be done with static access method only. So it must be provided.
- Build the set of the dynamic libraries for some project with two passes as shown on diagrams.

5 Conclusion

The present approach is used to build the dynamic and share libraries of the ROOT project. It includes the 20 DLLs built of 250 C/C++ source code files for many UNIX platforms (HP UX, Linux, Sun Solaris, DEC OSF/1, CGI etc) and for Windows NT and Windows 95 as well.

The utility (the `C` source code and binary), makefiles and CMZ macros are available to down-load via `ROOT` Web site as follows: `http://root.cern.ch`

References

[1] David G.Korn. "Porting UNIX to Windows NT", AT&T Laboratories Murray Hill, N.J. 07974, 1997.

[2] Exporting Data from a DLL or an Application. Microsoft knowledge base. PSS ID Number: Q90530

[3] Gintaras R. Gircys "Understanding and Using COFF". - O'Reilly & Associates Inc., 1988.