Microsoft® Visual C++™, 32-Bit Edition:
Most Commonly Asked Questions
July 2, 1993

For more information contact:

Microsoft Corporation
Eric Lang, (206) 882-8080

Waggener Edstrom
Martin Middlewood, (503) 245-0905

General/Policy Section

When will your final NT product be ready?

Visual C++ 32-Bit Edition final release will be available within 90 days of the release of Windows NT.

Does Visual C++ 32-Bit Edition really make it easier to learn C++?

Yes. The most difficult aspect of C++ programming is designing classes. Because Visual C++, 32-Bit Edition includes the Microsoft Foundation Class Library version 2.0, you don't have to design a class library from scratch. Microsoft Foundation Class Library version 2.0 provides a high level framework that you can "fill in" and extend with application specific code. To get started you can fill in the framework by writing C code. Then, as you get a feel for classes and Object Oriented Programming (OOP), you can begin to design your own classes to extend the framework.

The next most difficult aspect of C++ programming is learning the new syntax for working with classes (class declarations, constructors, and destructors). AppWizard and ClassWizard get you started by creating new classes with complete constructors and destructors for you. You can be immediately productive and then learn C++ by example and extension as you go.

How much disk space does Visual C++, 32-Bit Edition require?

Visual C++ 32-Bit Edition will require 80MB for complete hard disk installation. It can also be run directly from the CD-ROM, in which case only 6MB is needed on your local hard disk.

How much RAM does Visual C++, 32-Bit Edition require?

Visual C++ 32-Bit Edition requires a host computer running Windows NT version 3.1, with a minimum of 16 megabytes of RAM. 20 megabytes is recommended for improved performance.

By choosing Visual C++, 32-Bit Edition, will I be restricting available choices with respect to other or future development tool alternatives?

To further boost programmers' productivity, we have designed Visual C++, 32-Bit Edition to be a highly integrated system with support for additional tools through the Tools menu. We are supporting also tools standards, and producing standard output formats, such as the CodeView OMF, and the Windows NT .EXE format. We are promoting and supporting the notion of reusable code so that developers can develop applications faster and with less work. With these new tools, standards, and reusable components available from MICROSOFT and other vendors, we think that programmers in greater numbers will be able to develop richer solutions than ever before.

SDK Tools

Does Visual C++ 32-Bit Edition include all of the SDKs for both Windows and Windows NT?

Visual C++ 32-Bit Edition includes all of the Win32 SDK tools needed to build 32-bit applications for Windows NT. It also includes Win32s, a product that allows Win32 applications for Windows NT to run binary-compatibly on Microsoft Windows 3.1.

How do I override the system paths that point to the Win32 SDK?

If you installed the Win32 SDK for Windows NT while logged in as Administrator, the SDK Setup program modifies your System (rather than User) environment variables. If you then install Visual C++ for Windows NT at the user level, path conflicts will occur, since system environment variables are set before user environment variables.

To clear the appropriate system paths:

1. Log onto Windows NT as Administrator.
2. Start the REGEDT32 program.
3. Select the HKEY_LOCAL_MACHINE window.
4. Expand the following sub-folders in order:
    1. System
    2. CurrentControlSet
    3. Control
    4. Session Manager
    5. Environment
5. Remove the path entry that points to the Win32 SDK executable files by double-clicking the "Path" string in the right-hand split window. The String Editor dialog box appears. Delete only the section of the path string which points to the Win32 SDK subdirectory. The section to delete will look similar to the following:

    %SystemRoot%\mstools\bin;

Do not delete any other sections of this path string. When you are finished, choose OK.

6. Remove the entry that points to the Win32 SDK library files by double-clicking the "Lib" string in the right-hand split window. Again, the String Editor dialog box appears. Clear the contents of this dialog box and choose OK.
7. Remove the entry that points to the Win32 SDK include files by repeating step #5 for the "Include" string.
8. Close REGEDT32. This saves the new registry entries.
9. Restart Windows NT to use the new paths.

How do I move my Win32 SDK source files to Visual C++?

In the vast majority of cases, no changes need to be made. You'll need to make changes only in the following special situations:

- Use the double-underscore keywords **__try**, **__except**, **__finally**, and **__leave** instead of try, except, finally, and leave, or else add your own **#define** statements.

  The SDK header file EXCPT.H defines try, except, finally, and leave as their double-underscore counterparts, which are the actual keywords. These definitions have been removed from Visual C++ for Windows NT, to avoid conflicts with the ANSI specification.
- In header files, use the push and pop syntax when you use **#pragma** pack.

  The compiler issues a warning when a header file changes pack size, but header files from the Win32 SDK turn this warning off. The Visual C++ header files use the push and pop feature to save and restore the existing pack size and do not turn this warning off. As a result, if you have a header file that changes default pack size, this warning may appear.

To avoid this warning, use **#pragma** pack(push, *identifier*, *n*) and **#pragma** pack(pop, *identifier*) instead of **#pragma** pack(*n*) and **#pragma** pack(). See *C Language Reference* for more information on these pragmas.

- Examine code that uses the **localeconv** function, checks for the return value of CHAR_MAX, and is compiled with the /J command-line option. With Visual C++ for Windows NT, if you use the /J compiler option, **localeconv** will return CHAR_MAX as 255 instead of 127.

  The **localeconv** function from the Win32 SDK returns an error value 127, independent of the /J command-line option, which makes unsigned characters the default. In Visual C++ for Windows NT, the /J command-line option will adjust the return value to match the new value of CHAR_MAX.

How do I use my Win32 SDK Makefiles with Visual C++?

Update your SDK makefiles and then run the makefiles from the command line or as an external project in Visual Workbench. In general, the following changes need to be made:

- Remove calls to the CVTRES and CVTOMF utilities.

  These are tools in the SDK which convert .RES files (output from RC) and 32-bit OMF object files (typically from MASM) to COFF format for the linker. These utilities are obsolete, because the Visual C++ linker can link the .RES and OMF .OBJ files directly.
- Replace any occurrences of CRTDLL.LIB with MSVCRT.LIB.

  In the SDK, the DLL version of the C run-time library is supported with the import library CRTDLL.LIB. In Visual C++ for Windows NT, the import library and DLL are MSVCRT.LIB and MSVCRT*nn*.DLL, where *nn* is the version number of the DLL (currently 10).
- Remove references to non-Intel platforms.

  Visual C++ for Windows NT generates code for Intel-compatible processors only. The following command-line options are not supported: /Zh, /QmipsG1, /QmipsG2, and /Gt. (The /Gt option supports 32-bit applications for MIPS processors and 16-bit applications for Intel processors, but not 32-bit applications for Intel.)
- For DLLs with no entry point, add the /noentry option to the LINK command line.

  With the Windows NT SDK, if a DLL is linked without the /entry option, the DLL will be created with no entry point. In Visual C++ for Windows NT, the default for DLLs is to use the entry point _DllMainCRTStartup, which will automatically initialize the C run-time library.
- Optionally, remove the trailing "32" and "386" from LINK and CL.

  With Visual C++ for Windows NT, the "stub" utilities LINK32 and CL386 execute LINK and CL, respectively, and print a warning message. The actual linker and driver have been renamed to LINK and CL, but the stub utilities are included for makefile compatibility. You can slightly reduce your build time by calling the actual utilities directly.

Compiler

What are the major differences between Visual C++ 32-Bit Edition and the SDK compiler and utilities?

With few exceptions, the compiler, utilities, and run-time library support the same behavior. However, the Visual C++ utilities have new features for better usability and error detection. For example, the Visual C++ compiler supports the /G5 option for generating Pentium code. For information on new features, see Help on individual utilities. For specific changes you might need to make, see other questions in this category.

Visual C++ for Windows NT provides a complete set of well-tested and documented tools for the development of Win32 and Win32s applications. You can write most applications without the Win32 SDK, but in a few cases you may need to use the Win32 SDK and Visual C++ together. The table below outlines some differences between the tool sets.

Table of Win32 SDK and Visual C++ Tools

Win32 SDK	Visual C++
API Profiler	Source Profiler
BIND	EDITBIN /BIND
CVTRES	Incorporated into LINK
CVTOMF	Obsolete
Dialog Editor	App Studio
Image Editor	App Studio
Font Editor	None
WinDbg	Visual Workbench and CodeView for Win32s
M Editor	Visual Workbench
MASM	Inline assembler
Microsoft Test 1.0	None
NTSD	Visual Workbench
OLE 2.0 Toolkit (Beta)	None
RPC toolkit	None
REBASE	EDITBIN /REBASE
Setup Toolkit	None
Spy	SPY++
SYMEDIT	None
Working Set Tuner	Source Profiler

Microsoft compilers have been slow in the past. How does Visual C++ 32-Bit Edition perform.

Our compile and link speeds are very competitive. We have a white paper that goes into benchmarking issues in detail.

How can I modify the header of a 32-bit executable file?

EDITBIN.EXE lets you modify the header of a 32-bit .EXE, .DLL, or .OBJ file. To examine 32-bit executable files, use the DUMPBIN tool. EDITBIN is described in \MSVCNT\HELP\EDITBIN.WRI, and DUMPBIN is described in *Build Tools User's Guide*.

These tools replace the /edit and /dump options from the SDK linker.

Does Visual C++, 32-Bit Edition support templates? Does Visual C++, 32-Bit Edition support the 3.0 language standard?

This question really boils down to whether or not Visual C++, 32-Bit Edition supports templates. We considered this feature seriously and believe that, while there is a lot of market hype about templates, they aren't what C++ programmers need most today. For the majority of users, concerned with learning C++, templates make the learning issues more complex. Our goal in building Visual C++, 32-Bit Edition was to make moving to C++ and Microsoft Windows NT programming easier by offering streamlined integration, powerful reusable components, and wizards that perform the most complex tasks for you. We believe, however, that the ANSI standard is important. We will add support for templates to the compiler in the future when more customers indicate that this is high priority for them.

Resource Tools

What is App Studio?

App Studio provides complete windows resource editing. The most important advantage is that App Studio in combination with ClassWizard lets you connect user interface objects (forms, dialogs, menu items, accelerators, etc.) to code. One of the unique advantages of Visual C++, 32-Bit Edition over other systems is its streamlined integration. When you open a resource file in the Visual Workbench, it can be automatically opened into App Studio. When you compile an application in the Visual Workbench, any changes made to the project's resource file are automatically saved. This kind of integration, provides a high level of overall productivity.

- Copy and move resources, controls, menu items via drag and drop between resources or files.
- Search by prefix to find resources/symbols, instead of having to scroll through lists.
- Most recently used access to previously opened files.
- Toolbar.
- Context sensitive help by pointing to objects.
- ClassWizard and App Studio allow you to select user interface objects, browse their messages and connect them to code.
- Automatic linking to application framework features such as menu prompt strings, command ID management, toolbar editing.
- Open multiple files via MDI.
- App Studio will read any Microsoft .RC file.
- Automatic symbol value management (picks unique symbols, or allows English names)

- F7 will automatically size to content on any button, check box, or static text.
- Preview graphics before opening in editor.
- Greater graphics editing including custom brushes; direct manipulation cropping and scaling; editing tiled bitmaps; flip vertical/horizontal; invert colors' transparent and opaque drawing.
- Direct manipulation menu editing.
- View all strings in a string table in a single view.
- Search for string by ID or content.
- Select multiple strings.

Which SDK tools has App Studio replaced?

The tools that are no longer necessary, and have therefore been removed, are ImageEdit and DlgEdit. The other tools are still included.

What's different in the 32-bit edition of App Studio?

App Studio for Windows NT has the following differences:

- VBX custom controls are not supported
- App Studio for Windows NT cannot load and save Windows NT executable files (.EXE), dynamic-link libraries (.DLL) or resource files (.RES).
- App Studio for Windows NT does not support resources or resource objects specific to Windows NT.
- App Studio for Windows NT stores information about initialization and configuration within the configuration registry instead of APPSTUDIO.INI.

For more information, see “App Studio Update” in *App Studio User's Guide.*

Does App Studio let me work on my RC files without modification and keep my comments and conditionals?

AppStudio may not save some information defined in an RC file (i.e., comments, pre-processor directives, or macros), however, it will warn the user in this case.

Can App Studio manage resources in multiple resource files?

The App Studio Set Includes feature allows you to structure resource files any way you need to. It fully supports nested include files, multiple header files, and other features needed for sophisticated projects. App Studio also supports the notion of a "Compile Time Only" resource, which allows you to define resources that will not be edited by App Studio. This gives you full control of resource definitions that require macros or embedded preprocessor directives (features not supported by App Studio). For large projects with hundreds of resources you want to be able to selectively edit resources in specific files. In App Studio you can open the specific resource file you want to edit. When you open a resource file all included files are always opened as well.

Visual Basic

Can I use *all* Visual Basic Custom Controls with Visual C++, 32-Bit Edition?

Visual C++, 32-Bit Edition, is a 32-bit product, and as a result, does not support the use of existing 16-bit Visual Basic controls. The 32-bit architecture for Visual Basic controls is still under development. Visual C++ 32-Bit Edition will be updated to support the 32-bit control architecture when it becomes available.

Targeting (DOS, OS/2®, NT™ and Macintosh®. Also, 386, 486, P5)

Can I build 32-bit applications from MS-DOS?

Yes. All of the command-line tools provided with Visual C++ for Windows NT are MS-DOS extended. They can be run directly from the MS-DOS command prompt and do not require a separate DOS extender or memory manager.

Before building a 32-bit application, you should to set the INCLUDE, LIB, and PATH environment variables to point to the directories containing the 32-bit versions.

How do I build 16-bit applications from Windows NT?

If you have both Visual C++ for Windows, Version 1.0 and Visual C++ for Windows NT installed on your machine, you can develop 16-bit Windows and MS-DOS applications from the Visual Workbench or from the command line.

The BETA2FIX utility modifies the command-line tools from Visual C++ for Windows to run under Windows NT. You can also convert them back so that they can be used from Windows 3.1 and MS-DOS.

To make the command-line tools from Visual C++ for Windows available under Windows NT:

1. Make sure that BETA2.DLL is in your path.
2. From the Windows NT command line, move to the directory containing 16-bit Visual C++ executable files (by default, \MSVC\BIN).
3. Enter BETA2FIX /V *.EXE

To build 16-bit applications from the Visual Workbench:

1. Convert the command-line tools as described above.
2. Start the Visual Workbench from Visual C++ for Windows NT.
3. From the Options menu, choose Directories. Update the paths to point to the 16-bit version of each.
4. From the Project menu, choose Open. Select the "Use as an external makefile" option and load your project makefile.
5. Build and execute the program as usual.

Note Visual C++ for Windows NT cannot debug 16-bit executable files. In addition, context-sensitive keyword help does not work

To make the command-line tools from Visual C++ for Windows available under Windows 3.1 again:

1. From the Windows NT command line, move to the directory containing 16-bit Visual C+

+ executable files (by default, \MSVC\BIN).
2. Enter BETA2FIX /U *.EXE

BETA2FIX.EXE and BETA2.DLL are available from the following sources:

- CompuServe: Microsoft Windows NT download section
- Internet: FTP from UUNET.UU.NET under /vendors/Microsoft
- Microsoft Developer Network (MSDN) CD-ROM: Fall 1993 and later
- Microsoft Download Service (MSDL): (206) 936-MSDL in the United States (contact local subsidiary for other countries).

Can I convert 16-bit projects into 32-bit projects?

Yes. If you are converting your application to you can use the CVTMAKE sample program to convert your project files to work as internal makefiles.

CVTMAKE is ready to compile in the \MSVCNT\SAMPLES\CVTMAKE. See CVTMAKE.WRI in the same directory for more information.

How do I port 16-bit applications to 32 bits?

The Visual C++ for Windows NT documentation describes porting 16-bit applications.

- Migrating 16-bit MFC applications is described in “Microsoft Foundation Class Library Update” in *Class Library Reference*.
- For information on porting 16-bit Windows applications to Win32, see "Porting 16-bit code to 32-bit Windows" in *Programming Techniques*.
- Applications that use standard I/O calls such as **printf** and **scanf** can be built as Console applications under Windows NT. Function compatibility is described in *C Run-Time Library Reference*.

How do I build 32-bit applications from Visual C++ for Windows?

If you have both Visual C++ for Windows, Version 1.0 and Visual C++ for Windows NT installed on your machine, you can develop applications for Win32 and Win32s:

1. From MS-DOS, set up the Win32s subsystem by running the setup program in the \MSVC32S directory of the installation CD-ROM. You only need to do this once.
2. Start Windows 3.1 and run the Visual Workbench from Visual C++ for Windows.
3. From the Options menu, choose Directories. Update the paths to point to the 32-bit version of each.
4. From the Project menu, choose Open. Select the "Use as an external makefile" option and load your project makefile.
5. Build and execute the program as usual.

Note Visual C++ for Windows cannot debug 32-bit executable files. In addition, context-sensitive keyword help does not work

Does Visual C++ 32-Bit Edition let me write 16-bit applications for Windows 3.1?

No. Visual C++ 32-Bit Edition is designed to write 32-bit applications using the Win32 API. The product also includes Win32s which allows 32-bit applications to run under Windows 3.1.

Can I use the NT product to develop for Windows Version 3.1?

Yes, by using Win32s, you can create applications that run binary-compatibly on both Windows NT and Windows 3.1.

Can I use the NT product to target Windows Version 3.1 directly without going through Win32s *™*?

No. Win32s is required to target Windows 3.1 using Visual C++ 32-Bit Edition.

Can I use the NT product to target MS-DOS?

No. You can target MS-DOS using visual C++ Professional Edition.

Will you have a DOS application framework?

You can use non-user-interface features of Microsoft Foundation Class Library in DOS applications, and Microsoft Foundation Class Library compatible DOS application frameworks are available as additional tools, such as MEWEL, from third party vendors. Our current development work is focused on Windows.

Does Visual C++ 32-Bit Edition support the Pentium?

Yes. Visual C++ 32-Bit Edition includes specific new optimization and code generation technology for the Pentium and i486 processors.

Can I write POSIX or OS/2 applications using Visual C++ for Windows NT?

The current version of Visual C++ is targeted for the Win32 and Console subsystems. However, if you install the POSIX headers and libraries from the Microsoft Win32 Software Development Kit, you can use Visual C++ for Windows NT to write POSIX applications.

Does Visual C++ for Windows NT target the Silicon Graphics MIPS or Digital Equipment Alpha platforms?

No, only Intel-compatible microprocessors are supported with this version.

How do I build device drivers with Visual C++?

Although Visual C++ does not supply all of the libraries, headers, and tools needed to create device drivers for Windows NT, you can still use many Visual C++ components.

1. Update the command-line tools included with the DDK with the versions provided with Visual C++.

   Note Not all of the tools provided in the DDK (such as MASM) are supplied with Visual C++, so be careful when deleting files.
2. Make appropriate changes to the source files and makefile.
3. Use the BUILD or NMAKE utility to build the device driver.

What about the Macintosh tools I've heard about?

We are working on a product to allow cross platform development of applications for Windows and the Macintosh. No other details are available.

Integrated Development Environment

What's different in the 32-bit edition of Visual Workbench?

The Visual Workbench in Visual C++ for Windows NT has the following additions:

- 32-bit Project types
- Compiler and linker options for 32-bit build tools
- Integrated Source Profiler
- Find in files capability
- Multithread debugging capability
- Win32 Structured Exception Handling debugging
- Break command on the Debug menu
- Memory window in the debugger
- Generalized output window that accepts output from custom tools added to the Tools menu
- Stop command on the tools menu that will also stop build tools

There are also some option differences in the Editor, Colors, Tools, and Debug dialog boxes.

Does Visual C++, 32-Bit Edition come with an integrated Source Code Management System and is this system fully integrated in the development environment?

Shortly after Visual C++, 32-Bit Edition releases we will have a Source Code Management System, which can be integrated into the Visual Workbench using the Tools menu. The Source Control Management tools, however, will not be available as part of Visual C++, 32-Bit Edition but will be offered as a stand-alone product. You can also integrate other source control management systems with Visual C++, 32-Bit Edition by using the tools menu. The code editor automatically detects changes in file access attributes, making source control tasks easy (for instance, when a file changes from read only to read/write the code editor will automatically detect this change and then let you edit the file without having to close and then re-open the file).

Does the editor automatically insert matching brackets?

The editor supports jumping to matching brackets. It doesn't automatically insert matching brackets for you.

Does your editor support alternate keyboards?

No. We support alternate keyboard layouts for different countries, but no user customizable keyboard mappings.

Will Visual C++, 32-Bit Edition work for really large projects? I've got a lot of NMAKE files.

Visual C++, 32-Bit Edition works with existing PWB and NMAKE files as external projects. The internal project manager will also handle large projects. We've tested with projects like building Microsoft Excel and Word, and it functions correctly and efficiently.

Debugger

Does the debugger have a memory window?

Yes. This often requested feature has been integrated into the debugger.

Does the debugger support threads?

Yes. The debugger includes support for Windows NT threads. The threads dialog allows you to view the threads in your application, pause or resume them, get a stack trace, and more.

Does the debugger support Exceptions?

Yes. The debugger supports Windows NT exceptions. You can determine what the debugger should do for a given exception. Stop always, or stop only if unhandled.

Wizards

I have a large base of C code. Will the Wizards work for me too?

Both of the Wizards in Visual C++, 32-Bit Edition, AppWizard and ClassWizard, are designed expressly for the Microsoft Foundation Class Library (and C++) programmer. AppWizard is ideal for starting new projects that use Microsoft Foundation Class Library version 2.0 and C++. It's a great teaching tool as well—take a look at the skeleton application that it creates. If you are a Microsoft Foundation Class Library version 1.0 programmer, you can easily import your Microsoft Foundation Class Library version 1.0 code into ClassWizard by following the simple instructions outlined in the on-line technical note. We see the Wizards being used as people migrate their code to C++ or start new projects in C++. There are a few additions you need to make to your existing code to allow for mixed Microsoft Foundation Class Library/C++ and C code, and then new modules can be done in C++.

Why can't I change my mind with AppWizard and change from MDI to Non-MDI or add (Object Linking and Embedding) OLE support later?

AppWizard is a tool for creating an application quickly. To change, you should recreate a new application and move your implementations over. To switch an SDI app to MDI manually using the Visual WorkBench is relatively simple.

Why doesn't AppWizard have more options, such as letting me specify the base classes of my view, or OLE server?

AppWizard is designed to create a skeleton for a generic application with as few steps as necessary. Most other modifications can be done very simply with only a few changes using the Visual WorkBench editor. For example, removing the status bar requires deleting (or commenting out) only two lines of code. That's because an AppWizard application isn't a lot of code, but rather a good deal of bookkeeping information, such as declarations for derived classes. The power of AppWizard is in the application framework.

Why isn't ClassWizard a generic class editor? (It won't let you create/browse/edit arbitrary classes in a general way.)

We have an integrated class browser for this. ClassWizard is designed to manage classes that contain message maps, which are generally the visual classes in your application. The general rule is that ClassWizard can work with classes derived, directly or indirectly, from the Microsoft Foundation Class Library version 2.0 class CCmdTarget.

Why can't I override arbitrary virtual functions with ClassWizard?

ClassWizard maintains only functions that are mapped via message maps.

Why doesn't ClassWizard delete a function's definition when I delete the handler?

ClassWizard is designed to minimize the amount of modification it does to source files. Since ClassWizard works the way you do, allowing you to structure your code in an arbitrary manner, the cost of implementing this feature was too high and would potentially involve external database files as with code generators or CASE tools. We advise programmers to comment out the code, rather than delete it, because the code is often reused later. This practice is common on large scale applications.

Microsoft Foundation Class Library

What's the difference in terms of learning curve between Microsoft Foundation Class Library version 1.0 and version 2.0?

Microsoft Foundation Class Library version 2.0 is easier to use and to learn because of the higher level classes. For instance, in version 1.0 creating a toolbar involved calling the low level GDI calls to draw buttons, etc. In Microsoft Foundation Class Library version 2.0 all the details of a fully implemented toolbar are provided for you in a single class. So creating a toolbar is as simple as creating a new toolbar object. You no longer have to worry about the low level details.

What's different in the 32-bit edition of the Microsoft Foundation Class Library?

The Foundation Class Library shields the programmer from most of the changes brought about by moving to Win32. Only the message handlers **CWnd::OnCommand** and **CWnd::OnParentNotify** and the **CTime** class have changed. Also, there is no support for VBX controls, the Microsoft Windows for Pen Computing extensions, or AFXDLL (MFC200.DLL or MFC200D.DLL).

For more information, see “Microsoft Foundation Class Library Update” in *Class Library Reference.*

Does Microsoft Foundation Class Library handle message routing for you?

Microsoft Foundation Class Library handles message routing automatically using "message maps." Windows-oriented messages are automatically mapped to class member functions. Because messages are managed by Microsoft Foundation Class Library, the application framework can actually cache messages and make message handling faster than in C/SDK applications. Message maps eliminate the need to use the usual lengthy, error prone, CASE statements traditionally found in C Windows application code. In addition, ClassWizard lets you connect user interfaces messages to class member functions without having to write the necessary C++ code, making message handling fast, easy, and far less error prone.

Visual C++, 32-Bit Edition/Microsoft Foundation Class Library is creating a lot of code for me. Is it bullet-proof and bug free?

The reusable code that we provide in Visual C++, 32-Bit Edition is very solid. It's been thoroughly tested, and it's already being used in some professional and commercial applications—like App Studio—today.

How much of systems resources does Microsoft Foundation Class Library use?

The exact amount of memory, system resources, and hard disk space required by a Microsoft Foundation Class Library application varies depending on which features you use and other code you have written to extend the framework. However, creating smallest and fastest executables was a top priority in Microsoft Foundation Class Library. The Microsoft Foundation Class Library code is developed by a team of some of the most experienced Windows and C++ programmers at Microsoft. They focused on making the reusable code as tight as possible, giving developers the most highly optimized code. They also made sure that Microsoft Foundation Class Library uses as little memory and other system resources as possible. For instance, the toolbar is implemented with a single HWND (regardless of how many buttons there are in the toolbar) to minimize use of user heap.

How many classes are there in Microsoft Foundation Class Library version 2.0?

Microsoft Foundation Class Library version 2.0 offers approximately 13 new, high-level classes. In total, it includes over 100 classes.

Why does the CEditView class support only text files smaller than 64K and a single font?

CEditView is a high-level encapsulation of the Windows edit control. We are working with the Microsoft Windows team to provide a richer edit control in a future version of Windows that will support a larger feature set. CEditView's purpose is to serve as a simple text editor and not as a word processor.

Can Microsoft Foundation Class Library version 2.0 be compiled with other C++ compilers?

Yes. Microsoft Foundation Class Library version 2.0 was designed and implemented using techniques that will expedite porting to other compilers. As with Microsoft Foundation Class Library version 1.0, the burden of porting Microsoft Foundation Class Library version 2.0 to a compiler falls to the vendor since there are too many compiler vendors for Microsoft to support. We will work with any compiler vendor that wishes to participate in the Microsoft Foundation Class Library licensing program in order to compile Microsoft Foundation Class Library with its compiler. Microsoft Foundation Class Library version 2.0 requires the addition of three vendor-specific files: a version header file, a version source file, and a vendor-specific project file. For example, the version header file would contain any #define or #undef lines to change symbolic constants or compiler directives; the version source file would contain any non-implemented runtime functionality; and the project file would compile the library. The Borland versions of these files should be less than 100 lines total.

Why doesn't Microsoft Foundation Class Library version 2.0 support OLE version 2.0? Isn't OLE version 1.0 an old standard?

OLE 2.0 is only available in beta test at this time, and Microsoft Foundation Class Library version 2.0 is a shipping product. As OLE 2.0 matures, Microsoft Foundation Class Library version 2.0 support will be made available.

The standard OLE support built into NT is OLE 1, so we support OLE 1! OLE 2 support using MFC was demonstrated at the recent OLE 2 Developers Conference. Microsoft is still working on developing MFC OLE 2 support and will make this support available when it is generally supported in the operating system.

Why doesn't the Microsoft Foundation Class Library version 2.0 support Windows NT specific features?

The 32-bit version of MFC 2.0 is focused at providing a class library solution for people migrating 16-bit Windows 3.x applications to NT and/or Win32S or for people writing or maintaining applications that target more than one platform all using the same source code. Therefore, the current version does not rely on any features unique to Windows NT.

Does the Microsoft Foundation Class Library version 2.0 support multi-threading?

MFC supports a limited form of multi-threaded applications. The main user interface code written in MFC must reside in the single main thread. Additional helper threads can be written to offload work from the main UI thread. These helpers threads can make use of a very limited subset of the MFC API. Documentation provided with MFC includes a more in-depth discussion of this support.

Once you rely on threads your program will only be able to run on one of the three platforms that MFC supports (Win16 and Win32S do not support threads). Therefore it is recommended that you should try to abstract out those helper threads so that your program can run on these systems if appropriate.

How do I move existing 16-bit MFC 2 applications to Windows NT?

The easiest way to do this is to create a new project file, add the source files to it and recompile with the Visual C++ 32-bit edition compiler/IDE. You will, of course, have to fix any non-portable application code that you may have written.

#########