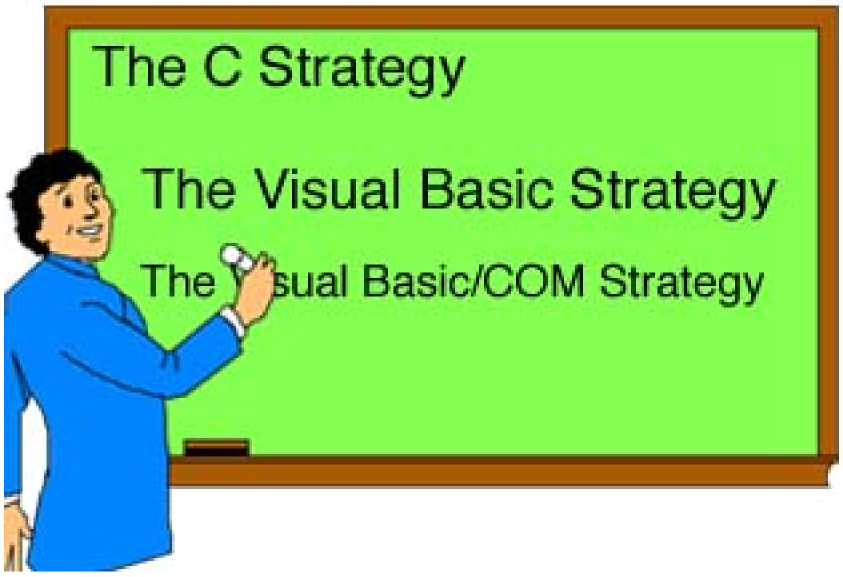
Crossroads
The ACM Student Magazine

Este artículo tambíen está en Español.

Microsoft Windows Programming Strategies

by *Mike Maxim*

Overview



Software developers are constantly in search of tools to make application programming easier. For developers working with the Microsoft Windows operating system, this task has traditionally been a complicated and often frustrating experience. The inherent complexity of handling all that must be handled by a program running under Windows prohibits a single development strategy from satisfying all needs. This is why the volatile world of Windows programming demands many different development strategies. The three options presented below focus on C and Visual Basic, but are by no means the only ways to program under Windows. Different languages such as Java and Pascal can also be used for Windows programming. It is important for the reader to keep in mind that even though the methods differ, the ultimate goal for each method is the same. That goal is to express application logic with the right balance of convenience and flexibility. This article presents three unique strategies in an attempt to guide the reader to the easiest Windows development solution.

The C Strategy

The fundamental way to program for Windows is to program directly to the operating system's interface using C or C++. This interface is commonly known as the Application Programming Interface (API). The API is a set of function calls the operating system provides in order for the programmer to interface programs to the particular environment. For example, in the Windows (Win32) API, the function `CreateWindow()` serves to create a window, and `ShowWindow()` displays it on the screen. Using C to program to this API gives the programmer the most control over the program's behavior. Of the three strategies presented, the C approach gives the programmer the most control, insuring that no unintended run-time libraries or dependencies will slow down performance. Only the minimum set of run-time libraries will be used, and executable files will most likely be small. In addition, C insures the program will run very fast compared with other high level languages. A desirable consequence of using this method is a deeper understanding of Windows internals since all work is done at the API level.

However, the C language approach is hard to learn and to implement. There are thousands of API calls and each one has many subtleties. The format all C programs must be in is awkward because of the messaging system Windows uses to notify programs of events. A giant and cumbersome ``switch-case'' block typically processes the messages Windows sends [3]. Programming in C++ is difficult as well because the mechanism for receiving messages from Windows does not allow C++ class functions to be used. Debugging also is a problem since it is very easy to forget to free critical resources thus crashing Windows. The inherent liberty the programmer gets when using this method entails responsibility and a greater demand for perfect code.

Microsoft Visual C++ is an excellent tool to use, in a C/C++ development strategy, due to its practical, easy-to-use, integrated, development environment (IDE). Another inviting feature of Visual C++ is a class hierarchy named the Microsoft Foundation Classes (MFC). MFC allows C++ programmers to program for Windows in a more object-oriented system and attempts to hide some of the complexity of the Win32 API. MFC provides an additional option for C++ programmers to pursue if they wish, however, the author believes a strong understanding of the complete API will help serve a programmer far into the future. A strong grasp of the API is best reached by learning to code straight to the API with limited library support. Many C++ programmers are put off by MFC because of the awkward nature of its ``document/view'' architecture and the questionable quality of MFC's object-oriented design [3]. Programmers who first started programming for Windows in the days of Windows 1.0

used the direct API system [3].

The Visual Basic Strategy

Visual Basic (VB) belongs to a category of development systems called Rapid Application Development (RAD). Other RAD tools that have much of the same functionality as VB include Borland Delphi and IBM Visual Age. Most of the features of VB mentioned in this article have counterparts in other RAD tools. VB will be the subject of discussion since it is impossible to cover every platform. The RAD classification implies that creating programs in VB is quick and easy. VB is indeed the easiest way to make computationally light programs for Windows and is gaining much popularity, particularly in the Information Technology field. With VB, any programmer can create an elaborate Graphical User Interface (GUI) in a fraction of the time it would take in C. This stems from the fact that the VB programming paradigm has the programmer create user interface objects and write code pertaining to these objects. One of the greatest strengths of VB is the intuitive way to design windows [3]. VB ships with a ``form editor'' which greatly aids in the design of windows and menus. The form editor allows a programmer to simply draw items such as text boxes, buttons, and data-bound grids, and write code associated with them. In addition to the form editor, VB also enables programmers to simply check boxes to add many different controls to their forms. Examples of controls include status bars, toolbars, progress bars, and many other useful tools that can be tricky to incorporate in C. The method of simply drawing and clicking a GUI into action is the main attraction to VB.

As far as database support for business applications is concerned, VB is hard to beat. VB provides many data access wizards and controls that make connecting to services like Microsoft SQL Server and Oracle database systems a breeze. In addition, VB simplifies the creation of Web-based programs by incorporating excellent support for ActiveX Controls and Microsoft's Internet Information Server [2]. Microsoft is pushing VB as the primary development tool in its DNA (Distributed interNet Applications) business development strategy, and justifiably so because VB makes developing multi-tier, scalable business applications efficient and practical [2]. The Visual Basic language is much simpler than C to learn and write, which enhances programmer productivity.

However, the fact that most of the work is already done for the programmer means different run-time libraries will greatly influence the speed and size of the program. VB programs must be distributed with the VB run-time Dynamic Link Library (DLL) that does not come standard with Windows. The Visual Basic language itself is not nearly as comprehensive and powerful as the C language. Visual Basic code is not as structured as C, which means Visual Basic code can quickly get ugly. Direct API access through VB is difficult because the language makes it tricky to access DLLs written in C. VB code is also slower and larger than C code for the same program

The Visual Basic/COM Strategy

The ideal strategy for the Windows programmer would be to combine the strengths of C ++ and Visual Basic into one glorious development solution. Unfortunately, until recently there has not been a practical means for accomplishing this goal. However, with the emergence of Microsoft's Component Object Model (COM) such a solution has become possible. Using COM, programs can easily be coded in multiple languages in order to utilize the best attributes of each language. With COM, a programmer can write speed sensitive algorithms in C++ and compile them into a COM component. These components can then plug into any application in any language that supports COM. Inserting COM components into VB projects is as easy as checking a box. This approach combines the efficiency and structure of C++ with the easy Windows GUI design of VB into one comprehensive solution.

For example, a programmer has written a library of functions in C++ that serves the purpose of playing chess, and wants a GUI on which to display the game. Rather than converting all the code to Visual Basic, or using the API method, or even making a standard C DLL, the programmer should opt to use COM. The reason for this is simplicity and speed in development. Most of the simplicity derives from the VB programmer only needing to check a box in order to add the COM chess component and use its functions. COM provides a more object-oriented system since all COM components are essentially classes that have member procedures and member variables [1]. A VB program would access a COM component by declaring a variable associated with the component and accessing different members in it using the dot (.) operator (e.g., component.ChessMove()). Another point about COM is that it defines its own data types. COM defines data types internally so all data conversion between VB's String and Integer data types to BSTR and INT is done automatically by COM [1]. The function prototypes that are often necessary in C DLLs are never required in VB, simplifying the programmer's job.

At first glance this system may seem to be the panacea of Windows programming. Unfortunately, there are many issues surrounding this method that decrease its value. First, COM is difficult to learn and execute. In the C++ environment (NOT C), Microsoft provides a class template library named Active Template Library (ATL) so a programmer does not need to learn the COM API (a C programmer must code to the COM API). ATL is still hard to use and may seem like too much trouble for some programmers. The problem of many dependencies only gets worse, as the vendor must ship DLLs for the COM runtime, COM components, and possibly ATL.

Recommendations

Deciding on a particular programming environment depends heavily on the type of application being developed. If the application involves database operations, Web interfaces, and other business tasks, then Visual Basic is an ideal choice. Keep in mind, COM components are easily created in VB and are encouraged by Microsoft in the development of distributed applications. Do not think the only use of COM is for joining C++ and Visual Basic into one. COM is very useful in building multi-tier business solutions that demand network and database capability [1]. COM is a broad and difficult subject to learn and the reader is encouraged to learn as much as possible about it. If your application demands fast algorithms and natural logic design while maintaining a first-rate GUI, then the Visual Basic/COM approach should be applied. COM provides the combination of language independence while VB provides easy GUI creation and integration of COM components. This approach guarantees speed and practicality. Lastly, if you are the type of programmer who wants everything under their control, then the API approach is the choice. No other method (save coding Windows in assembly) gives the programmer more control over the operation of his program than does the C/API method. The programmer must realize however, that everything is hand-coded and could take substantially longer to write than a solution designed in the VB or VB/COM approach. This is especially important for business and database intense work. I strongly discourage the C/API method for any program that is planned to access remote databases and perform Web-based tasks. The work required is extreme and is not worth the time.

References

1

Grimes, Richard. *Professional ATL COM Programming* Wrox Press, Birmingham, Alabama, 1998.

2

Pattison, Ted. *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0.* Microsoft Press, Redmond, Washington, 1998.

3

Petzold, Charles. *Programming Windows, Fifth Edition.* Microsoft Press, Redmond, Washington, 1999.