Module 1: Microsoft Windows, Visual C++ and Microsoft Foundation Class (MFC)

Program examples compiled using Visual C++ 6.0 (MFC 6.0) compiler on Windows XP Pro machine with Service Pack 2. Topics and sub topics for this Tutorial are listed below:

Introduction

After you have completed the C++ and object oriented and Standard Template Library (STL) tutorials, hopefully you got the fundamental ideas how the classes were constructed and used and how the objects instantiated, how to organize multiple files in declaration, implementation, main program parts etc. You also were introduced the principles of the object encapsulation, inheritance and polymorphism. In inheritance you have learnt that child class can inherit the properties of the parent or base classes. You also have learnt how we send messages to get tasks done instead of the traditional manner using function call. The following Table lists important terms that we have used in the C++ Tutorial.

Term	Description
class	Is a group of data and methods (functions). A class is very much like a structure type as used in ANSI-C, it is just a type used to create a variable which can be manipulated through method in a program.
object	Is an instance of a class, which is similar to an analogous of a variable, defined as an instance of a type. An object is what you actually use in a program since it contains

Table 1

And the following figure shows a very simple class hierarchy used in Tenouk's C++ Tutorial.

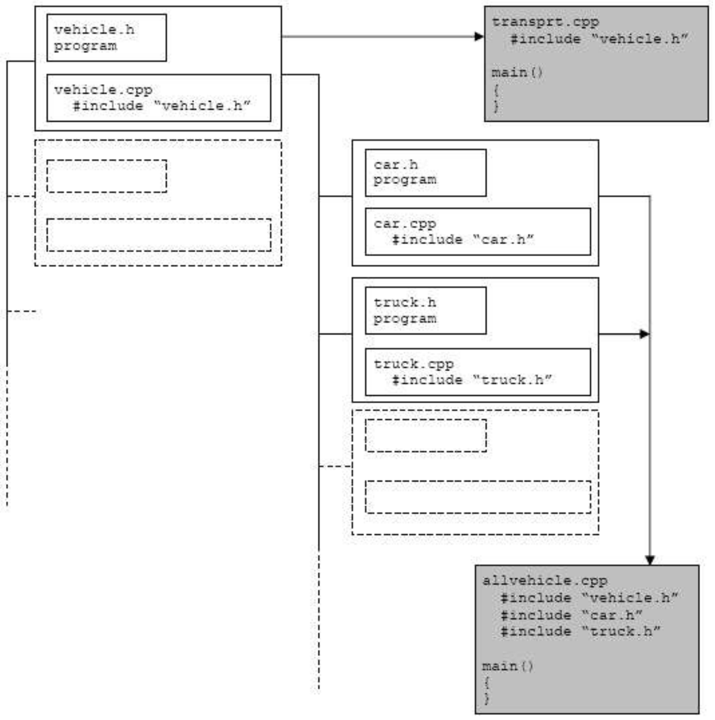

vehicle.h program
vehicle.cpp #include "vehicle.h"
transprt.cpp #include "vehicle.h" main() { }
car.h program
car.cpp #include "car.h"
truck.h program
truck.cpp #include "truck.h"
allvehicle.cpp #include "vehicle.h" #include "car.h" #include "truck.h" main() { }

Figure 1: A very simple C++ Class hierarchy.

Though the previous Tutorials just provide the basics of the Object Oriented, but it is a very useful to grab the fundamentals in order to understand the more complex solutions such as Microsoft Foundation Class (MFC) or other object oriented programming such as Java and C#. Our next task is to explore the implementation specific of the classes in MFC. MFC contains a bunch of classes that ready to be used mainly for Windows graphic user interface programming. MFC is designed to be a great class library for creating graphically rich, sophisticated Windows applications. Before we go any further we will get some idea about the compiler, Visual C++ 6.0. Be prepared, you will find many object oriented terms, principles and many more program files in MFC programming.

The Visual C++ Components

You still can develop C-language Windows programs using only the Win32 API using Visual C++ as used in many C programming in Tenouk Tutorial. You can use many Visual C++ tools, including the resource editors, to make low-level Win32 programming easier. Visual C++ also includes the ActiveX Template Library (ATL), which you can use to develop ActiveX controls for the Internet. ATL programming is neither Win32 C-language programming nor MFC programming. In this Tutorial we will concentrate the C++ and MFC programming using Visual C++ 6.0. Use of the MFC library programming interface doesn't cut you off from the Win32 functions. In fact, you'll almost always need some direct Win32 calls in your MFC library programs. A quick run-through of the Visual C++ components will help you get your bearings before you zero in on the application framework. Figure 2 shows an overview of the Visual C++ application build process.

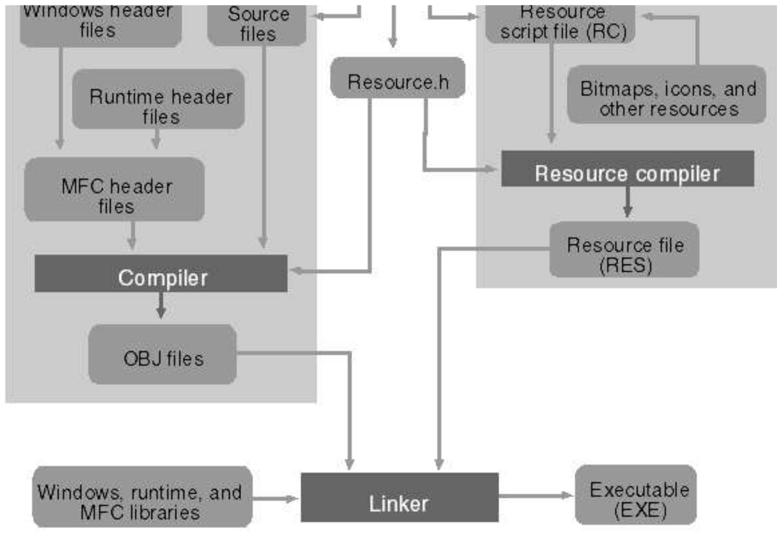
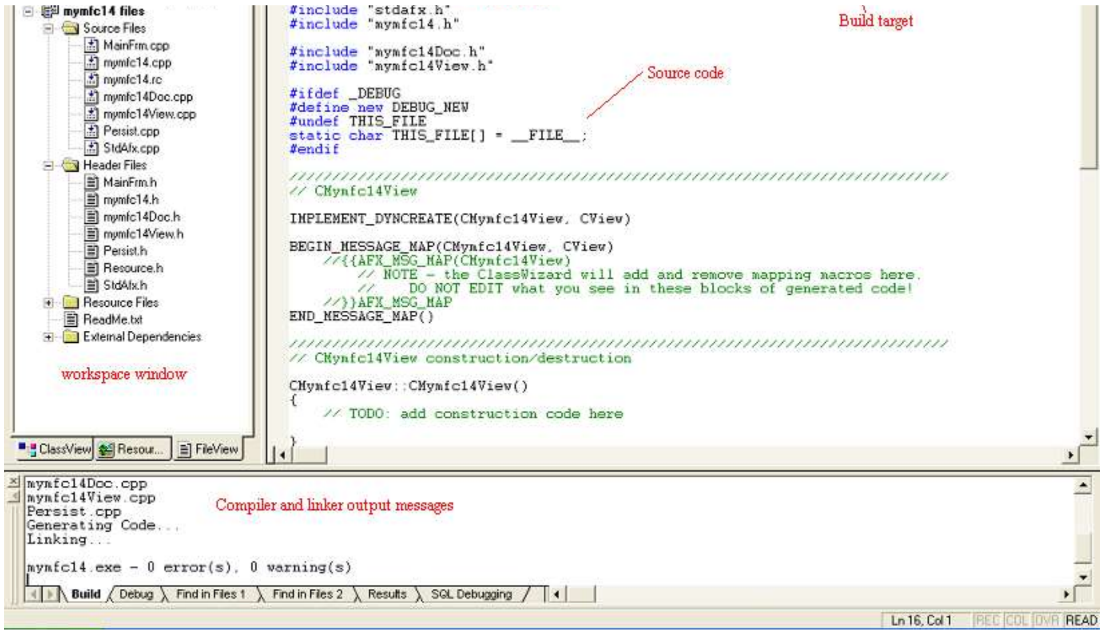


Figure 2: The Visual C++ application build process.

Microsoft Visual C++ 6.0 and the Build Process

Visual Studio 6.0 is a suite of developer tools that includes Visual C++ 6.0. The Visual C++ IDE is shared by several tools including Microsoft Visual J++. The IDE has come a long way from the original **Visual Workbench**, which was based on **QuickC** for Windows. Docking windows, configurable toolbars, plus a customizable editor that runs macros, are now part of Visual Studio. The online help system (now integrated with the MSDN Library viewer) works like a Web browser. Figure 3 shows Visual C++ 6.0 in action.



```
#include "stdafx.h"
#include "mymfc14.h"

#include "mymfc14Doc.h"
#include "mymfc14View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMymfc14View

IMPLEMENT_DYNCREATE(CMymfc14View, CView)

BEGIN_MESSAGE_MAP(CMymfc14View, CView)
    //{{AFX_MSG_MAP(CMymfc14View)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMymfc14View construction/destruction

CMymfc14View::CMymfc14View()
{
    // TODO: add construction code here

}
```

```
mymfc14Doc.cpp
mymfc14View.cpp
Persist.cpp
Generating Code...
Linking...

mymfc14.exe - 0 error(s), 0 warning(s)
```

Figure 3: Visual C++ 6.0 windows with main components displayed.

If you've used earlier versions of Visual C++ or another vendor's IDE, you already understand how Visual C++ 6.0 operates. But if you're new to IDEs, you'll need to know what a project is. A project is a collection of interrelated source files that are compiled and linked to make up an executable Windows-based program or a DLL. Source files for each project are generally stored in a separate subdirectory. A project depends on many files outside the project subdirectory too, such as include files and library files. Experienced programmers are familiar with makefiles. A makefile stores compiler and linker options and expresses all the interrelationships among source files. A source code file needs specific include files, an executable file requires certain object modules and libraries, and so forth. A make program reads the makefile and then invokes the compiler, assembler, resource compiler, and linker to produce the final output, which is generally an executable file. The make program uses built-in inference rules that tell it, for example, to invoke the compiler to generate an OBJ file from a specified CPP file.

In a Visual C++ 6.0 project, there is **no makefile** (with an **MAK** extension) unless you tell the system to export one. A text-format **project file** (with a **DSP extension**) serves the same purpose. A separate text-format **workspace file** (with a **DSW extension**) has an **entry** for each project in the workspace. It's possible to have multiple projects in a workspace, but all the examples in this Tutorial have just one project per workspace. To work on an existing project, you tell Visual C++ to open the DSW file and then you can edit and build the project. The project directory also can be copied to other directory or system and can be edited, built and run as usual.

The Resource Editors: Workspace ResourceView

When you click on the **ResourceView** tab in the Visual C++ Workspace window as shown below, you can select a resource for editing by double clicking the resource such as the About dialog box shown below.

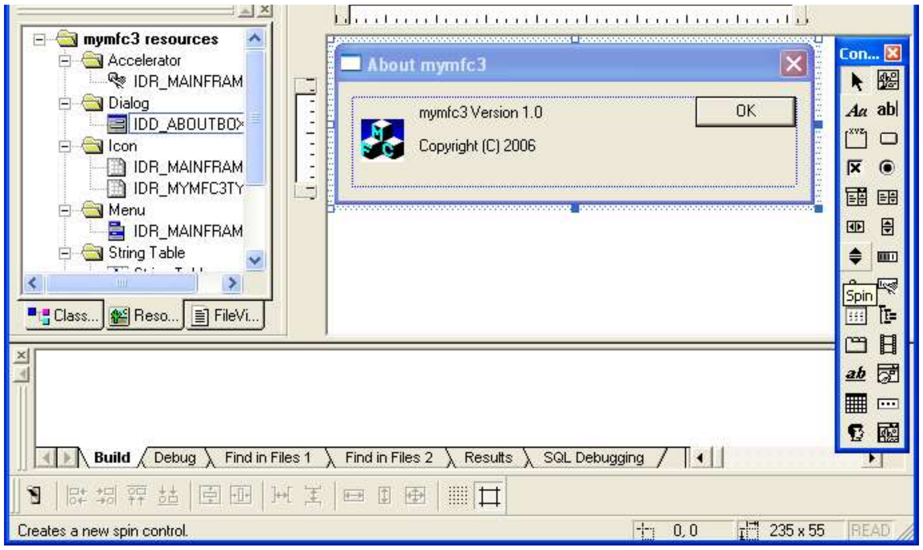


Figure 4: Visual C++ ResourceView.

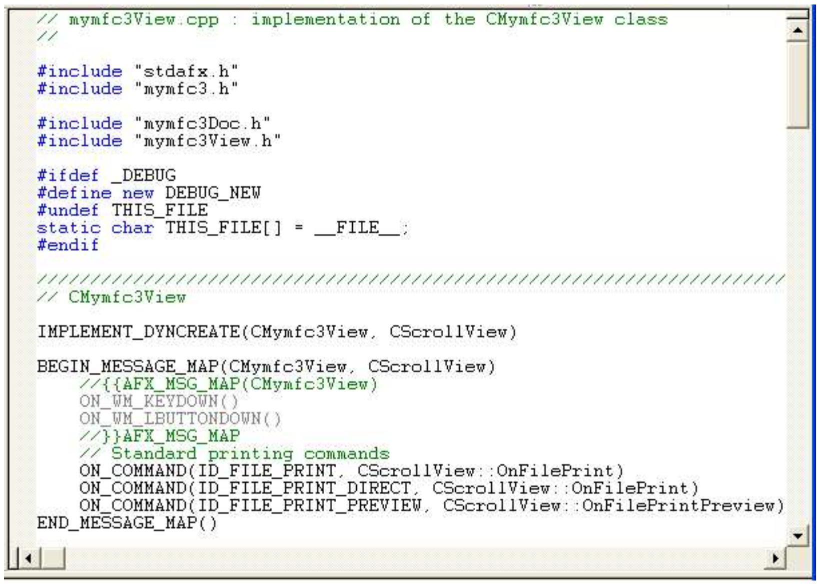
The main window hosts a resource editor appropriate for the resource type. The window can also host a WYSIWYG editor for menus and a powerful graphical editor for dialog boxes, and it includes tools for editing icons, bitmaps, and strings. The dialog editor allows you to insert ActiveX controls in addition to standard Windows controls and the new Windows common controls which have been further extended in Visual C++ 6.0. Each project usually has one text-format resource script (RC) file that describes the project's menu, dialog, string, and accelerator resources. The RC file also has #include statements to bring in resources from other subdirectories. These resources include project-specific items, such as bitmap (BMP) and icon (ICO) files, and resources common to all Visual C++ programs, such as error message strings. The resource editors can also process EXE and DLL files, so you can use the clipboard to "steal" resources, such as bitmaps and icons, from other Windows applications.

The C/C++ Compiler

The Visual C++ compiler can process both C source code and C++ source code. It determines the language by looking at the source code's filename extension. A C extension indicates C source code, and CPP or CXX indicates C++ source code. The compiler is compliant with all ANSI standards, including the latest recommendations of a working group on C++ libraries, and has additional Microsoft extensions. Templates, exceptions, and runtime type identification (RTTI) are fully supported in Visual C++ version 6.0. The C++ Standard Template Library (STL) is also included, although it is not integrated into the MFC library.

The Source Code Editor

Visual C++ 6.0 includes a sophisticated source code editor that supports many features such as dynamic syntax coloring, auto-tabbing, keyboard bindings for a variety of popular editors (such as VI and EMACS), and pretty printing.



```
// mymfc3View.cpp : implementation of the CMymfc3View class
//

#include "stdafx.h"
#include "mymfc3.h"

#include "mymfc3Doc.h"
#include "mymfc3View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMymfc3View

IMPLEMENT_DYNCREATE(CMymfc3View, CScrollView)

BEGIN_MESSAGE_MAP(CMymfc3View, CScrollView)
    //{{AFX_MSG_MAP(CMymfc3View)
    ON_WM_KEYDOWN()
    ON_WM_LBUTTONDOWN()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

Figure 5: Visual VC++ source code editor.

The Visual C++ resource compiler reads an ASCII resource script (RC) file from the resource editors and writes a binary RES file for the linker.

The Linker

The linker reads the OBJ and RES files produced by the C/C++ compiler and the resource compiler, and it accesses LIB files for MFC code, runtime library code, and Windows code. It then writes the project's EXE file. An incremental link option minimizes the execution time when only minor changes have been made to the source files. The MFC header files contain #pragma statements (special compiler directives) that specify the required library files, so you don't have to tell the linker explicitly which libraries to read.

The Debugger

For the serious and real application developers, many times they will encounter problems in their codes. Unfortunately those problems, normally called bugs, could not be found through the code inspections or reviews. They need a debugger to debug their programs to find those bugs. The Visual C++ debugger has been steadily improving, though it doesn't actually fix the bugs yet. The debugger works closely with Visual C++ to ensure that breakpoints are saved on disk. Toolbar buttons insert and remove breakpoints and control single-step execution. Figure 9 illustrates the Visual C++ debugger in action. Note that the Variables and Watch windows can expand an object pointer to show all data members of the derived class and base classes. If you position the cursor on a simple variable, the debugger shows you its value in a little window. To debug a program, you must build the program with the compiler and linker options set to generate debugging information. The debug session can be invoked through the **Build** menu and **Start Debug** sub menu shown below.

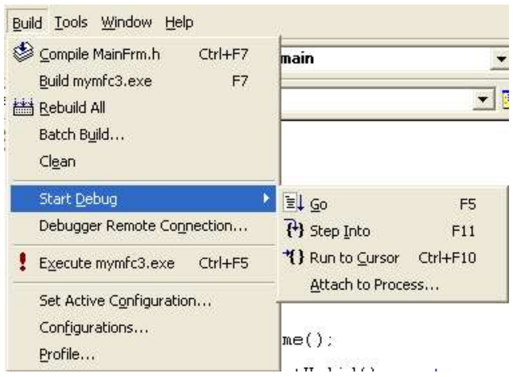
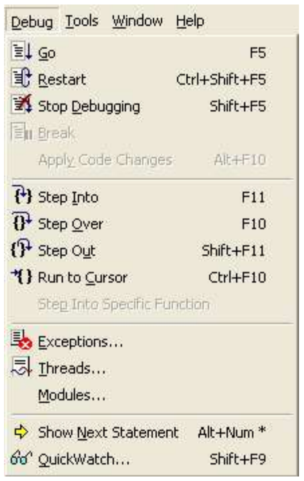


Figure 6: Visual C++ Debugging menu.

After the debug session been invoked, a new dynamic **Debu**g menu will be displayed as shown below.



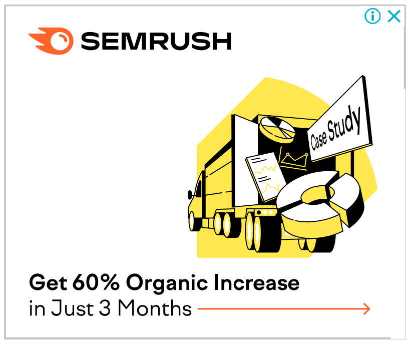
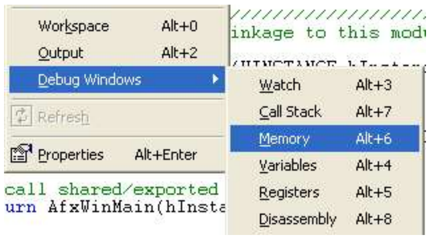


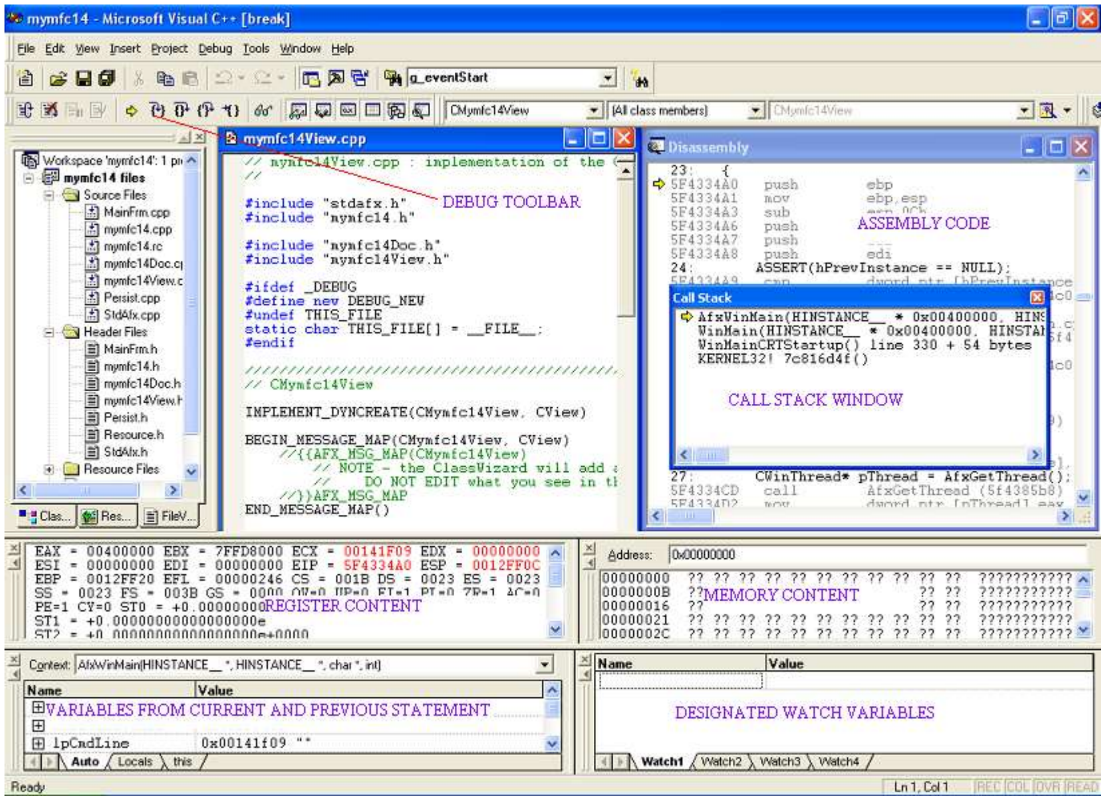
Figure 7: Visual C++ debug options.

Then you can view much more information through the **View** menu and **Debug Windows** sub menu.



```
///////////////////
inkage to this modu
call shared/exported
urn AfxWinMain(hInsta
```

Figure 8: Visual C++ Debug window options.



```
// mymfc14View.cpp : implementation of the
//

#include "stdafx.h"
#include "mymfc14.h"

#include "mymfc14Doc.h"
#include "mymfc14View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////
// CMymfc14View

IMPLEMENT_DYNCREATE(CMymfc14View, CView)

BEGIN_MESSAGE_MAP(CMymfc14View, CView)
    //{{AFX_MSG_MAP(CMymfc14View)
        // NOTE - the ClassWizard will add a
        //    DO NOT EDIT what you see in th
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
23:   {
5F4334A0   push       ebp
5F4334A1   mov        ebp,esp
5F4334A3   sub
5F4334A6   push
5F4334A7   push
5F4334A8   push       edi
24:          ASSERT(hPrevInstance == NULL);
27:          CWinThread* pThread = AfxGetThread();
5F4334CD   call       AfxGetThread (5f4385b8)
```

```
AfxWinMain(HINSTANCE__ * 0x00400000, HINS
WinMain(HINSTANCE__ * 0x00400000, HINSTAN
WinMainCRTStartup() line 330 + 54 bytes
KERNEL32! 7c816d4f()
```

```
EAX = 00400000 EBX = 7FFD8000 ECX = 00141F09 EDX = 00000000
ESI = 00000000 EDI = 00000000 EIP = 5F4334A0 ESP = 0012FF0C
EBP = 0012FF20 EFL = 00000246 CS = 001B DS = 0023 ES = 0023
SS = 0023 FS = 003B GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=1 AC=0
PE=1 CY=0 ST0 = +0.00000000
ST1 = +0.0000000000000000e
```

```
Address: 0x00000000
00000000  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ???????????
0000000B  ??                          ?? ??  ???????????
00000016  ??                          ?? ??  ???????????
00000021  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ???????????
0000002C  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ???????????
```

Context: AfxWinMain(HINSTANCE__ *, HINSTANCE__ *, char *, int)

Name	Value
lpCmdLine	0x00141f09 ""

Figure 9: The Visual C++ debugger window.

Visual C++ 6.0 adds a new twist to debugging with the **Edit And Continue** feature. Edit And Continue lets you debug an application, change the application, and then continue debugging with the new code. This feature dramatically reduces the amount of time you spend debugging because you no longer have to manually leave the debugger, recompile, and then debug again. To use this feature, simply edit any code while in the debugger and then hit the continue button. Visual C++ 6.0 automatically compiles the changes and restarts the debugger for you.

AppWizard

AppWizard is a code generator that creates a working skeleton of a Windows application with features, class names, and source code filenames that you specify through dialog boxes.

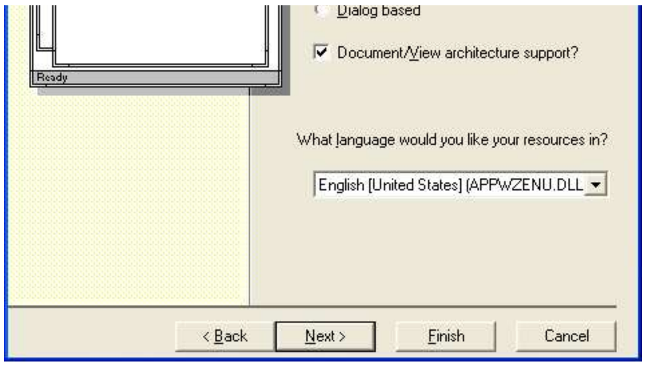

Ready
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

Figure 10: Visual C++ AppWizard step 1 of 6.

You'll use AppWizard extensively as you work through the examples in this Tutorial. AppWizard code is minimalist code; the functionality is inside the application framework base classes. AppWizard gets you started quickly with a new application. Advanced developers can build custom AppWizards. Microsoft Corporation has exposed its macro-based system for generating projects. If you discover that your team needs to develop multiple projects with a telecommunications interface, you can build a special wizard that automates the process.

ClassWizard

ClassWizard is a program (implemented as a DLL) that's accessible from Visual C++'s View menu. ClassWizard takes the drudgery out of maintaining Visual C++ class code. It can be access through the **View** menu and **ClassWizard…** sub menu.

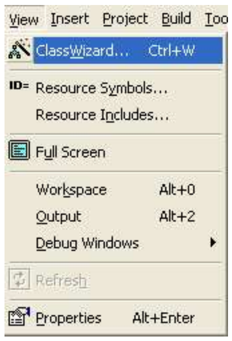


Figure 11: Invoking the ClassWizard through the **View** menu.

The following is the MFC ClassWizard dialog box.

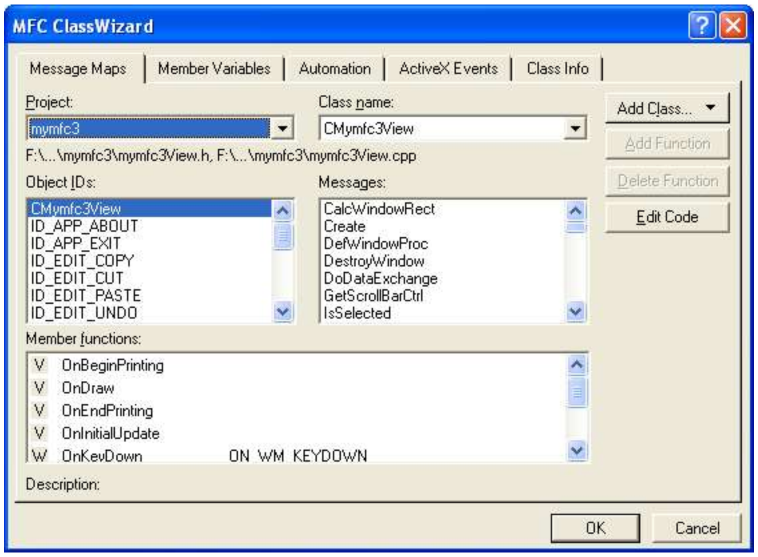
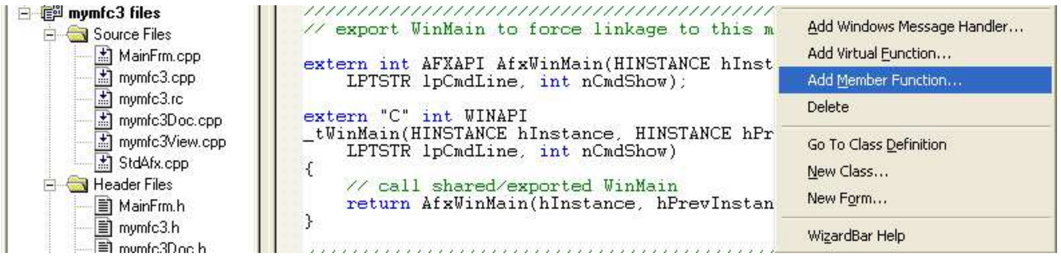


Figure 12: ClassWizard dialog.

When you need a new class, a new virtual function, or a new message-handler function (Message maps), ClassWizard writes the prototypes, the function bodies, and (if necessary) the code to link the Windows message to the function. ClassWizard can update class code that you write, so



```
/////////////////////////////////////////////////
// export WinMain to force linkage to this m

extern int AFXAPI AfxWinMain(HINSTANCE hInst
    LPTSTR lpCmdLine, int nCmdShow);

extern "C" int WINAPI
_tWinMain(HINSTANCE hInstance, HINSTANCE hPr
    LPTSTR lpCmdLine, int nCmdShow)
{
    // call shared/exported WinMain
    return AfxWinMain(hInstance, hPrevInstan
}
```

Figure 13: Visual C++'s WizardBar toolbar.

The Source Browser

If you write an application from scratch, you probably have a good mental picture of your source code files, classes, and member functions. If you take over someone else's application, you'll need some assistance. The Visual C++ Source Browser (the browser, for short) lets you examine (and edit) an application from the class or function viewpoint instead of from the file viewpoint.

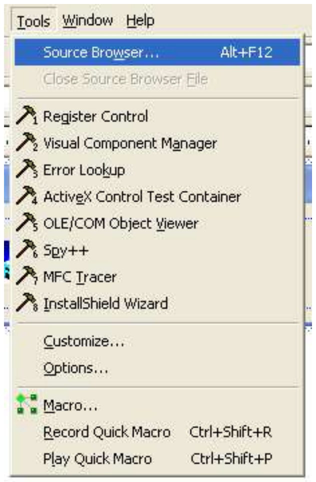


Figure 14: Visual C++ Source Browser.

It's a little like the "inspector" tools available with object-oriented libraries such as Smalltalk. The browser has the following viewing modes:

- **Definitions and References**: You select any function, variable, type, macro, or class and then see where it's defined and used in your project.
- **Call Graph/Callers Graph**: For a selected function, you'll see a graphical representation of the functions it calls or the functions that call it.
- **Derived Classes and Members/Base Classes and Members**: These are graphical class hierarchy diagrams. For a selected class, you see the derived classes or the base classes plus members. You can control the hierarchy expansion with the mouse.
- **File Outline**: For a selected file, the classes, functions, and data members appear together with the places in which they're defined and used in your project.

If you rearrange the lines in any source code file, Visual C++ regenerates the browser database when you rebuild the project. This increases the build time. In addition to the browser, Visual C++ has a **ClassView** option (shown below) that does not depend on the browser database. You get

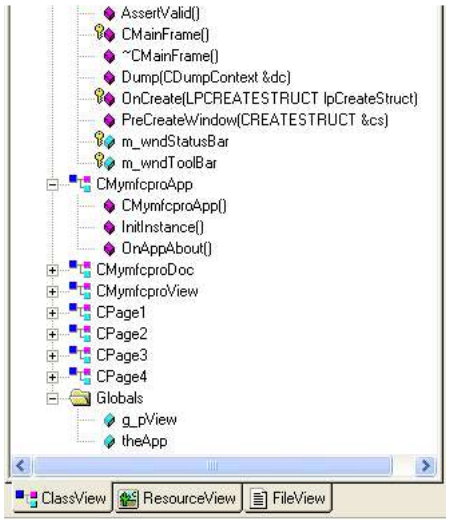


Figure 15: Visual C++ ClassView.

Windows Diagnostic Tools

Visual C++ 6.0 contains a number of useful diagnostic tools. For example, SPY++ gives you a tree view of your system's processes, threads, and windows. It also lets you view messages and examine the windows of running applications.

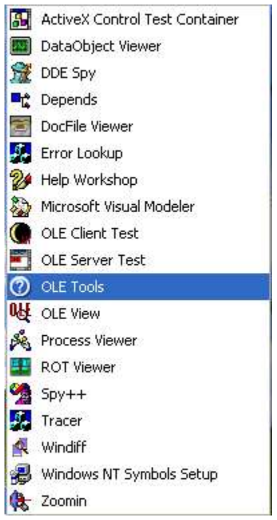
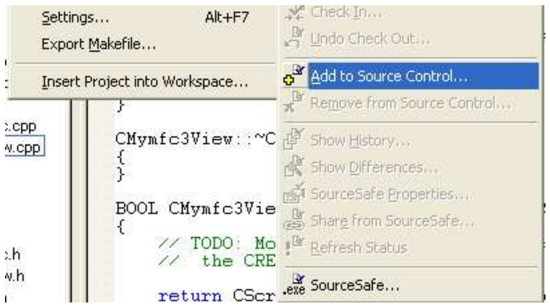


Figure 16: Visual C++ diagnostic tools.

You'll find **PVIEW** (PVIEW95 for Windows 95) useful for killing errant processes that aren't visible from the Windows 95 task list. The Windows NT Task Manager, which you can run by right-clicking the toolbar, is an alternative to PVIEW. Visual C++ also includes a whole suite of ActiveX utilities, an ActiveX control test program (now with full source code in Visual C++ 6.0), the help workshop (with compiler), a library manager, binary file viewers and editors, a source code profiler, and other utilities.

Source Code Control

During development of Visual C++ 5.0, Microsoft bought the rights to an established source code control product named **SourceSafe**.



```
}
CMymfc3View::~C
{
}

BOOL CMymfc3Vie
{
    // TODO: Mo
    //   the CRE

    return CScr
```

Figure 17: Visual C++ Source Code control.

This product has since been included in the Enterprise Edition of Visual C++ and Visual Studio Enterprise, and it is integrated into Visual C++ so that you can coordinate large software projects. The master copy of the project's source code is stored in a central place on the network, and programmers can check out modules for updates. These checked-out modules are usually stored on the programmer's local hard disk. After a programmer checks in modified files, other team members can synchronize their local hard disk copies to the master copy. Other source code control systems can also be integrated into Visual C++.

Continue on next module, part 2...

Further reading and digging:

1. MSDN MFC 7.0 class library online documentation.
2. MSDN MFC 9.0 class library online documentation - latest version.
3. Porting & Migrating your older programs.
4. MSDN Library
5. DCOM at MSDN.
6. COM+ at MSDN.
7. COM at MSDN.
8. Windows data type.
9. Win32 programming Tutorial.
10. The best of C/C++, MFC, Windows and other related books.
11. Unicode and Multi-byte character set: Story and program examples.

| Tenouk C & C++ | MFC Home | MFC Intro 2 | Download | Site Index |

[illegible]