MAM College of Engineering and Technology, Trichy-621105

Department Of Information Technology

CS1308 VISUAL PROGRAMMING LAB MANUAL

INDEX

CS1308 – VISUAL PROGRAMMING LABORATORY

LTP

0 0 3

LIST OF EXPERIMENTS:

WINDOWS SDK / VISUAL C++

- Writing code for keyboard and mouse events
- Dialog Based applications.
- Creating MDI applications

VISUAL C++

- Threads
- Document view Architecture, Serialization.
- Dynamic controls
- Menu, Accelerator, Tool tip, Tool bar.
- Creating DLLs and using them.
- Data access through ODBC.
- Creating ActiveX control and using it.

HARDWARE REQUIRED

- Pentium IV/III Processor
- HDD 40GB
- RAM 128 MB or Above

SOFTWARE REQUIRED

- Window 98/2000/XP
- Visual Studio 6.0
- MS-Access

RATIONALE BEHIND VP LAB

Visual C++ comes within Microsoft Visual Studio 6.0. Visual Studio 6.0 also contains Visual Basic, Visual C#, and Visual J#. Using Visual Studio 6.0, you can mix and match languages within one "solution". We will, however, focus on developing C++ code throughout these labs.

Visual Studio 6.0 is a package that contains all the libraries, examples, and documentation needed to create applications for Windows. Instead of talking about programs we talk about projects and work space.

Microsoft Visual C++ provides a powerful and flexible development environment for creating Microsoft Windows-based and Microsoft .NET-based applications. It can be used as an integrated development system, or as a set of individual tools. Visual C++ is comprised of these components:

Visual C++ makes use of this class library and facilitates the programmer by making available a good number of programming tools like menu editor and dialog editor for designing menus and dialog boxes respectively. It takes us away from the conventional method of starting from the scratch and coding similar programs & objects for normally every application. Visual C++ also provides the facility of various wizards that lead us through a step by step process of building the skeleton of application. It also has a set of integrated debugging tools. All these features make Visual C++ a complete programming environment specifically suited for system level applications. Once you start appreciating the great power of Visual C++, you can design your own editors, scribble applications or may be a whole GUI environment in a very short time.

Some of the features of Visual C++ are:

- Code reusability

- Integrated development environment
- Application wizards for MFC applications, DLLs, ActiveX controls, ATL projects, ATL COM Objects and ISAPI extensions
- Components and Controls Gallery to store and access reusable controls and components
- Portability across platforms

In addition to conventional graphical user-interface applications, Visual C++ enables developers to build Web applications, smart-client Windows-based applications, and solutions for thin-client and smart-client mobile devices. C++ is the world's most popular systems-level language, and Visual C++ gives developers a world-class tool with which to build soft

Visual C++ 6.0

Visual C++ 6.0 in particular provides lot of new features, some of which are mentioned below.

- Visual C++ 6.0 provides a whole new set of wizards to build the skeleton applications.
- It provides easier application coding and building and a greater support for ActiveX1 and Internet Technologies.

Visual C++ 6.0 components:

- **VC++ Developer Studio** - An integrated application, which provides a set of programming- Tools.
- **VC++ Runtime Libraries** - These are the libraries that provide standard functions like *strlen and strcpy* etc. that can be called from a standard C or C++ functions.
- **MFC and Template Libraries** - The extensive C++ class library especially designed for creating GUI programs.
- **VC++ Build Tools** - It comprises of C/C++ Compiler, the Linker, resource compiler especially designed for compiling the resources and some other tools required for generating a 32-bit Windows programs.
- **ActiveX**

- **Data Access Components** - This includes Database drivers, controls and other tools required by VC++ to interact with Databases.
- **Enterprise Tools** - These are the advanced level Tools like Application Performance explorer or Visual Studio Analyzer.2
- **Graphics** - Consist of Bitmaps, Metafiles, Cursors and icons available for inclusion in the application

Objectives:

- After completing this lab, students will upgrade their knowledge in the field of VC++.
- Students will also understand the concepts of Visual C++ programming
- Getting more knowledge about windows programming.
- It clears the basics concepts of Object Oriented Programming (C++).
- Students will understand and deal with editors, tools, class libraries Debugging techniques and more.

PROGRAM 1

OBJECT OF THE PROGRAM:* Study Window's API and Their *Relationship with MFC classes

API is an acronym for Application Programming Interface. It is simply a set of functions that are part of Windows OS. Programs can be created by calling the functions present in the API. The Programmer doesn't have to bother about the internal working of the functions. By just knowing the function prototype and return value he can invoke the API Functions.

A good understanding of Windows API would help you to become a good Windows programmer. Windows itself uses the API to perform its amazing GUI magic. The Windows APIs are of two basic varieties:

- API for 16-bit Windows(Win16 API)
- API for 32-bit Windows(Win32 API)

Each of these has sub-APIs within it. If you are to program Windows 3.1 then you have to use Win16 API, whereas for programming Windows 95 and Windows NT you have to use Win32 API.

Win16 is a 16-bit API that was created for 16-bit processors, and relies on 16-bit values. Win32 is a 32-bit API created generation of 32-bit CPUs and it relies on 32-bit values.

Win16 API	Win32 API	Description
USER.EXE	USER32.DLL	The USER components is responsible for window management, including messages, menus, cursors, communications, timer etc.

GDI.EXE	GDI32.DLL	The GDI management is the Graphics Device Interface; it takes care of the user interface And graphics drawing, including Windows metafiles, bitmaps, device contexts, and fonts.
KRNL386.EXE	KERNEL32.DLL	The KERNEL component handles the low level functions of memory, task, and resource Management that are the heart of Windows.

The Win16 and Win32 APIs ere similar in most respects, but the Win16 API can be considered as a subset of Win32 API.Win32 API contains almost everything that the Win16 API has, and much more.

At its core each relies on three main components to provide most of the functionality of Windows. These core components along with their purpose are shown in the table given above. Although the Win16 versions of these components have .EXE extensions, they are actually all DLLs and cannot execute on their own.

The Win32 API has many advantages, some obvious and others that are not so obvious. The following lists major advantages that applications have when developed with the Win32 API and a 32-bit compiler (such as Microsoft's Visual C++):

• **True multithreaded applications**.

Win32 applications support true preemptive multitasking when running on Windows 95 and Windows NT.

• **32-bit linear memory.**

Applications no longer have limits imposed by segmented memory. All memory pointers are based on the applications virtual address and are represented as a 32-bit integer.

• **No memory model.**

The memory models (small, medium, large, etc.) have no meaning in the 32-bit environment. This means there is no need for near and far pointers, as all pointers can be thought of as far.

• **Faster .**

A well-designed Win32 application is generally faster. Win32 applications execute more efficiently than 16-bit applications.

• **Common API for all platforms**.

The Win32 API is supported on Windows 95, Windows NT on all supported hardware, Windows CE, and the Apple Macintosh.

MFC (Microsoft Foundation's Class)

The C++ class library that Microsoft provides with its C++ compiler is to assist programmers in creating Windows-based applications. MFC hides the fundamental Windows API in class hierarchies, so that, programmers can write Windows-based applications without needing to know the details of the native Windows API.

The MFC classes are coded in C++. It provides the necessary code for managing windows, menus and dialog-boxes. It also helps in carrying out basic tasks like performing basic input-output, storing the collection of data objects etc. It provides the basic framework for the application on which the programmer can build the rest of the customized code.

MFC Library is a collection of classes in hierarchical form, where classes are derived from some base class. For most of the classes provided by MFC, base class is CObject from which most of the classes are inherited. The rest few classes are independent classes. The classes in MFC Library can be categorized as:

- Root Class: CObject
- MFC Application Architecture Classes
- Window, Dialog, and Control Classes

- Drawing and Printing Classes
- Simple Data Type Classes
- Array, List, and Map Classes
- File and Database Classes
- Internet and Networking Classes
- OLE Classes
- Debugging and Exception Classes

In addition to these classes, the Microsoft Foundation Class Library contains a number of global functions, global variables, and macros. The Microsoft Foundation Class Library (MFC) supplies full source code. The full source code is supplied in the form of Header files (.H) that are in the MFC\Include directory and implementation files (.Cpp) that are in the MFC\Src directory

MFC Library was especially designed to provide an object oriented interface to Windows, simultaneously providing the compatibility with the windows programs written in C language. The compatibility with C language was required as the Windows was developed long before MFC came into existence and hence all the applications for Windows were initially written in C, a large number of which is still in use.

MFC Library provides the following features:

- Significant reduction in the effort to write an application for Windows.
- Execution speed comparable to that of the C-language API.
- Minimum code size overhead.
- Ability to call any Windows C function directly.
- Easier conversion of existing C applications to C++.
- Ability to leverage from the existing base of C-language Windows programming experience.
- Easier use of the Windows API with C++ than with C.
- Easier-to-use yet powerful abstractions of complicated features such as ActiveX, database support, printing, toolbars, and status bars.

- True Windows API for C++ that effectively uses C++ language features.

Advantages of MFC

Microsoft Foundation Class Library greatly reduces the development time, makes the code more portable, provides support without reducing the programming freedom and flexibility and provides easy access to user interface elements and technologies like ActiveX and Internet programming which are otherwise very hard to program. By using MFC, developers can add many capabilities to their applications in an easy, object-oriented manner. MFC simplifies database programming by providing Data Access Objects (DAO) and Open Database Connectivity (ODBC) and network programming through Windows Sockets.
MFC offers many advantages like:

- Application Framework
- Largest base of reusable C++ source code
- Integration with Visual C++
- Flexible and fast database access
- Support for Internet and ActiveX technology
- Support for messaging API
- Support for multithreading

MFC is not only library of classes. MFC is also an application framework.MFC helps to define the structure of an application and handles many routine chores on the application's behalf.

Starting with CWinApp, the class that represents that represents the application itself, MFC encapsulates virtually every aspect of a program's operation. The framework supplies the WinMain() function, and WinMain() in turn calls the application object's member functions to make the program go. One of the CWinApp member functions called by WinMain()- Run() - encapsulates the message loop that literally runs the program.

PROGRAM 2

OBJECT OF THE PROGRAM: Study essential classes in Document View Architecture and Their Relationship with each other

Parts of Application

The application source code generated for you by AppWizard actually consists of 4 major classes. All these classes can be seen by expanding the class listing in the Class View tab. For each of these classes corresponding header file (.h) and an implementation file (.cpp) is generated.

The MainFrame Window Class

This class is named CMainFrame and is derived from the MFC class CFrameWnd. This class manages the main window of the application that contains the window frame, Menubar, Toolbar, Status bar, System menu and Minimise, Maximise and Close boxes and also contains the view window of the application.

The Application Class

This class named C ProjectNameApp is derived from the MFC class CWinApp. It manages the application as a whole by managing the tasks, like initializing the application instance, managing the message loop and performing final cleanup of the program.

The Document Class

This class named C ProjectNameDoc is derived from the MFC class CDocument. It is responsible for storing the program data as a single string and reading and writing this data to disk files.

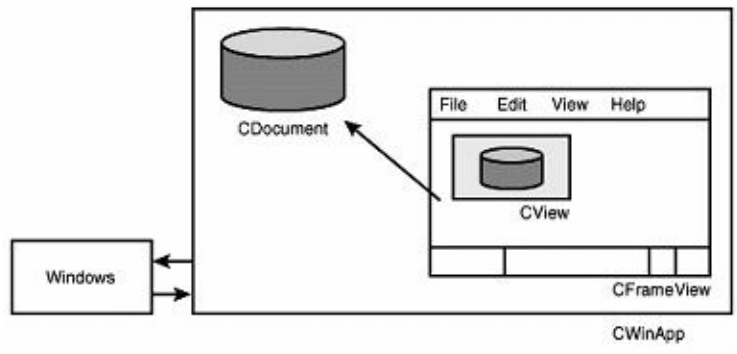
The View Class

The class named C projectNameView is derived from CView class from the MFC classes. It manages the view window, which is used to display program data on the screen and also used in processing the input from the user.

MFC and AppWizard use the Document/View architecture to organize programs written for Windows. Document/View separates the program into four main classes:

- A document class derived from CDocument
- A view class derived from CView
- A frame class derived from CFrameWnd
- An application class derived from CWinApp

Each of these classes has a specific role to play in an MFC Document/View application. The document class is responsible for the program's data. The view class handles interaction between the document and the user. The frame class contains the view and other user interface elements, such as the menu and toolbars. The application class is responsible for actually starting the program and handling some general-purpose interaction with Windows. Figure shows the four main parts of a Document/View



The Document/View architecture.

Although the name "Document/View" might seem to limit you to only word-processing applications, the architecture can be used in a wide variety of program types. There is no limitation as to the data managed by CDocument; it can be a word processing file, a spreadsheet, or a server at the other end of a network connection providing information to your program.

Likewise, there are many types of views. A view can be a simple window, as used in the simple SDI applications presented so far, or it can be derived from CFormView, with all the capabilities of a dialog box. You will learn about form views in Section 23, "Advanced Views."

SDI and MDI Applications

There are two basic types of Document/View programs:

- SDI, or Single Document Interface
- MDI, or Multiple Document Interface

An SDI program supports a single type of document and almost always supports only a single view. Only one document can be open at a time. An SDI application focuses on a particular task and usually is fairly straightforward.

Several different types of documents can be used in an MDI program, with each document having one or more views. Several documents can be open at a time, and the open document often uses a customized toolbar and menus that fit the needs of that particular document.

Why Use Document/View?

The first reason to use Document/View is that it provides a large amount of application code for free. You should always try to write as little new source code as possible, and that means using MFC classes and letting AppWizard and ClassWizard do a lot of the work for you. A large amount of the code that is written for you in the form of MFC classes and AppWizard code uses the Document/View architecture.

The Document/View architecture defines several main categories for classes used in a Windows program. Document/View provides a flexible framework that you can use to create almost any type of Windows program. One of the big advantages of the Document/View architecture is that

it divides the work in a Windows program into well-defined categories. Most classes fall into one of the four main class categories:

- Controls and other user-interface elements related to a specific view
- Data and data-handling classes, which belong to a document
- Work that involves handling the toolbar, status bar, and menus, usually belonging to the frame class
- Interaction between the application and Windows occurring in the class derived from CWinApp

Dividing work done by your program helps you manage the design of your program more effectively. Extending programs that use the Document/View architecture is fairly simple because the four main Document/View classes communicate with each other through well- defined interfaces. For example, to change an SDI program to an MDI program, you must write little new code. Changing the user interface for a Document/View program impacts only the view class or classes; no changes are needed for the document, frame, or application classes.

Documents

Documents are the basic elements that are created and manipulated by the application. Windows provides a graphic interface that gives a user a natural way to use the application. To implement this interface, the developer has to provide ways to see and interact with the information that the application creates and uses. A document is simply a place to collect common data elements that form the processing unit for the application.

Documents and the Application Relationship

The application class uses documents as a way to organize and present information to the user. Each application derived from the MFC defines at least one type of document that is a part of the application. The type of document and the number of documents that the application uses are

defined in the code of the application class.

As we have already discussed, MFC supports two types of applications - MDI and SDI. MDI - In applications that support Multiple Document Interface, a number of text files can be opened for editing at once; each in a different window. Each of the open files has a corresponding document. Also, in MDI, the same document can have multiple views, where a window is split. Each pane of the window can show a different portion of the data whereas this data is coming from a common source, the document. SDI - In applications with Single Document Interface only one document is open at a time. A SDI application does not have a Window menu and the File menu does not have a Close option because only one document can be open at a time, opening a document automatically closes the current document. These are two common characteristics of a SDI application.

Why Documents

The fundamental responsibility of the document is to store all the elements that constitute an application unit. An application may support more than one type of data like numbers, text, drawings etc. The document also controls all the views associated with it. As the user opens and manipulates windows on the screen, views are created and the document is associated with each view. The document is responsible for controlling and updating the elements within a given view.

The view may either request the document to draw its components or directly request the components to draw themselves. It must provide a device context where the drawing occurs. All elements of a drawing must be able to display themselves correctly upon request.

Views

The view is the user's window to the document. The user interacts with a document using the view. Each active document will have one or more active views available on the display. Generally, each view is displayed in a single window.

Why Views

The view gives the document a place to display information. It is the intermediary between the

document, which contains the information and the user. The view organizes and displays the document information onto the screen or printer and takes in the user input as information or operation on the document. The view is the active area of the document.

It has two functions -

- Acts as the display area for the document data and
- Acts as the input area where the user interacts with the document, normally providing additional commands or data for the document to process.

All the information passes through the view before reaching the document. Therefore, if an application handles mouse messages in functions like OnLButtonDown(), it first receives the message, translates the information into an appropriate form and then calls the required document function to process the mouse command.

Although view is responsible for displaying the document and its information, there are two possibilities -

- Let the view directly access the document's data elements
- Create a document function that handles access to the appropriate data member

The choice of writing directly in the view has two advantages -

- It is the responsibility of the view to handle document display, so having the code there is more appropriate
- Placing the code in the view keeps all the code regarding the device context in the view where it seems natural.

A view is always tied to a single document. One document can have several views. Opening another window on the document can create two views of the same document. Alternative ways can be presented to look at the same information by implementing different types of views for the document. For example, as we discussed in the beginning of the chapter, in MS Excel same data can be viewed either in the form of a table or in the form of different graphs.

Gaining Access to Document Data from the View

The view accesses its document's data either with the GetDocument() function, which returns a pointer to the document or by making the view class a C++ friend of the document class. The view then uses its access to the data to obtain the data when it is ready to draw or otherwise manipulate it. For example, from the view's OnDraw() member function, the view uses GetDocument() to obtain a document pointer. Then it uses that pointer to access a CString data member in the document. The view passes the string to the TextOut() function.

M.A.M COLLEGE OF ENGINEERING

CS1308 VISUAL PROGRAMMING LAB

DEGREE / BRANCH**: B.TECH / IT** YEAR / SEM**: III/V**

TABLE OF CONTENTS

1. SIMPLE WINDOW PROGRAM

AIM:

To write a windows SDK program to display a text in the client area using VC++.

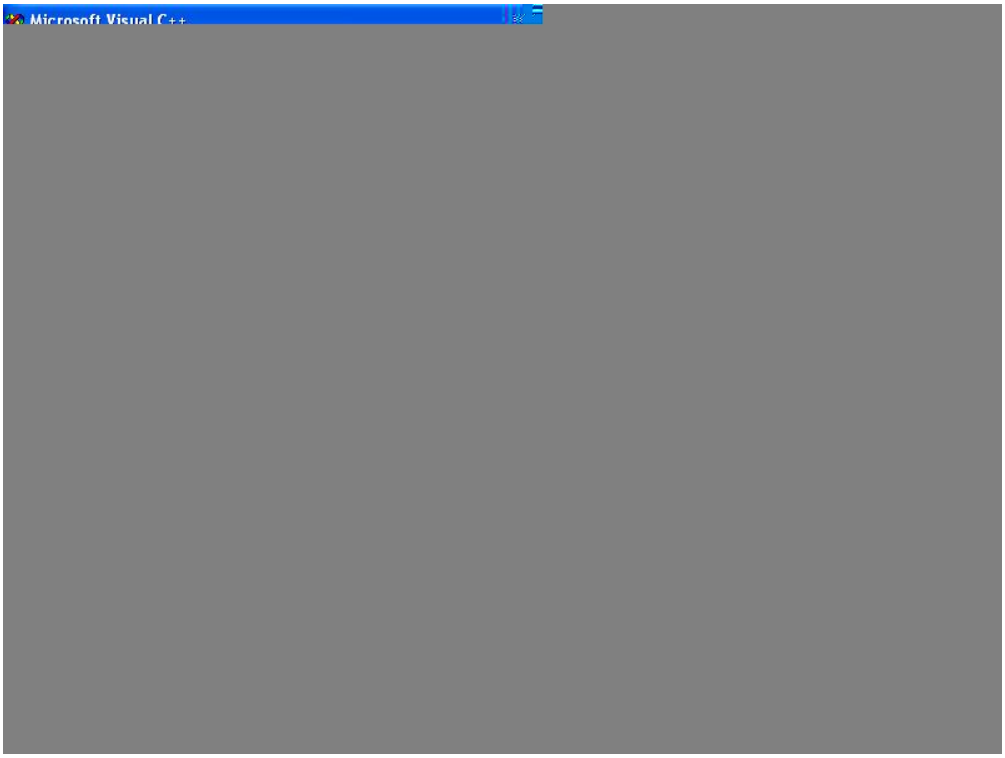
CONCEPT:

This program creates a normal application window, and displays, “Welcome to Windows Environment” in the center of that window. WINDOWS.H is a master include file that includes other Windows header files, some of which also include other header files. These header files define all the Windows data types, function calls, data structures, and constant identifiers.

PROCEDURE:

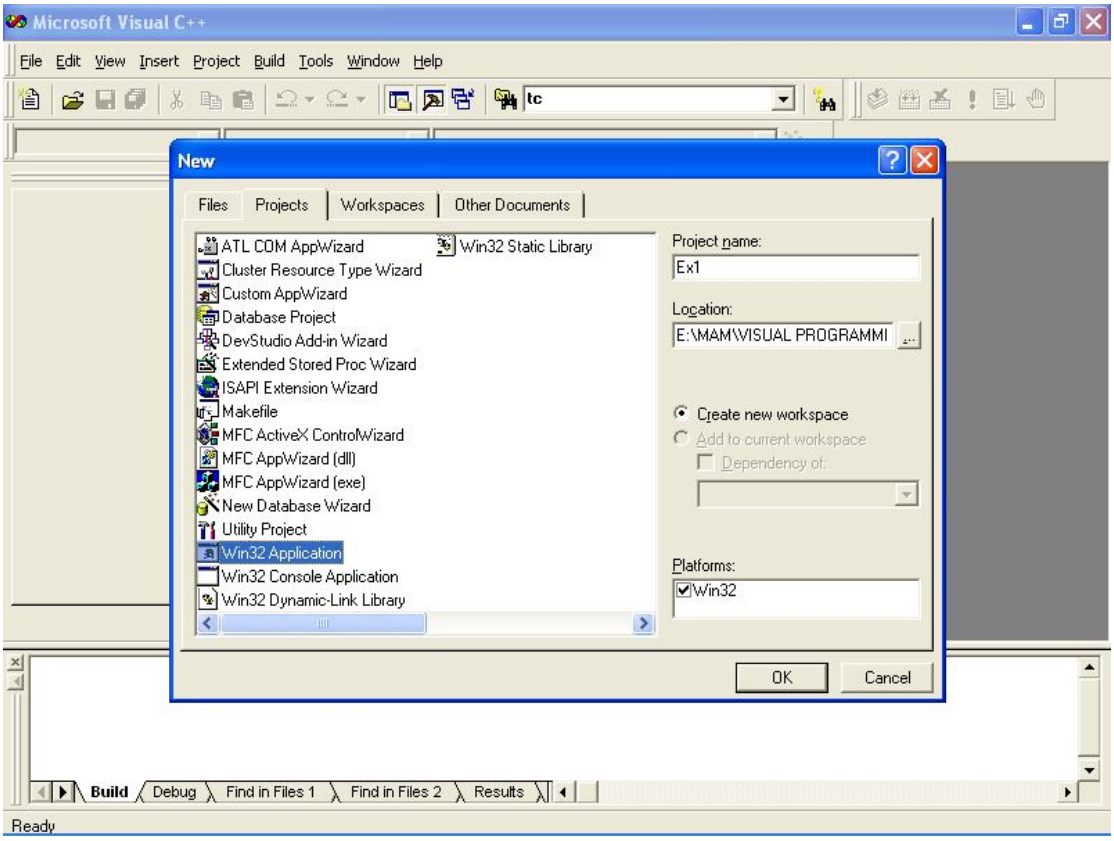
STEP 1:

Open visual studio 6.0 → Visual C++ 6.0

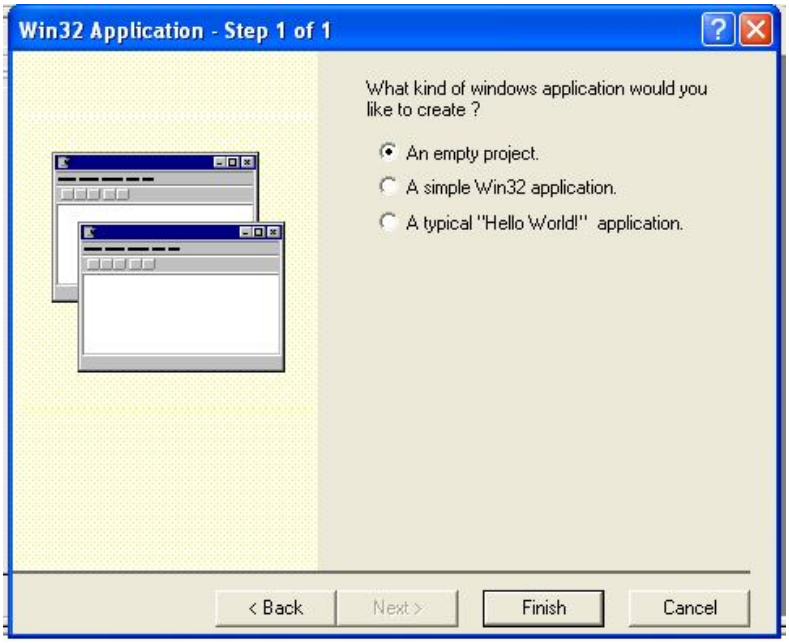


STEP 2:

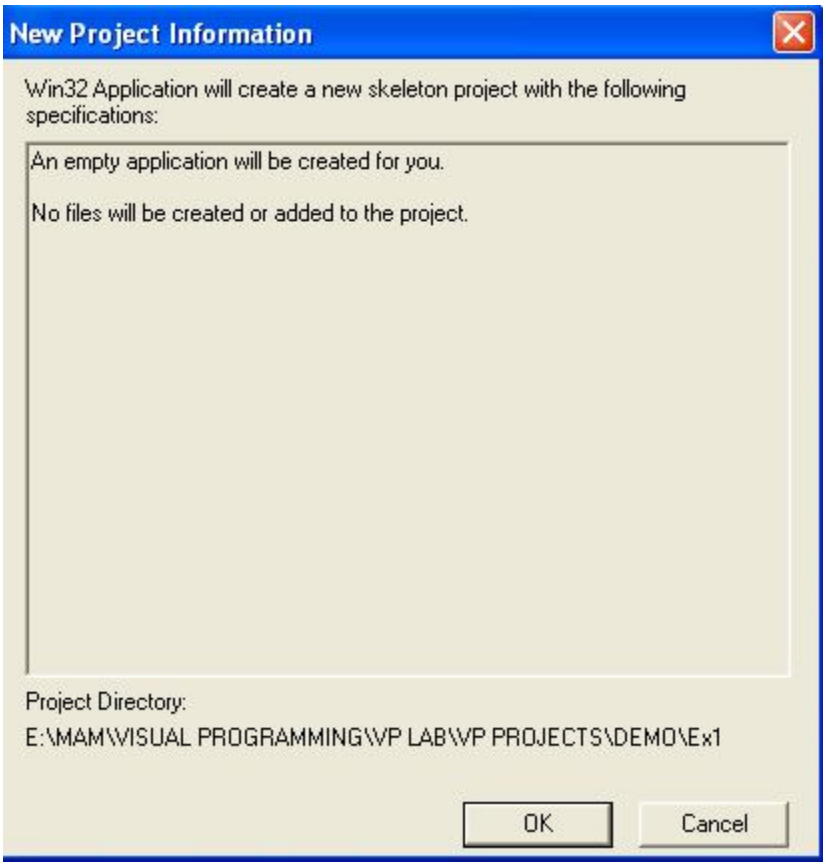
- Select New from the File menu.
- In the New dialog box, pick the Projects tab.
- Select Win32 Application.
- In the Location field, select a directory.
- In the Project Name field, type the name of the project, “Ex1”.
- This will be a subdirectory of the directory indicated in the Location field.
- The Create New Workspace button should be checked.
- The Platforms section should indicate Win32.
- Choose OK.



STEP 3:
A dialog box labeled Win32 Application - Step 1 of 1 will appear. Choose “An empty project” to create an Empty Project, and press the Finish button.

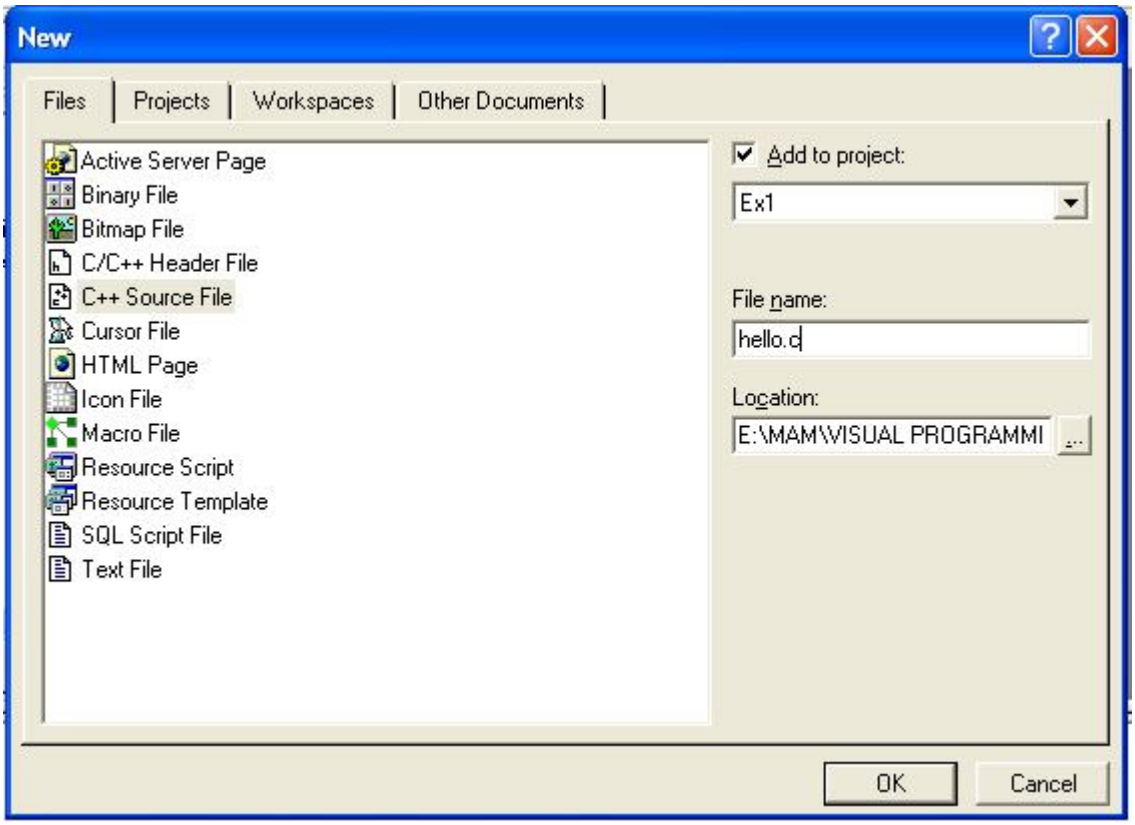

Win32 Application - Step 1 of 1
What kind of windows application would you like to create ?
An empty project.
A simple Win32 application.
A typical "Hello World!" application.
< Back
Next >
Finish
Cancel

STEP 4:

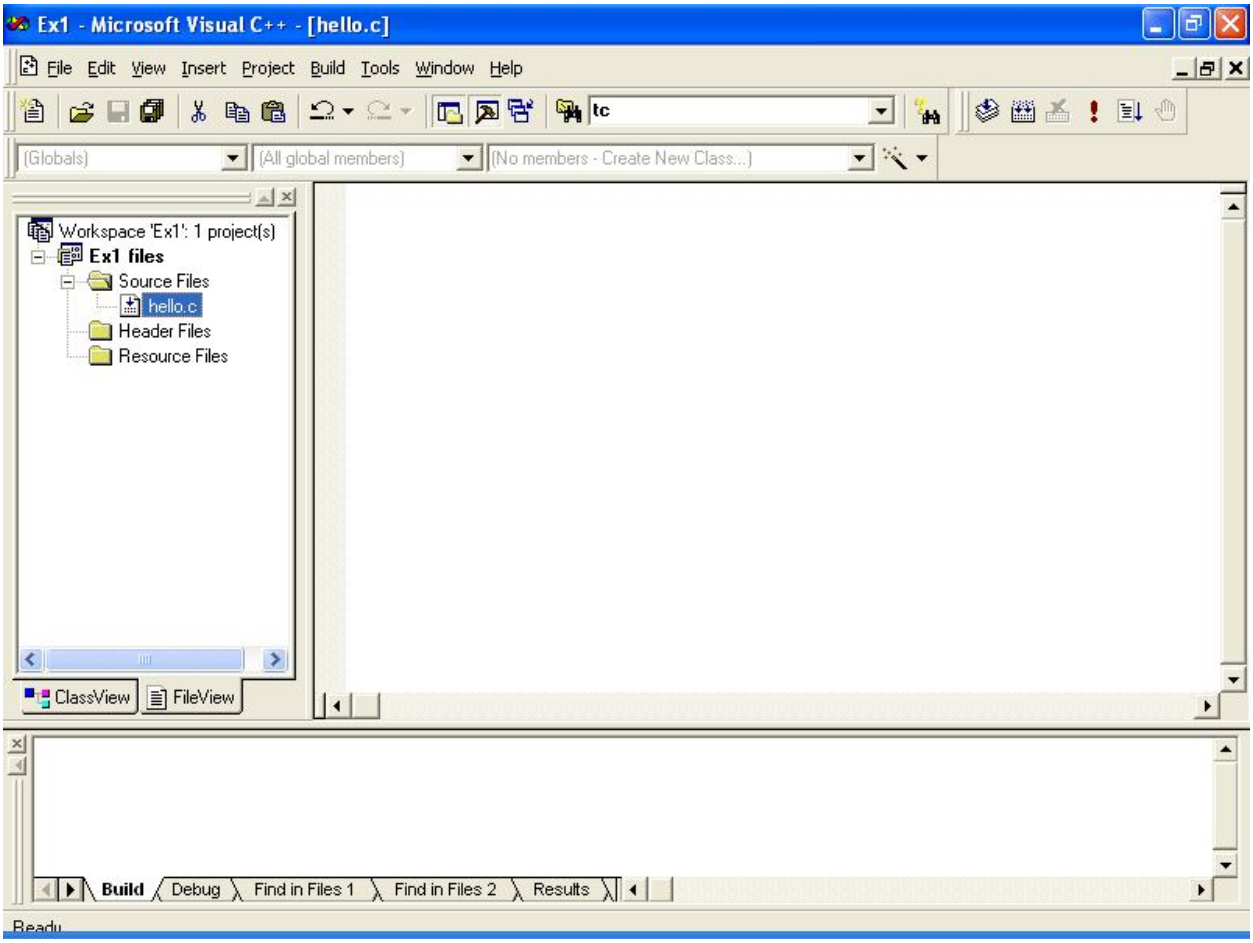

New Project Information
Win32 Application will create a new skeleton project with the following specifications:
An empty application will be created for you.
No files will be created or added to the project.
Project Directory:
E:\MAM\VISUAL PROGRAMMING\VP LAB\VP PROJECTS\DEMO\Ex1
OK
Cancel

STEP 5:

- Select New from the File menu again.
- In the New dialog box, pick the Files tab.
- Select C++ Source File.
- The Add To Project box should be checked, and Project “Ex1” should be selected.
- Type filename “hello.c” in the File Name field.
- Choose OK.



STEP 6:



STEP 7:

- In the hello.C should begin with preprocessor directive
- Specify the prototype for windows procedure *WndProc* that handles the window message.
- Implement the *Winmain* function which instantiates the *wndclass* to create the window.
- Implement the window procedure *WndProc* to handle the message in the message queue.

PROGRAM:

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow )
```

```
{
 static  TCHAR szAppName[]=TEXT ("HELLO");
HWND hwnd;
MSG msg;
// DECLARING AND INTIAZING THE WINDOW CLASS
WNDCLASS wc;                                    // Window class declaration
wc.style = CS_HREDRAW| CS_VREDRAW;              // Window style
wc. lpfnWndProc = WndProc;                      // Window Procedure
wc.cbClsExtra = 0;                              // Extra Parameters
wc.cbWndExtra = 0;                              // Extra Parameters
wc.hInstance = hInstance;                       // Instance handle
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION); // Icon
wc.hCursor = LoadCursor (NULL, IDC_ARROW);  // Window Cursor
wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
                                                // Set Background Color
wc.lpszMenuName = NULL;                         // Menu Handle
wc.lpszClassName = szAppName;                   // Class Name
// REGISTERING THE WINDOW CLASS
if (!RegisterClass (&wc))
{
        MessageBox(NULL, TEXT("This program requires WindowsNT!"),
szAppName, MB_ICONERROR);
        return 0;
}
// CREATING THE WINDOW
hwnd = CreateWindow ( szAppName,                        // window class name
                      TEXT("The Hello Program"),        // window caption
```

```
                    WS_OVERLAPPEDWINDOW,      // window style
                    CW_USEDEFAULT,            // initial x position
                    CW_USEDEFAULT,            // initial x position
                    CW_USEDEFAULT,            // initial x position
                    CW_USEDEFAULT,            // initial x position
                    NULL,                     // parent window handle
                    NULL,                     // window menu handle
                    hInstance,                //program instance handle
                    NULL);                    // creation parameters
```

// DISPLAYING THE WINDOW

```
    ShowWindow (hwnd, iCmdShow);
    UpdateWindow (hwnd);
```

// PROCESSING THE MESSAGE LOOP

```
    while (GetMessage(&msg,NULL,0,0))      // extracts message from Message Queue
    {
        TranslateMessage (&msg);           // dispatches the message received
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

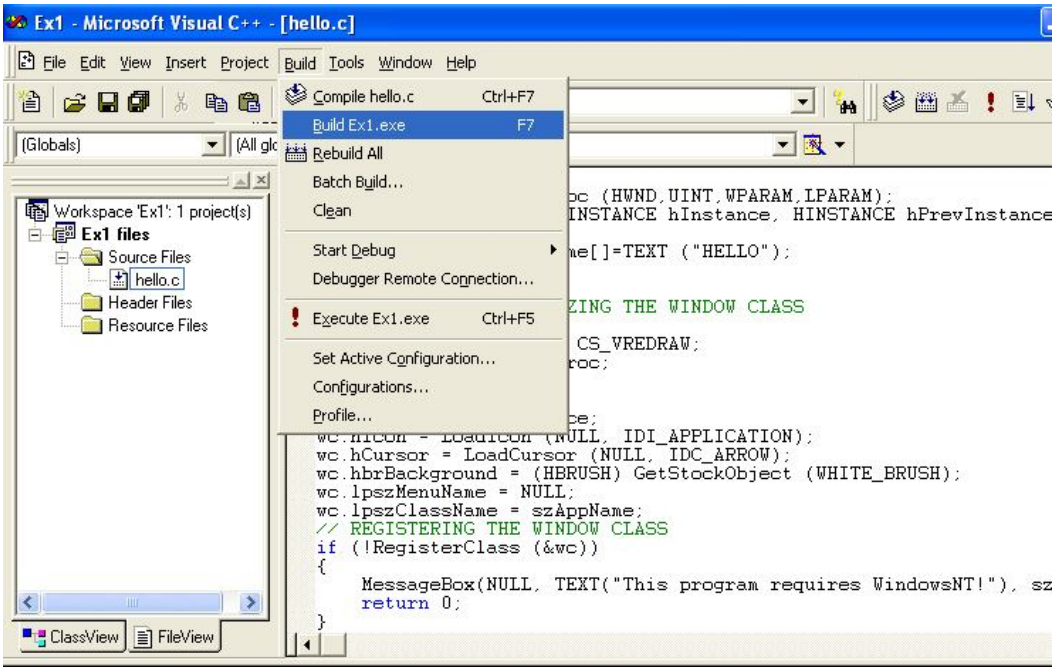
// WINDOW PROCEDURE FUNCTION

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wparam,
LPARAM lparam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
```

```
        switch (message)
        {
        case WM_PAINT:                  // Used to draw on client area using objects
                hdc = BeginPaint (hwnd, &ps);       // Gets Device Context
                GetClientRect (hwnd, &rect);        // To get Client area size
                DrawText (hdc, TEXT ("HELLO WINDOWS 98"),-1, &rect,
DT_SINGLELINE | DT_CENTER | DT_VCENTER);
                EndPaint(hwnd,&ps);                 // Releases Device Context
                return 0;

        case WM_DESTROY:                // Used to close the window
                PostQuitMessage(0);         // used to terminate while loop in WinMain()
                return 0;
        }
        return DefWindowProc (hwnd, message, wparam, lparam);
}
```

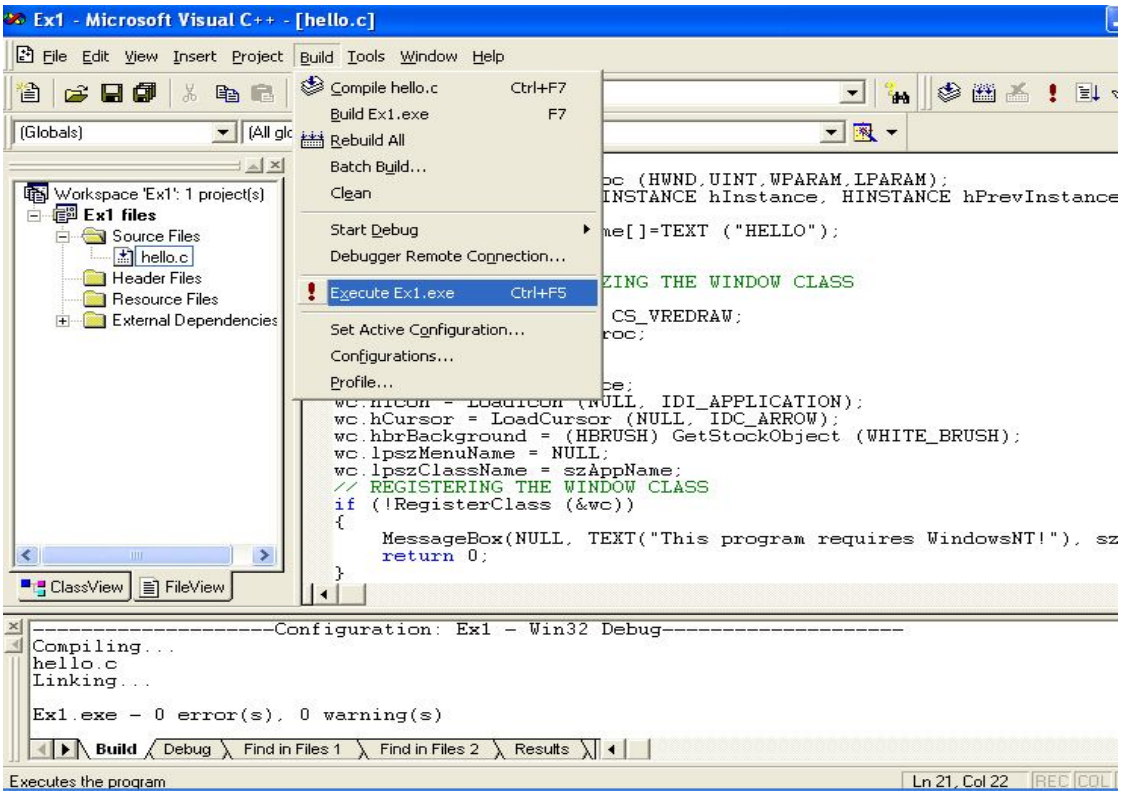
STEP 8:

Now select Build Hellomsg.exe from the Build menu, or press F7, or select the Build icon from the Build toolbar.



STEP 9:

Then select Execute Hellomsg.exe from the Build menu, or press Ctrl+F5, or click the Execute Program icon (which looks like a red exclamation point) from the Build toolbar.



Sample Output:

DESCRIPTION:

1. In WM_PAINT message, use *BeginPaint* and *EndPaint* functions to handle the window, which is passed to the window procedure as an argument, and the address of a structure variable of type PAINTSTRUCT, which is defined in the WINUSER.H header file. Also use *DrawText* function is to draw the character string in the center of the client area.

2. In WM_DESTROY message, use *PostQuitMessage (0)* function to inserts a WM_QUIT message in the program's message queue. Genrally *GetMessage* returns nonzero for any message other than WM_QUIT that it retrieves from the message queue. When *GetMessage* retrieves a WM_QUIT message, *GetMessage* returns 0. This causes *WinMain* to drop out of the message loop. And also use

return msg.wParam

The *wParam* field of the structure is the value passed to the *PostQuitMessage* function (generally 0). The return statement exits from *WinMain* and terminates the program.

Result:

Thus program to display text in the center of window has been developed, build and executed using Win32 application.

2. MOUSE EVENT I

Aim:

To write the windows SDK program to handle the mouse events to process each mouse message to display a message in the window.

Concept:

This program to shows some simple mouse processing to get a good feel and how the Windows sends mouse messages to the program. The window procedure processes each mouse message as it comes and then quickly returns control to Windows. It processes three mouse messages:

- WM_LBUTTONDOWN – Displays a message box
- WM_RBUTTONDOWN – Displays a message box

Steps to build MouseEvents application:

1. Select New from the File menu. In the New dialog box, pick the Projects tab.
2. Select Win32 Application. In the Location field, select a subdirectory. In the Project Name field, type the name of the project, MouseEvents.
    - This will be a subdirectory of the directory indicated in the Location field. The Create New Workspace button should be checked. The Platforms section should indicate Win32. Choose OK.
3. A dialog box labeled Win32 Application - Step 1 of 1 will appear. Indicate that you want to create an Empty Project, and press the Finish button.
4. Select New from the File menu again. In the New dialog box, pick the Files tab. Select C++ Source File. The Add To Project box should be checked, and MouseEvents should be indicated. Type MouseEvents.c in the File Name field. Choose OK.
5. Specify the prototype for windows procedure that handles the window message.
6. Implement the *Winmain* function which instantiates the *wndclass* to create the window.

7. Implement the window procedure to handle the message in the message queue.

8. Build and execute the MouseEvents application.

Program:

MouseEvents.C

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow )

{

 static  TCHAR szAppName[]=TEXT ("HELLO");

HWND hwnd;

MSG msg;

// DECLARING AND INTIAZING THE WINDOW CLASS

WNDCLASS wc;                                    // Window class declaration

wc.style = CS_HREDRAW| CS_VREDRAW;              // Window style

wc. lpfnWndProc = WndProc;                      // Window Procedure

wc.cbClsExtra = 0;                              // Extra Parameters

wc.cbWndExtra = 0;                              // Extra Parameters

wc.hInstance = hInstance;                       // Instance handle

wc.hIcon = LoadIcon (NULL, IDI_APPLICATION); // Icon

wc.hCursor = LoadCursor (NULL, IDC_ARROW);  // Window Cursor

wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);

                                                // Set Background Color

wc.lpszMenuName = NULL;                         // Menu Handle

wc.lpszClassName = szAppName;                   // Class Name
```

```
// REGISTERING THE WINDOW CLASS
if (!RegisterClass (&wc))
{
        MessageBox(NULL, TEXT("This program requires WindowsNT!"), szAppName, MB_ICONERROR);
        return 0;
}
// CREATING THE WINDOW
hwnd = CreateWindow ( szAppName,                    // window class name
                      TEXT("The Hello Program"),    // window caption
                      WS_OVERLAPPEDWINDOW,          // window style
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      NULL,                         // parent window handle
                      NULL,                         // window menu handle
                      hInstance,                    //program instance handle
                      NULL);                        // creation parameters
// DISPLAYING THE WINDOW
        ShowWindow (hwnd, iCmdShow);
        UpdateWindow (hwnd);
// PROCESSING THE MESSAGE LOOP
        while (GetMessage(&msg,NULL,0,0))       // extracts message from Message Queue
        {
                TranslateMessage (&msg);
                DispatchMessage (&msg);         // dispatches the message received
        }
```

```
        return msg.wParam;
    }
```

// WINDOW PROCEDURE FUNCTION

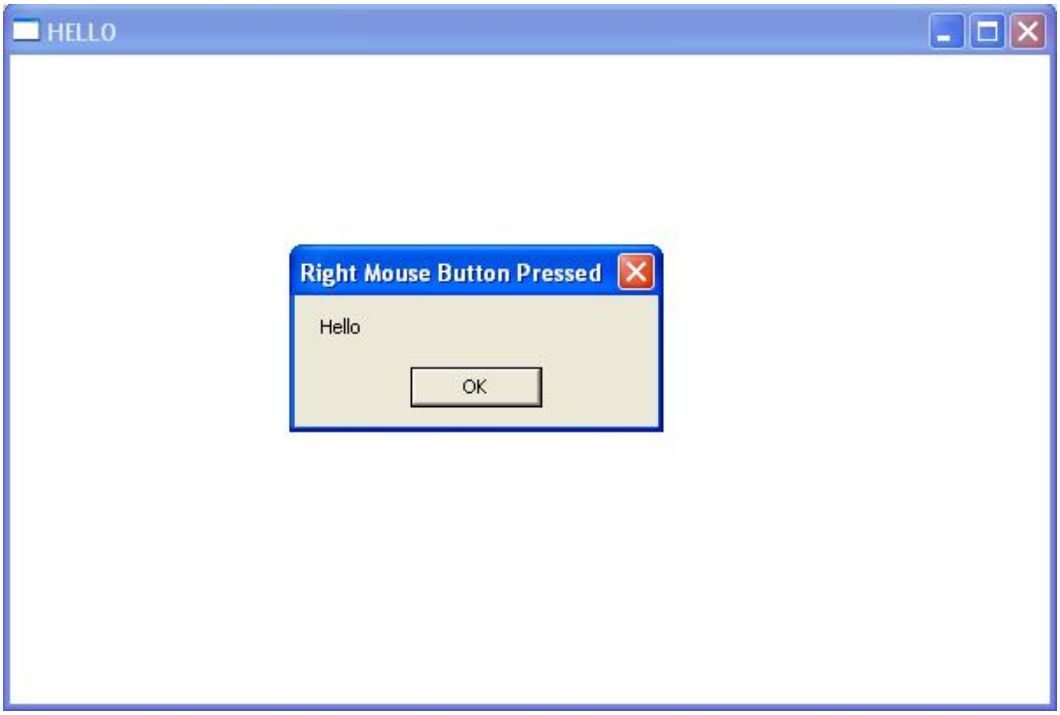
```
LRESULT CALLBACK WndProc (HWND hwnd,UINT message,WPARAM
wparam,LPARAM lparam)
{
    char left[]="How are u?";
    char right[]="Hello";
    switch (message)
    {
    case WM_LBUTTONDOWN:        // Left mouse button press message
        MessageBox(GetFocus(),left,"Left Mouse Button Pressed",MB_OK);
        return 0;
    case WM_RBUTTONDOWN:        // Right mouse button press message
        MessageBox(GetFocus(),right,"Right Mouse Button Pressed",MB_OK);
        return 0;
    case WM_DESTROY:            // used to close the window
        PostQuitMessage(0);     // used to terminate while loop in WinMain()
        return 0;
    }
    return DefWindowProc(hwnd,message,wparam,lparam);
}
```

Sample Output:

- Bring the mouse cursor into the client area, press the left button, it displays the message box with the greeting message also indicating that left button was pressed in the title bar of the message box.

- Now press the right button by bringing the mouse cursor in the client area, it displays the message box with the greeting message also indicating that right button was pressed in the title bar of the message box

- If you move the mouse cursor out of the client area and press any mouse button, it does not display the message box because window procedure doesn't receive any window message.



Description:

WM_LBUTTONDOWN:

This message is generated when the Left mouse button was pressed in the client area.

WM_RBUTTONDOWN:

This message is generated when the Left mouse button was pressed in the client area.

WM_DESTROY

This message is used to terminate the window when close button of the window is pressed.

MessageBox : Displays a message

a. First parameter – Handle to a window

b. Second parameter – Message

c. Third parameter – Message box title

d. Fourth parameter – Used to display type of button to be present in the message box.

Result:

Thus the program to handle the mouse events to process each mouse message has been developed, builds and executed using Win32 application.

3. MOUSE EVENT II

AIM:

To write the windows SDK program to handle the mouse events to process each mouse message to display a message in the window.

Concept:

This program to shows some simple mouse processing to get a good feel and how the Windows sends mouse messages to the program. The window procedure processes each mouse message as it comes and then quickly returns control to Windows. It processes three more mouse messages:

- WM_MOUSEMOVE - Draw text on the client area
- WM_NCMOUSEMOVE – Draw text on the client area
- WM_NCLBUTTONDOWN - Draw text on the client area
- WM_NCRBUTTONDOWN - Draw text on the client area

Steps to build MouseEvents2 application:

9. Select New from the File menu. In the New dialog box, pick the Projects tab.

10. Select Win32 Application. In the Location field, select a subdirectory. In the Project Name field, type the name of the project, MouseEvents2.

    - This will be a subdirectory of the directory indicated in the Location field. The Create New Workspace button should be checked. The Platforms section should indicate Win32. Choose OK.

11. A dialog box labeled Win32 Application - Step 1 of 1 will appear. Indicate that you want to create an Empty Project, and press the Finish button.

12. Select New from the File menu again. In the New dialog box, pick the Files tab. Select C++ Source File. The Add To Project box should be checked, and MouseEvents2 should be indicated. Type MouseEvents2.c in the File Name field. Choose OK.

13. Specify the prototype for windows procedure that handles the window message.

14. Implement the *Winmain* function which instantiates the *wndclass* to create the window.

15. Implement the window procedure to handle the message in the message queue.

16. Build and execute the MouseEvents application.

Program:

MouseEvents2.C

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow )

{

 static  TCHAR szAppName[]=TEXT ("Mouse Event 2");

HWND hwnd;

MSG msg;

// DECLARING AND INTIAZING THE WINDOW CLASS

WNDCLASS wc;                                    // Window class declaration

wc.style = CS_HREDRAW| CS_VREDRAW;              // Window style

wc. lpfnWndProc = WndProc;                      // Window Procedure

wc.cbClsExtra = 0;                              // Extra Parameters

wc.cbWndExtra = 0;                              // Extra Parameters

wc.hInstance = hInstance;                       // Instance handle

wc.hIcon = LoadIcon (NULL, IDI_APPLICATION); // Icon

wc.hCursor = LoadCursor (NULL, IDC_ARROW);  // Window Cursor

wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);

                                                // Set Background Color
```

```
wc.lpszMenuName = NULL;                    // Menu Handle
wc.lpszClassName = szAppName;              // Class Name
// REGISTERING THE WINDOW CLASS
if (!RegisterClass (&wc))
{
        MessageBox(NULL, TEXT("This program requires WindowsNT!"),
szAppName, MB_ICONERROR);
        return 0;
}
// CREATING THE WINDOW
hwnd = CreateWindow ( szAppName,                    // window class name
                      TEXT("Mouse Event 2"),        // window caption
                      WS_OVERLAPPEDWINDOW,          // window style
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      CW_USEDEFAULT,                // initial x position
                      NULL,                         // parent window handle
                      NULL,                         // window menu handle
                      hInstance,                   //program instance handle
                      NULL);                        // creation parameters
// DISPLAYING THE WINDOW
        ShowWindow (hwnd, iCmdShow);
        UpdateWindow (hwnd);
// PROCESSING THE MESSAGE LOOP
        while (GetMessage(&msg,NULL,0,0))      // extracts message from Message Queue
        {
                TranslateMessage (&msg);
```

```
            DispatchMessage (&msg);          // dispatches the message received
        }
        return msg.wParam;
    }
```

// WINDOW PROCEDURE FUNCTION

```
RESULT CALLBACK WndProc (HWND hwnd, UINT wm, WPARAM wp, LPARAM lp)
{
    switch (wm)
    {
    case WM_MOUSEMOVE:
        TextOut ( hdc, 10, 10,"MOUSE MOVED ON CLIENT AREA",26);
        break;
    case WM_NCMOUSEMOVE:
        TextOut (hdc, 10, 30,"MOUSE MOVED ON NON-CLIENT AREA",30);
        break;
    case WM_PAINT:
        hdc=BeginPaint(hwnd,&ps);
        break;
    case WM_LBUTTONDOWN:
        TextOut (hdc, 10, 60,"PRESSED LEFT MOUSEBUTTON ON CLIENT AREA",
        39);
        break;
    case WM_NCLBUTTONDOWN:
        TextOut (hdc, 10, 90," PRESSED LEFT MOUSEBUTTON ON NON-CLIENT
        AREA", 44);
```

```
            break;
        case WM_RBUTTONDOWN:
            TextOut (hdc, 10, 120,"PRESSED RIGHT MOUSEBUTTON ON CLIENT
            AREA",40);
            break;
        case WM_NCRBUTTONDOWN:
            TextOut (hdc, 10, 150," PRESSED RIGHT MOUSEBUTTON ON NON-
            CLIENT AREA",45);
            break;
        case WM_DESTROY:            // used to close the window
            PostQuitMessage(0);     // used to terminate while loop in WinMain()
            return 0;
        }
        return DefWindowProc(hwnd,message,wparam,lparam);
}
```

Sample Output:

- Bring the mouse cursor into the client area, “MOUSE MOVED ON CLIENT AREA” message will be displayed on the client area.

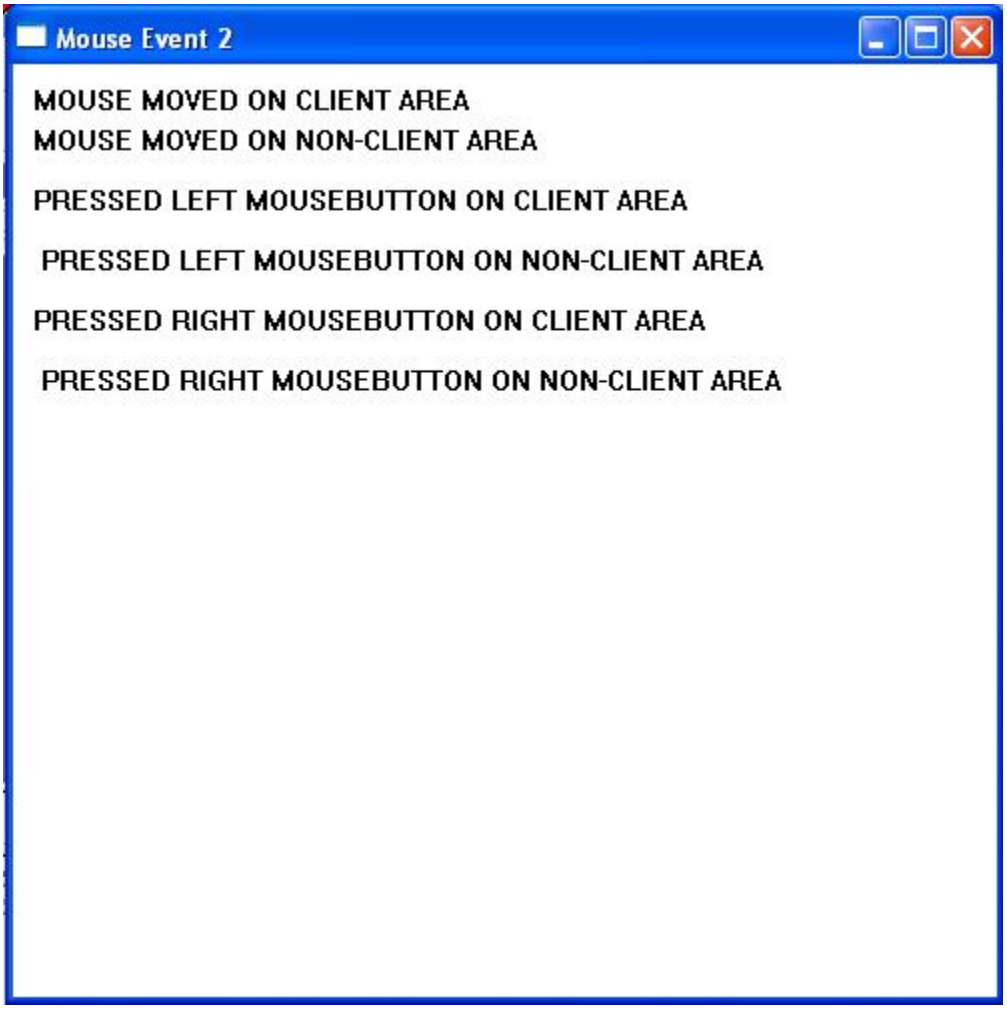
- Move the mouse cursor in title bar of the window, “MOUSE MOVED ON NON CLIENT AREA” message will be displayed on the client area.

- If you press the left mouse button, while keeping the mouse cursor on the title bar of the window i.e. Non client area, “PRESSED LEFT MOUSE BUTTION ON NON CLIENT AREA” message will be displayed on the client area and if you press the right button,

"PRESSED RIGHT MOUSE BUTTION ON NON CLIENT AREA" message will be displayed

- If you move the mouse back into the client area and press the left button again, PRESSED LEFT MOUSE BUTTION ON CLIENT AREA" message will be displayed on the client area and if you press the right button, "PRESSED RIGHT MOUSE BUTTION ON CLIENT AREA" message will be displayed

- If you move the mouse cursor out of the client area and press any mouse button, it does not display the message box because window procedure doesn't receive any window message.



Description:

WM_MOUSEMOVE:

This message is generated when the mouse moved on client area.

WM_NCMOUSEMOVE:

This message is generated when the mouse moved on non client area i.e. Tiltlebar, Statusbar, etc.

WM_LBUTTONDOWN:

This message is generated when the Left mouse button was pressed in the client area.

WM_NCLBUTTONDOWN:

This message is generated when the Left mouse button was pressed in the non client area.

WM_RBUTTONDOWN:

This message is generated when the Right mouse button was pressed in the client area.

WM_NCRBUTTONDOWN:

This message is generated when the Right mouse button was pressed in the client area.

WM_PAINT:

Makes the client area valid

WM_DESTROY

This message is used to terminate the window.

TextOut: displays a text on the client area.

e. First parameter – Device Context Handle
f. Second parameter – x position
g. Third parameter – y position
h. Fourth parameter – Actual Message
i. Fifth parameter – Message size.

Result:

Thus the program to handle the mouse events to process each mouse message has been developed, builds and executed using Win32 application.

4. KEYBOARD EVENT

Aim:

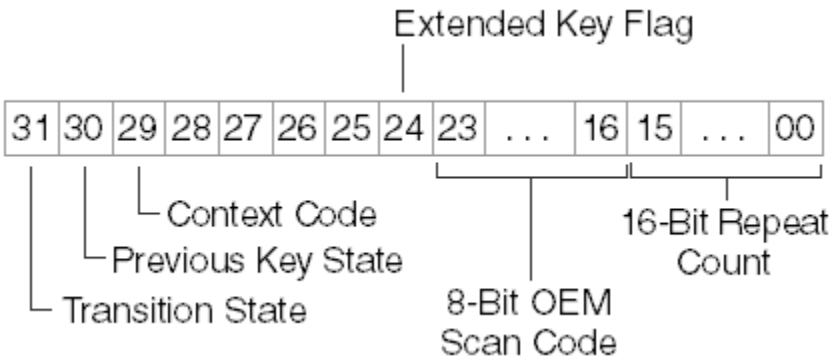
To write the windows SDK program to handle the keyboard events to display the different keyboard messages.

Concept:

This program displays the information in the client area that Windows sends the window procedure for the eight different keyboard messages. It displays the contents of each keystroke and character message that it receives in its window procedure. It saves the messages in an array of MSG structures. The size of the array is based on the size of the maximized window size and the fixed-pitch system font. If the user resizes the video display while the program is running the array is reallocated. It uses the standard C *malloc* function to allocate memory for this array.

The* lParam *Information:

In the keystroke messages (WM_KEYDOWN, WM_KEYUP), the *wParam* message parameter contains the virtual key code and the *lParam* message parameter contains other information useful in understanding the keystroke. The 32 bits of *lParam* are divided into six fields as shown below,



Steps to build keyboard events application:

1. Select New from the File menu. In the New dialog box, pick the Projects tab.

2. Select Win32 Application. In the Location field, select a subdirectory. In the Project Name field, type the name of the project, KeyboardEvents.

   This will be a subdirectory of the directory indicated in the Location field. The Create New Workspace button should be checked. The Platforms section should indicate Win32. Choose OK.

3. A dialog box labeled Win32 Application - Step 1 of 1 will appear. Indicate that you want to create an Empty Project, and press the Finish button.

4. Select New from the File menu again. In the New dialog box, pick the Files tab. Select C++ Source File. The Add To Project box should be checked, and KeyboardEvents should be indicated. Type KeyboardEvents.c in the File Name field. Choose OK.

5. Run Win32 Application to generate win32/keyevents application.

6. Specify the prototype for windows procedure that handles the window message.

7. Implement the *Winmain* function which instantiates the *wndclass* to create the window.

8. Implement the window procedure to handle the message in the message queue.

9. Using the *TextOut* function, display the remaining six columns, this means the status of the six fields in the lParam message parameter.

10. Build and execute the keyboardEvents application.

Program:

KeyboardEvents.c

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow )

{

 static  TCHAR szAppName[]=TEXT ("Mouse Event 2");
```

```
HWND hwnd;
MSG msg;
// DECLARING AND INTIAZING THE WINDOW CLASS
WNDCLASS wc;                                    // Window class declaration
wc.style = CS_HREDRAW| CS_VREDRAW;              // Window style
wc. lpfnWndProc = WndProc;                      // Window Procedure
wc.cbClsExtra = 0;                              // Extra Parameters
wc.cbWndExtra = 0;                              // Extra Parameters
wc.hInstance = hInstance;                       // Instance handle
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION); // Icon
wc.hCursor = LoadCursor (NULL, IDC_ARROW);  // Window Cursor
wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
                                                // Set Background Color
wc.lpszMenuName = NULL;                         // Menu Handle
wc.lpszClassName = szAppName;                   // Class Name
// REGISTERING THE WINDOW CLASS
if (!RegisterClass (&wc))
{
        MessageBox(NULL, TEXT("This program requires WindowsNT!"),
szAppName, MB_ICONERROR);
        return 0;
}
// CREATING THE WINDOW
hwnd = CreateWindow ( szAppName,                    // window class name
                      TEXT("Mouse Event 2"),        // window caption
                      WS_OVERLAPPEDWINDOW,          // window style
                      CW_USEDEFAULT,                // initial x position
```

```
                    CW_USEDEFAULT,          // initial x position
                    CW_USEDEFAULT,          // initial x position
                    CW_USEDEFAULT,          // initial x position
                    NULL,                   // parent window handle
                    NULL,                   // window menu handle
                    hInstance,              //program instance handle
                    NULL);                  // creation parameters
```

// DISPLAYING THE WINDOW

```
    ShowWindow (hwnd, iCmdShow);
    UpdateWindow (hwnd);
```

// PROCESSING THE MESSAGE LOOP

```
    while (GetMessage(&msg,NULL,0,0))     // extracts message from Message Queue
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);           // dispatches the message received
    }
    return msg.wParam;
}
```

// WINDOW PROCEDURE FUNCTION

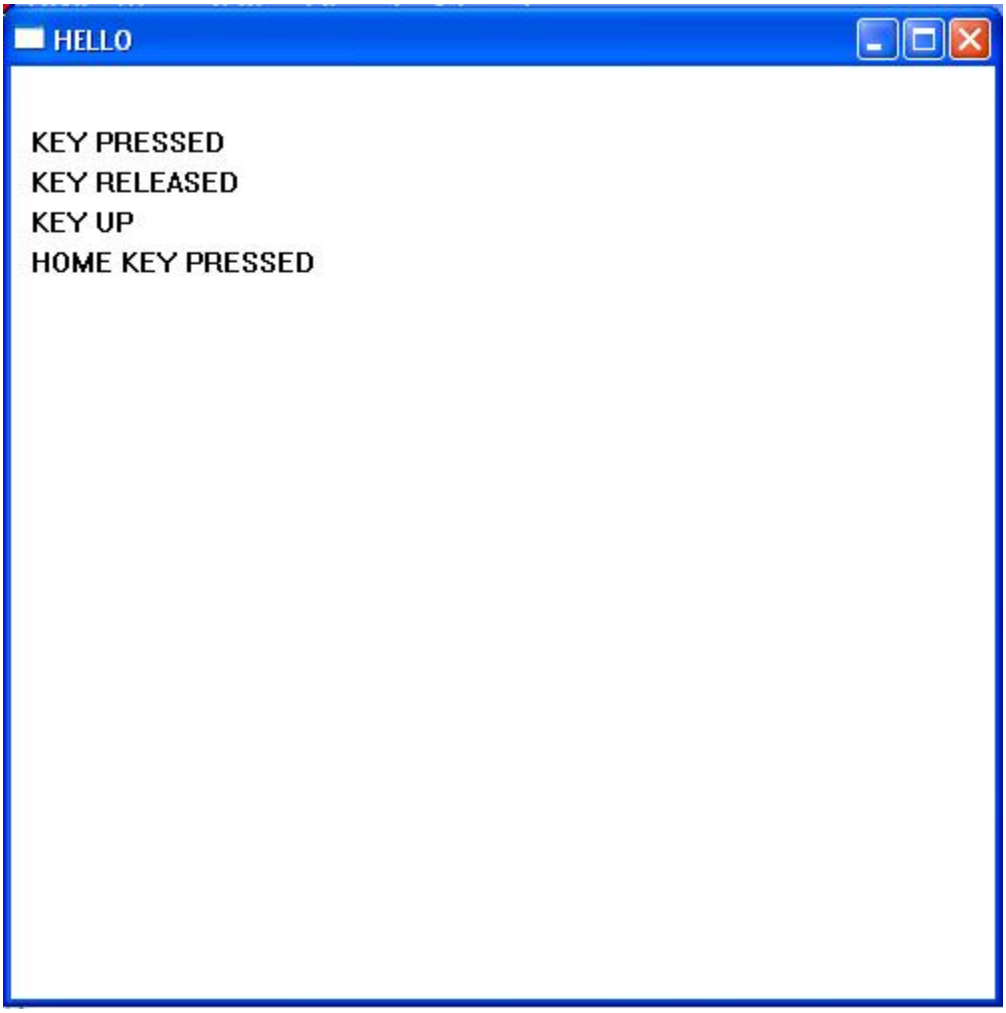
```
LRESULT CALLBACK WndProc (HWND hwnd, UINT wm, WPARAM wp, LPARAM lp)
{
    switch (wm)
    {
    case WM_PAINT:
        hdc=BeginPaint(hwnd,&ps);
        break;
```

```
case WM_KEYUP:
        TextOut (hdc, 10, 30,"KEY PRESSED",11);
        return 0;

case WM_KEYDOWN:
        TextOut (hdc, 10, 50,"KEY RELEASED",12);

        switch(wp)
        {
        case VK_HOME:
                TextOut(hdc,10,120,”HOME KEY IS PRESSED”,20);
                break;
        case VK_UP:
                TextOut (hdc, 10, 70,"KEY UP",6);
                break;
        }
        return 0;

case WM_CHAR:
        TextOut (hdc, 10, 90,"CHARACTER IS PRESSED",25);
        break;

case WM_DESTROY:
        EndPaint (hwnd, &ps);
        PostQuitMessage(0);
```

```
            return 0;
        }
        return DefWindowProc(hwnd, wm, wp, lp);
}
```

SAMPLE OUTPUT:

- Press and hold any key in the keyboard, “KEY PRESSED” message is displayed on the client area. Now release that key, you can see “KEY RELEASED” message on the client area.
- If you press the UP ARROW key, then “KEY UP” message is displayed. If you press HOME KEY then “HOME KEY PRESSED” message will be displayed.



DESCRIPTION:

WM_KEYUP:

This message is used to indicate that an UP key is pressed on the client area.

WM_KEYDOWN:

This message is used to indicate that a DOWN key is pressed on the client area.

To identify and capture the individual key, separate messages under WM_KEYDOWN message are available to identify and process the individual key function. The following messages are few of them for example.

VK_HOME:

When Home key in the keyboard is pressed, “Home Key Pressed” Message is displayed on the client area.

VK_UP:

When UP arrow key in the keyboard is pressed, “KEY UP” Message is displayed on the client area.

WM_CHAR:

This message is used to indicate that a character is pressed on the client area.

Result:

Thus the program for keyboard events to display different keyboard messages has been developed, builds and executed using Win32 application.

5. GDI

Aim:

To write the windows SDK program to draw and display the different GDI objects using GDI functions.

Concept:

This program displays various GDI objects in the client area using corresponding GDI functions. It displays the GDI objects by first capturing the device context in the GDI functions. The size and position of the GDI objects will be defined in the function call. These are the few function call used in this program.

- **Rectangle** - Draws a rectangle.
- **Ellipse** - Draws an ellipse.
- **RoundRect** - Draws a rectangle with rounded corners.
- **Pie** - Draws a part of an ellipse that looks like a pie slice.
- **Chord** - Draws part of an ellipse formed by a chord.
- **MoveToEx** – Staring point of a line
- **LineTo** – Ending point of a line.

About GDI:

The subsystem of Microsoft Windows responsible for displaying graphics on video displays and printers is known as the Graphics Device Interface (GDI). GDI is an extremely important part of Windows. Not only the applications of Windows use GDI for the display of visual information, but Windows itself uses GDI for the visual display of user interface items such as menus, scroll bars, icons, and mouse cursors.

This program is provided with the basics of drawing lines and filled areas

Steps to build GDI application:

11. Select New from the File menu. In the New dialog box, pick the Projects tab.

12. Select Win32 Application. In the Location field, select a subdirectory. In the Project Name field, type the name of the project, GDI.

    This will be a subdirectory of the directory indicated in the Location field. The Create New Workspace button should be checked. The Platforms section should indicate Win32. Choose OK.

13. A dialog box labeled Win32 Application - Step 1 of 1 will appear. Indicate that you want to create an Empty Project, and press the Finish button.

14. Select New from the File menu again. In the New dialog box, pick the Files tab. Select C++ Source File. The Add To Project box should be checked, and KeyboardEvents should be indicated. Type gdi.c in the File Name field. Choose OK.

15. Run Win32 Application to generate win32/GDI application.

16. Specify the prototype for windows procedure that handles the window message.

17. Implement the *Winmain* function which instantiates the *wndclass* to create the window.

18. Implement the window procedure to handle the message in the message queue.

19. Build and execute the keyboardEvents application.

Program:

gdi.c

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND,UINT,WPARAM,LPARAM);

int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, int iCmdShow )

{

 static  TCHAR szAppName[]=TEXT ("Graphical Device Interface");

HWND hwnd;
```

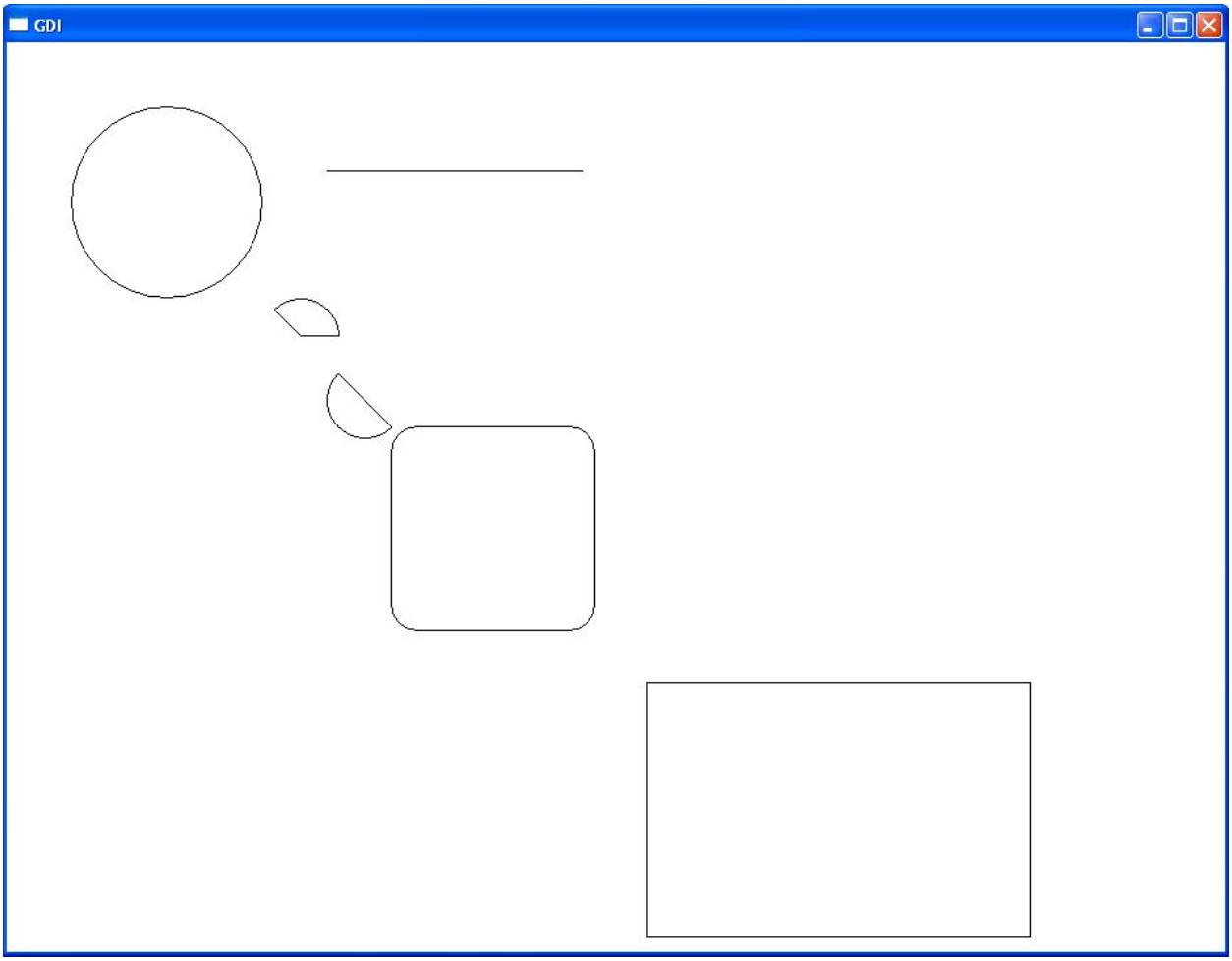
```
MSG msg;
// DECLARING AND INTIAZING THE WINDOW CLASS
WNDCLASS wc;                                    // Window class declaration
wc.style = CS_HREDRAW| CS_VREDRAW;              // Window style
wc. lpfnWndProc = WndProc;                      // Window Procedure
wc.cbClsExtra = 0;                              // Extra Parameters
wc.cbWndExtra = 0;                              // Extra Parameters
wc.hInstance = hInstance;                       // Instance handle
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION); // Icon
wc.hCursor = LoadCursor (NULL, IDC_ARROW);  // Window Cursor
wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
                                                // Set Background Color
wc.lpszMenuName = NULL;                         // Menu Handle
wc.lpszClassName = szAppName;                   // Class Name
// REGISTERING THE WINDOW CLASS
if (!RegisterClass (&wc))
{
        MessageBox(NULL, TEXT("This program requires WindowsNT!"),
szAppName, MB_ICONERROR);
        return 0;
}
// CREATING THE WINDOW
hwnd = CreateWindow ( szAppName,                // window class name
                      TEXT("GDI"),              // window caption
                      WS_OVERLAPPEDWINDOW,      // window style
                      CW_USEDEFAULT,            // initial x position
                      CW_USEDEFAULT,            // initial x position
```

```
                    CW_USEDEFAULT,          // initial x position
                    CW_USEDEFAULT,          // initial x position
                    NULL,                   // parent window handle
                    NULL,                   // window menu handle
                    hInstance,              //program instance handle
                    NULL);                  // creation parameters
// DISPLAYING THE WINDOW
        ShowWindow (hwnd, iCmdShow);
        UpdateWindow (hwnd);
// PROCESSING THE MESSAGE LOOP
        while (GetMessage(&msg,NULL,0,0))   // extracts message from Message Queue
        {
                TranslateMessage (&msg);
                DispatchMessage (&msg);     // dispatches the message received
        }
        return msg.wParam;
}

// WINDOW PROCEDURE FUNCTION
LRESULT CALLBACK WndProc (HWND hwnd, UINT wm, WPARAM wp, LPARAM lp)
{

        switch (wm)
        {
        case WM_PAINT:
                hdc=BeginPaint(hwnd,&ps);
                Ellipse(hdc,50,50,200,200);
                break;
```

```
        case WM_LBUTTONDOWN:
                MoveToEx(hdc,250,100,NULL);
                LineTo(hdc,450,100);
                break;

        case WM_NCLBUTTONDOWN:
                Pie (hdc, 200, 200, 260, 260, 230, 230, 225, 225);
                Chord (hdc, 250, 250, 310, 310, 270, 270, 300, 300);
                break;

        case WM_RBUTTONDOWN:
                Rectangle(hdc, 500,500,800,700);
                break;

        case WM_NCRBUTTONDOWN:
                RoundRect(hdc, 300,300,460,460,40,40);
                break;

        case WM_DESTROY:
                EndPaint(hwnd,&ps);
                PostQuitMessage(0);
                return 0;
        }
        return DefWindowProc(hwnd,wm,wp,lp);
}
```

SAMPLE OUTPUT:

- If you move the mouse cursor inside client area, a Circle will be drawn on the client area.
- Bring the mouse cursor in to the client area, and then press left mouse button, Line will be drawn on the client area. Now press Right mouse button, a Rectangle will be appear.
- Bring the mouse cursor to the title bar (i.e. Non client area) and then press left mouse button, a Chord and Pie will be drawn on the client area. Now press Right mouse button, a Rounded rectangle will be appear.



DESCRIPTION:

LINE:

To draw a straight line, you must call two functions. The first function specifies the point at which the line begins, and the second function specifies the end point of the line:

```
MoveToEx (hdc, xBeg, yBeg, NULL);
LineTo (hdc, xEnd, yEnd) ;
```

RECTANGLE:

```
Rectangle (hdc, xLeft, yTop, xRight, yBottom) ;
```

ELLIPSE:

Once you know how to draw a rectangle, you also know how to draw an ellipse, because it uses the same arguments:

```
Ellipse (hdc, xLeft, yTop, xRight, yBottom) ;
```

ROUNDRECT:

The function to draw rectangles with rounded corners uses the same bounding box as the *Rectangle* and *Ellipse* functions but includes two more arguments:

```
RoundRect (hdc, xLeft, yTop, xRight, yBottom,
        xCornerEllipse, yCornerEllipse) ;

xCornerEllipse = (xRight - xLeft) / 4 ;
yCornerEllipse = (yBottom- yTop) / 4 ;
```

CHORD and PIE:

The *Chord*, and *Pie* functions all take identical arguments:

```
Chord (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd);
Pie   (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd);
```

Windows first draws an arc in an elliptical structure and the arc is an elliptical line rather than a filled area. For the *Chord* function, Windows connects the endpoints of the arc. For the *Pie* function, Windows connects each endpoint of the arc with the center of the ellipse. The interiors of the chord and pie-wedge figures are filled with the current brush.

Result:

Thus the program for GDI functions to display different GDI objects has been developed, builds and executed using Win32 application.

6. DIALOG BASED APPLICATION

Aim:

To develop a dialog based application using MFC Appwizard in VC++.

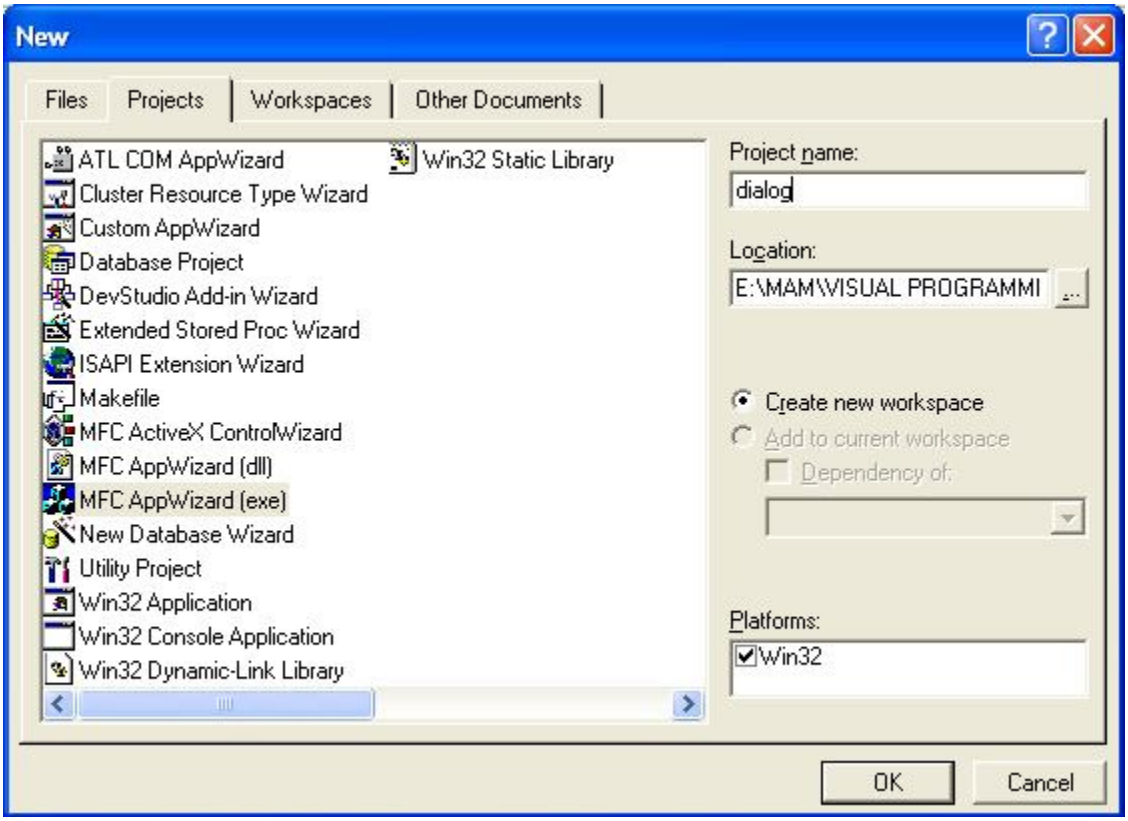
Concept:

For many applications, a dialog provides a sufficient user interface. The dialog window immediately appears when the user starts the application. The user can minimize and maximize the dialog window, as long as the dialog is not system modal, the user can freely switch to other application.

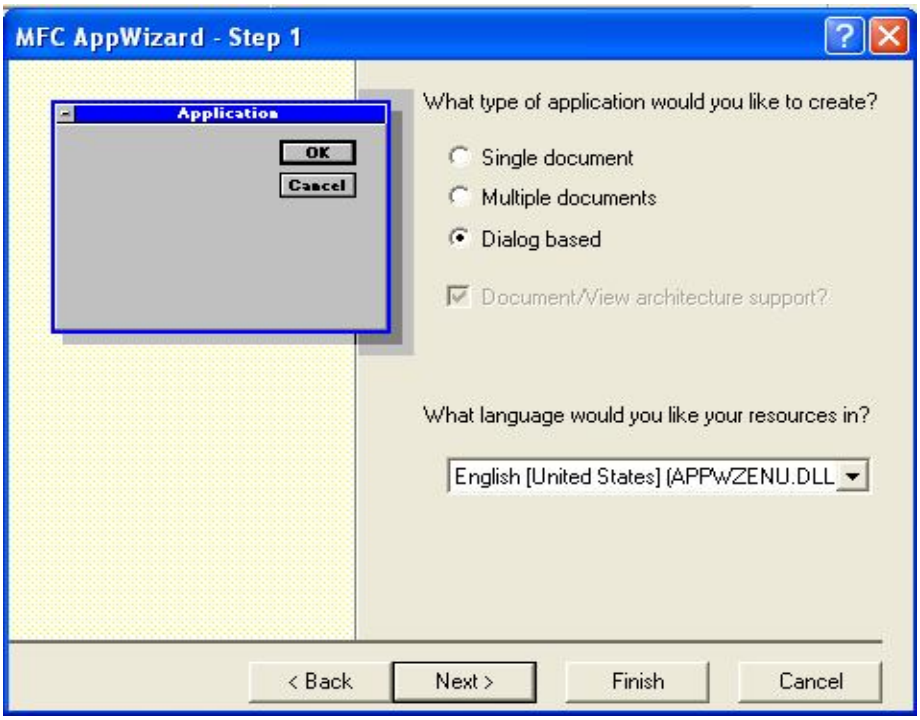
This program is to do arithmetic operation such as addition, subtraction, multiplication and division. User has to enter the 2 values and click the corresponding button to perform the operation which the user needs. The result will be displayed on the third text box.For division operation, if the second operand is zero, then Divide by Zero error will be indicated in a separate dialog box.

Procedure:

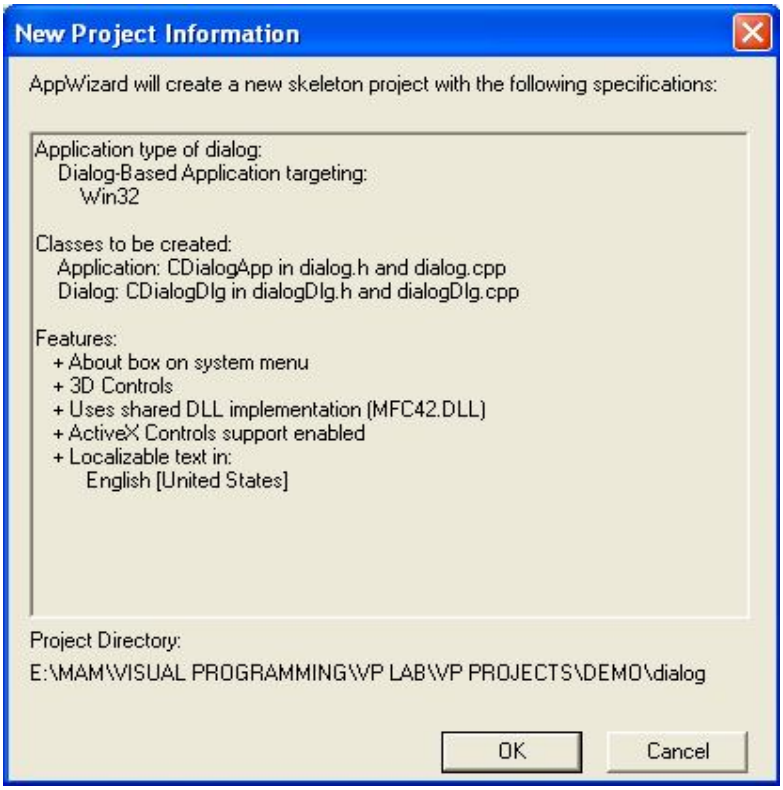
Step 1:



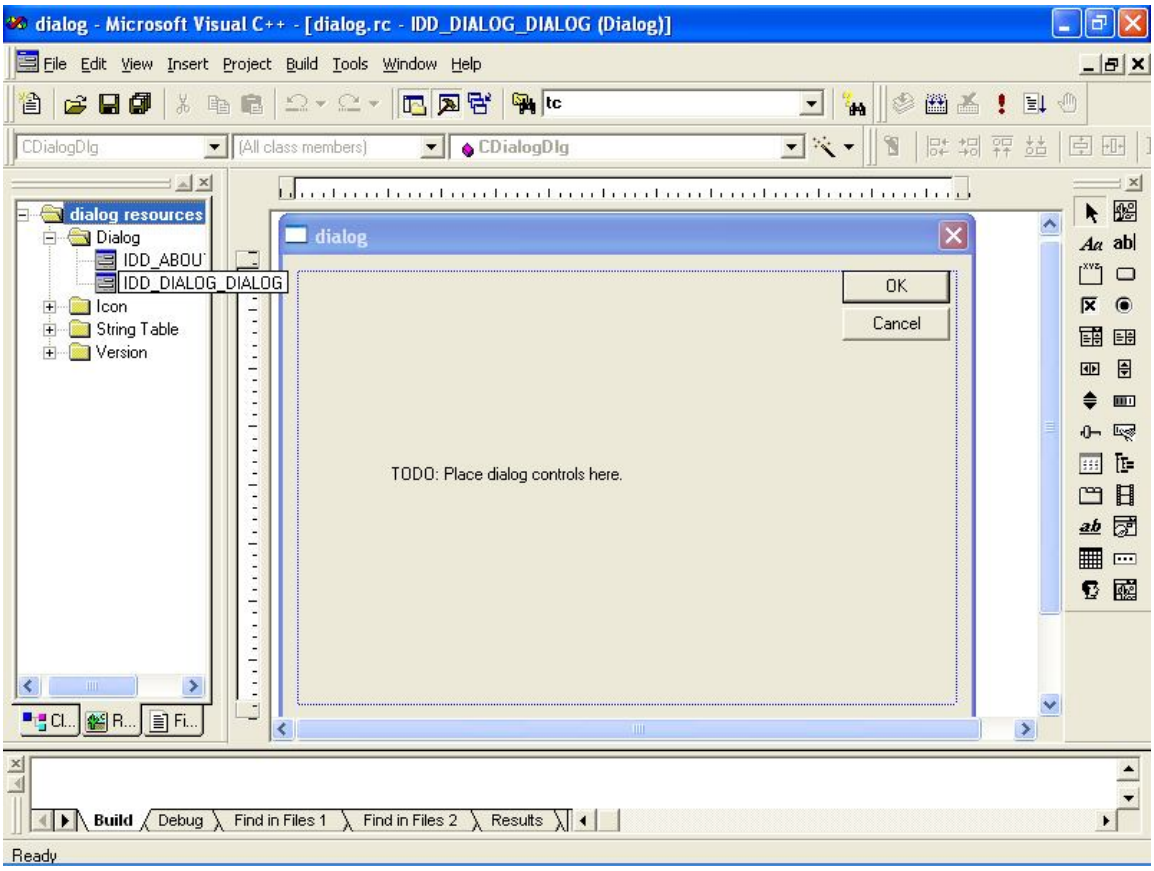
Step 2:


MFC AppWizard - Step 1
Application
OK
Cancel
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

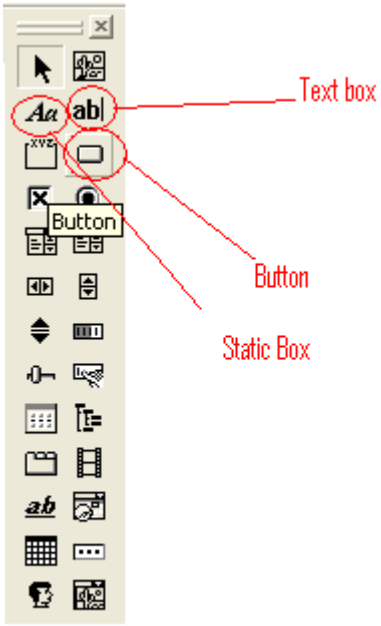
Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of dialog:
Dialog-Based Application targeting:
Win32
Classes to be created:
Application: CDialogApp in dialog.h and dialog.cpp
Dialog: CDialogDlg in dialogDlg.h and dialogDlg.cpp
Features:
+ About box on system menu
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\MAM\VISUAL PROGRAMMING\VP LAB\VP PROJECTS\DEMO\dialog
OK
Cancel

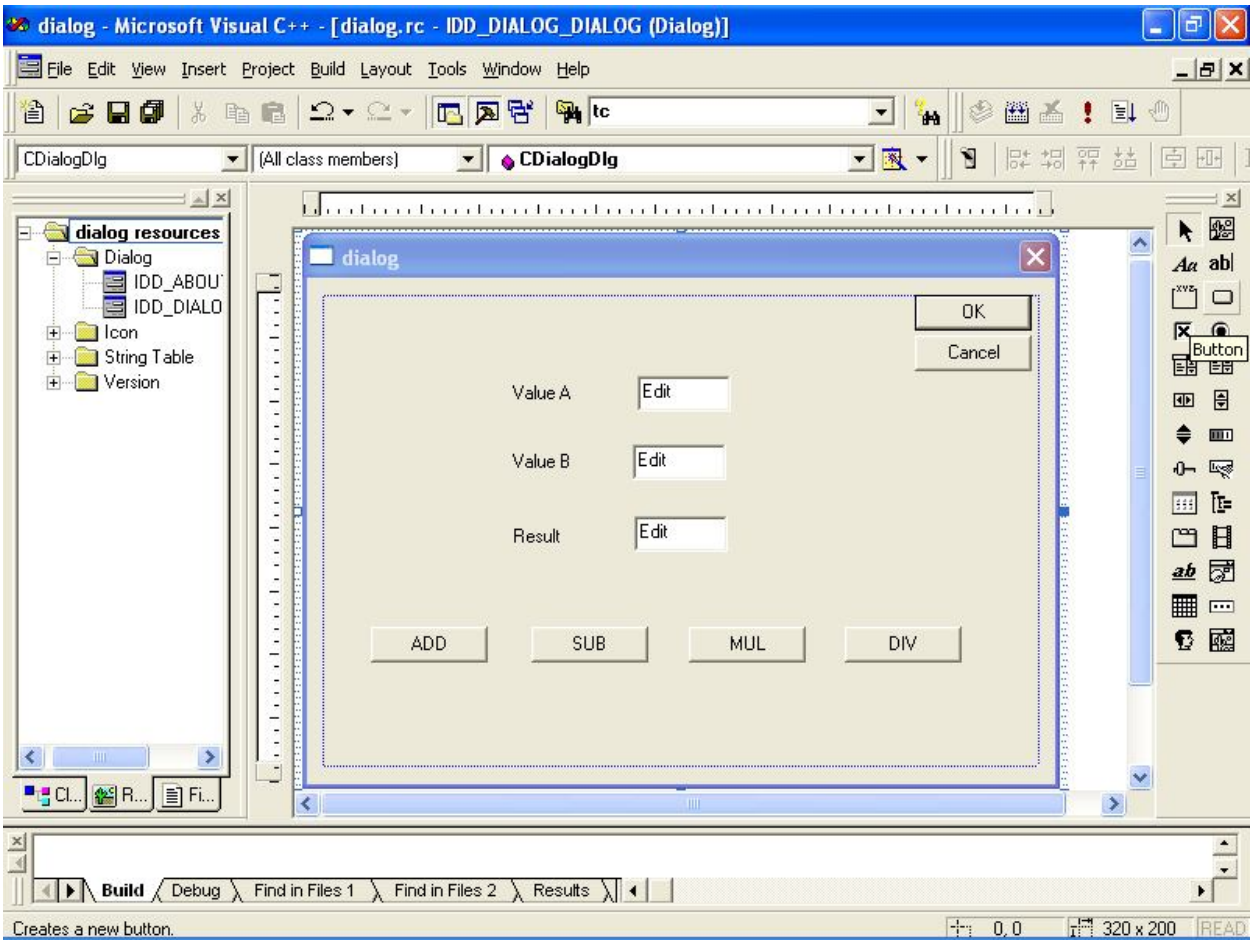
Step 4:


dialog - Microsoft Visual C++ - [dialog.rc - IDD_DIALOG_DIALOG (Dialog)]
File Edit View Insert Project Build Tools Window Help
dialog resources
Dialog
IDD_ABOU
IDD_DIALOG_DIALOG
Icon
String Table
Version
dialog
OK
Cancel
TODO: Place dialog controls here.
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

Step 5:


Text box
Button
Static Box
Button

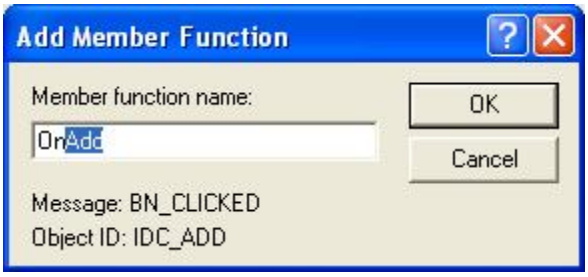
Step 6:


dialog - Microsoft Visual C++ - [dialog.rc - IDD_DIALOG_DIALOG (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
CDialogDlg
(All class members)
CDialogDlg
dialog resources
Dialog
Icon
String Table
Version
dialog
OK
Cancel
Value A
Edit
Value B
Edit
Result
Edit
ADD
SUB
MUL
DIV
Button
Build
Debug
Find in Files 1
Find in Files 2
Results
Creates a new button.
0, 0
320 x 200
READ

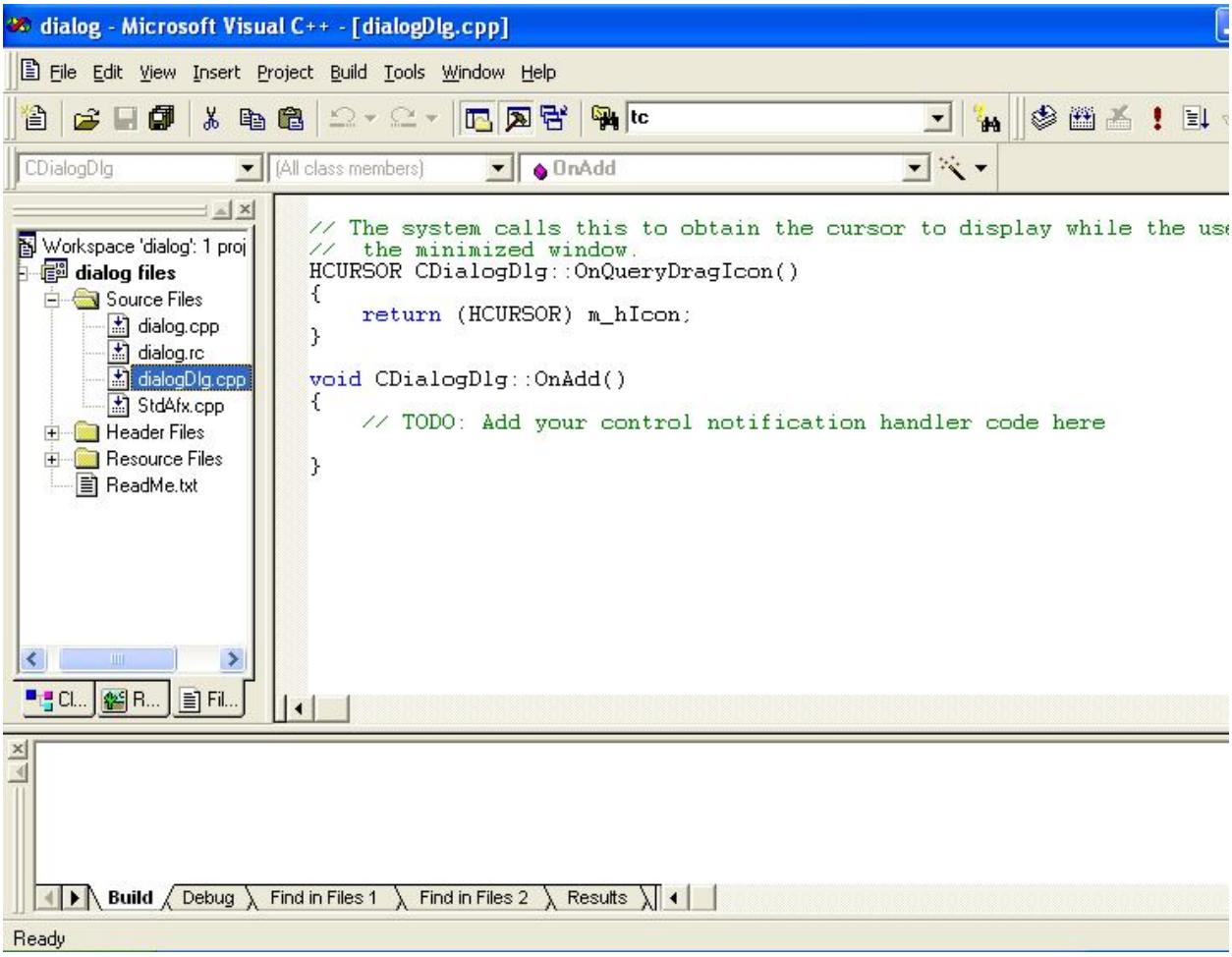
Step 7:


Push Button Properties
General
Styles
Extended Styles
ID:
IDC_ADD
Caption:
ADD
Visible
Group
Help ID
Disabled
Tab stop

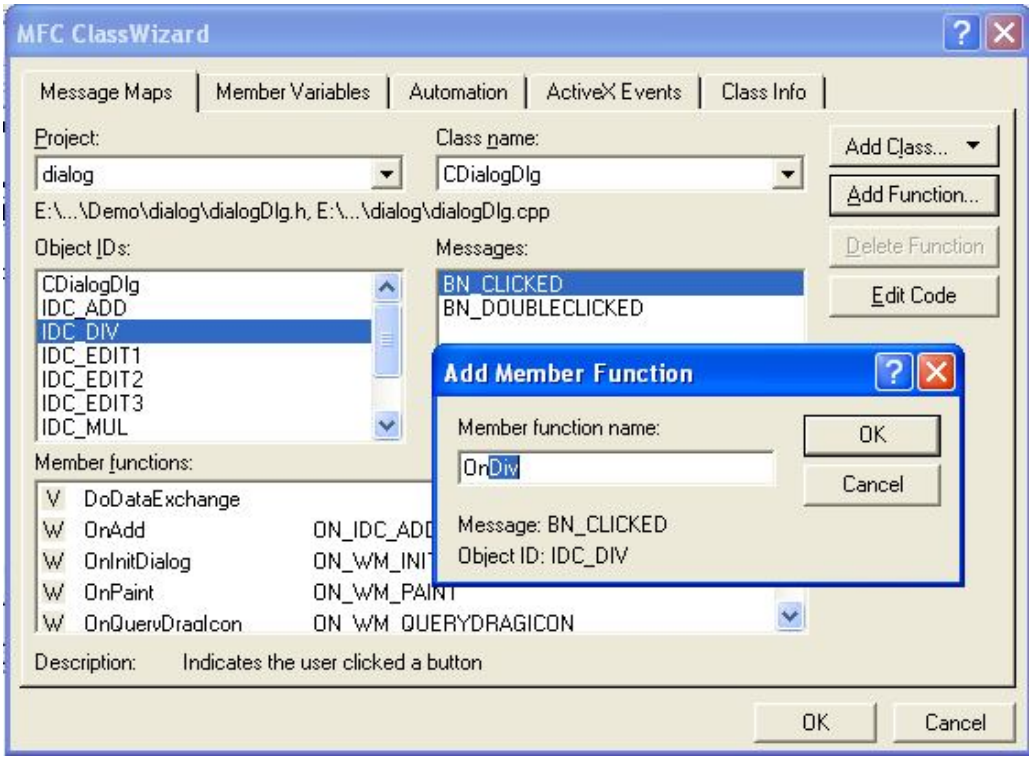
Step 8:


Add Member Function
Member function name:
OnAdd
OK
Cancel
Message: BN_CLICKED
Object ID: IDC_ADD

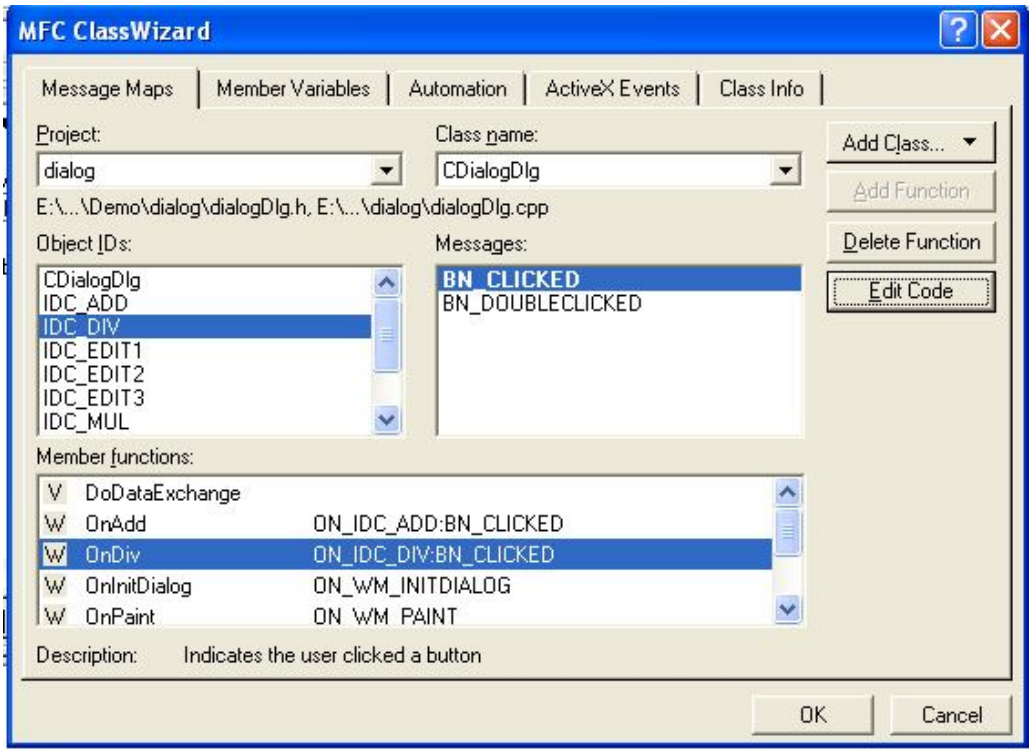
Step 9:


dialog - Microsoft Visual C++ - [dialogDlg.cpp]
File Edit View Insert Project Build Tools Window Help
CDialogDlg
(All class members)
OnAdd
Workspace 'dialog': 1 proj
dialog files
Source Files
dialog.cpp
dialog.rc
dialogDlg.cpp
StdAfx.cpp
Header Files
Resource Files
ReadMe.txt
// The system calls this to obtain the cursor to display while the use
// the minimized window.
HCURSOR CDialogDlg::OnQueryDragIcon()
{
return (HCURSOR) m_hIcon;
}
void CDialogDlg::OnAdd()
{
// TODO: Add your control notification handler code here
}
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

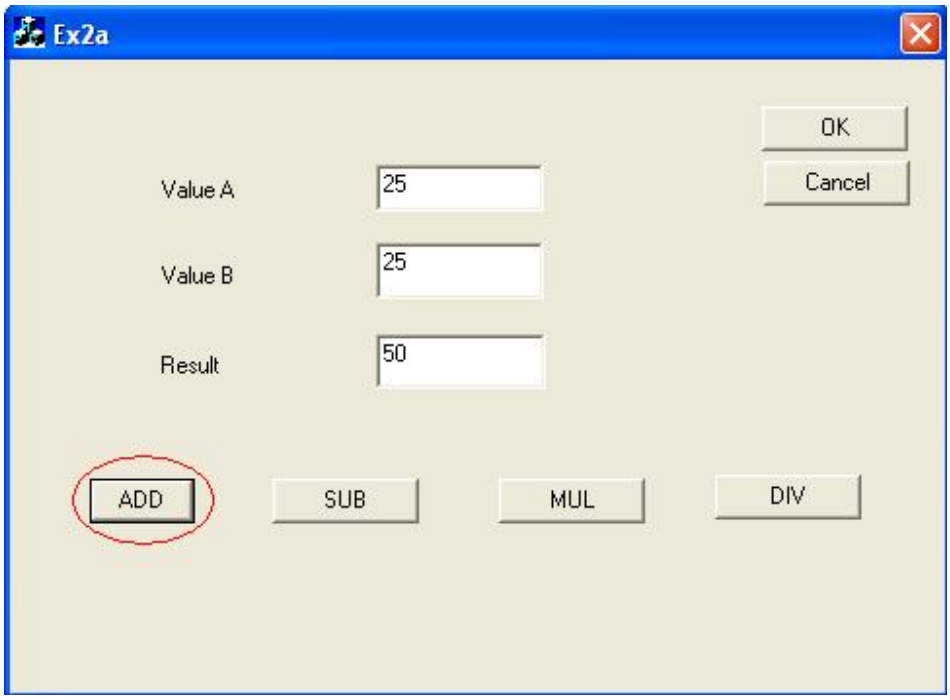
Step 10:



Step 11:



Output:



Description:

Result:

Thus the VC++ program to handle arithmetic operation using MFC dialog based application has been developed, builds and verified

7. Creating MDI application

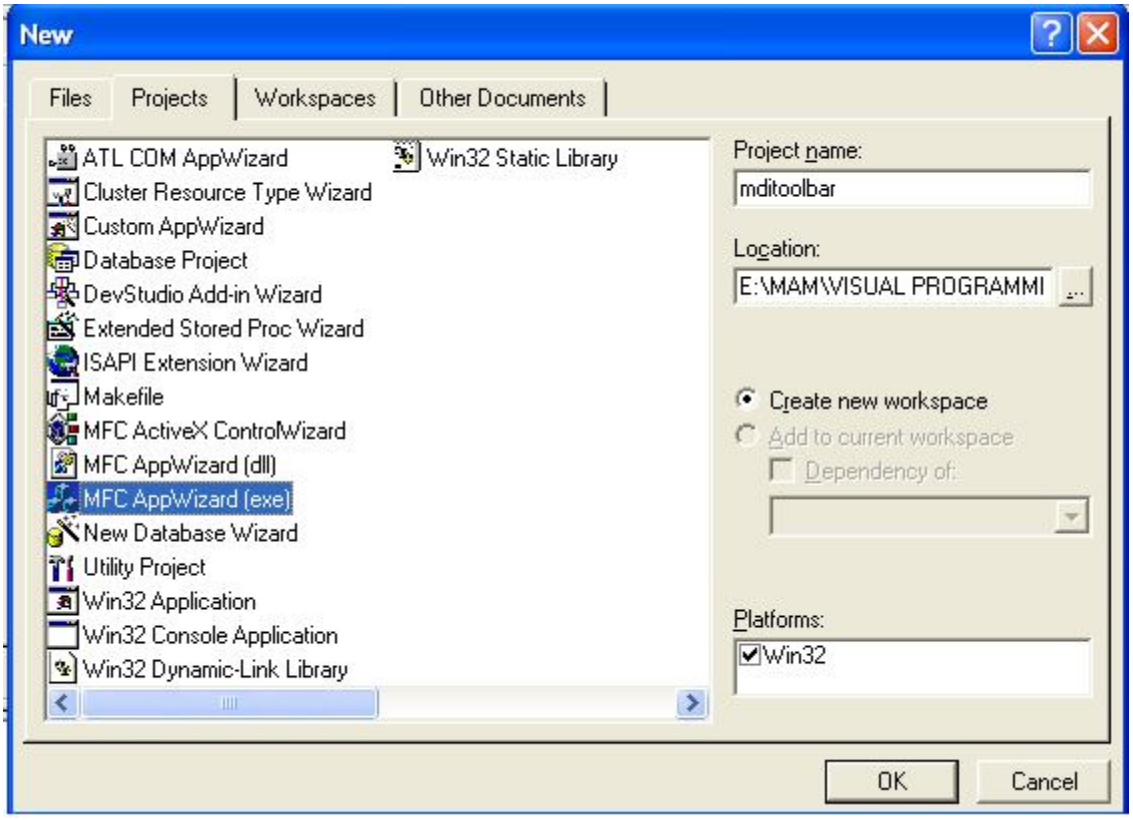
Aim:

To develop a MDI application using MFC appwizard in VC++

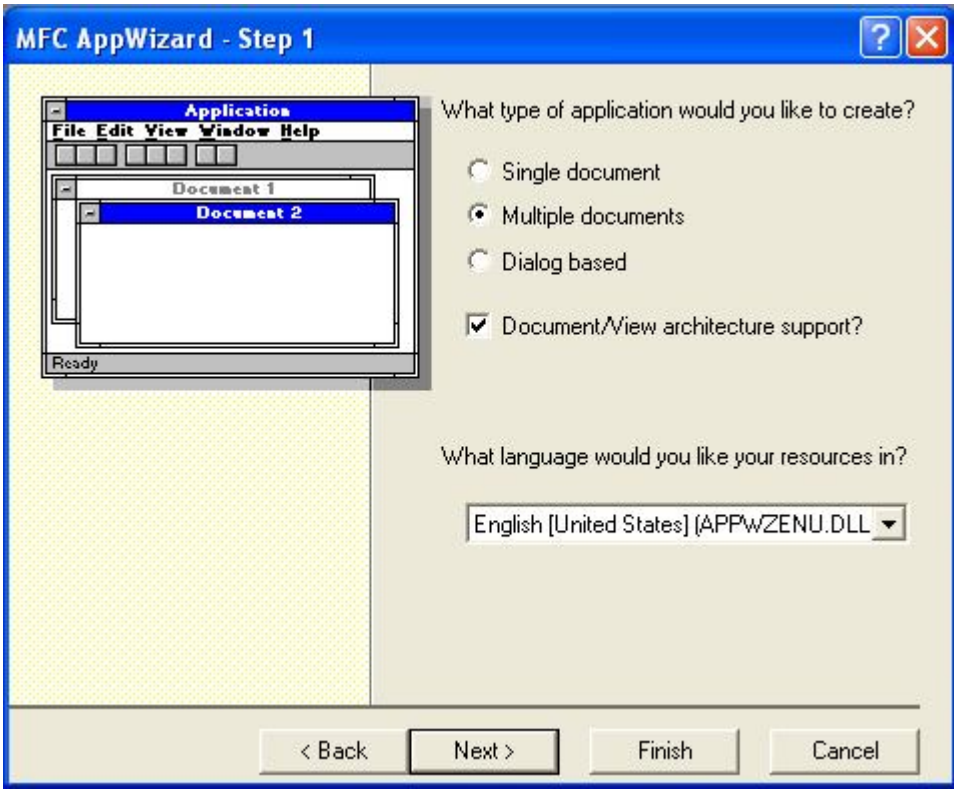
Concept:

Procedure:

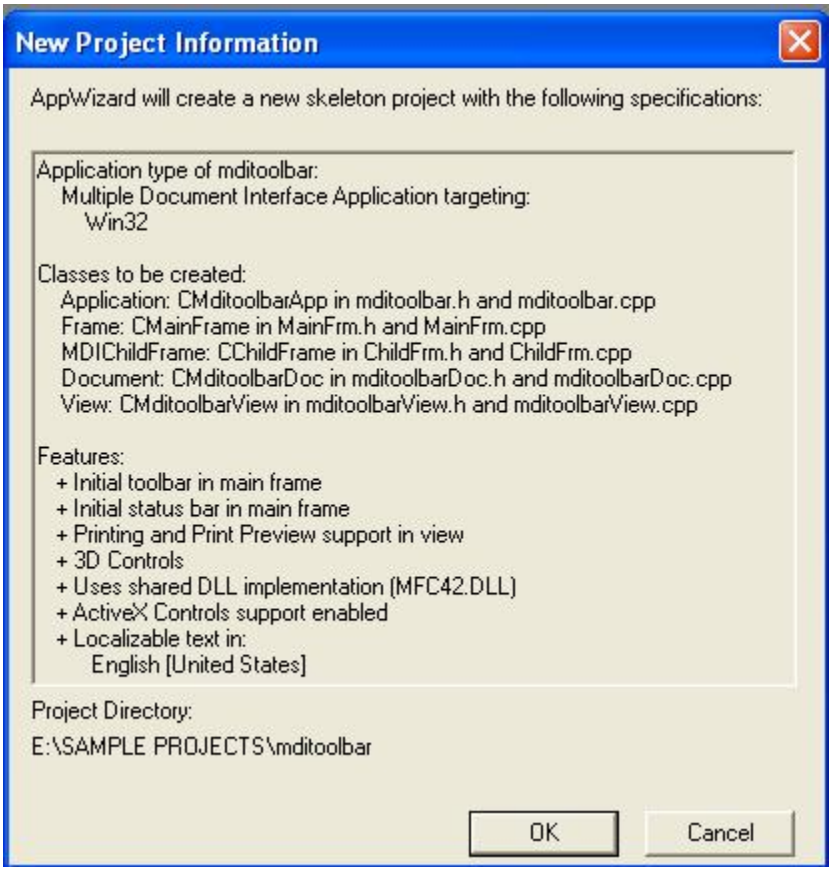
Step 1:



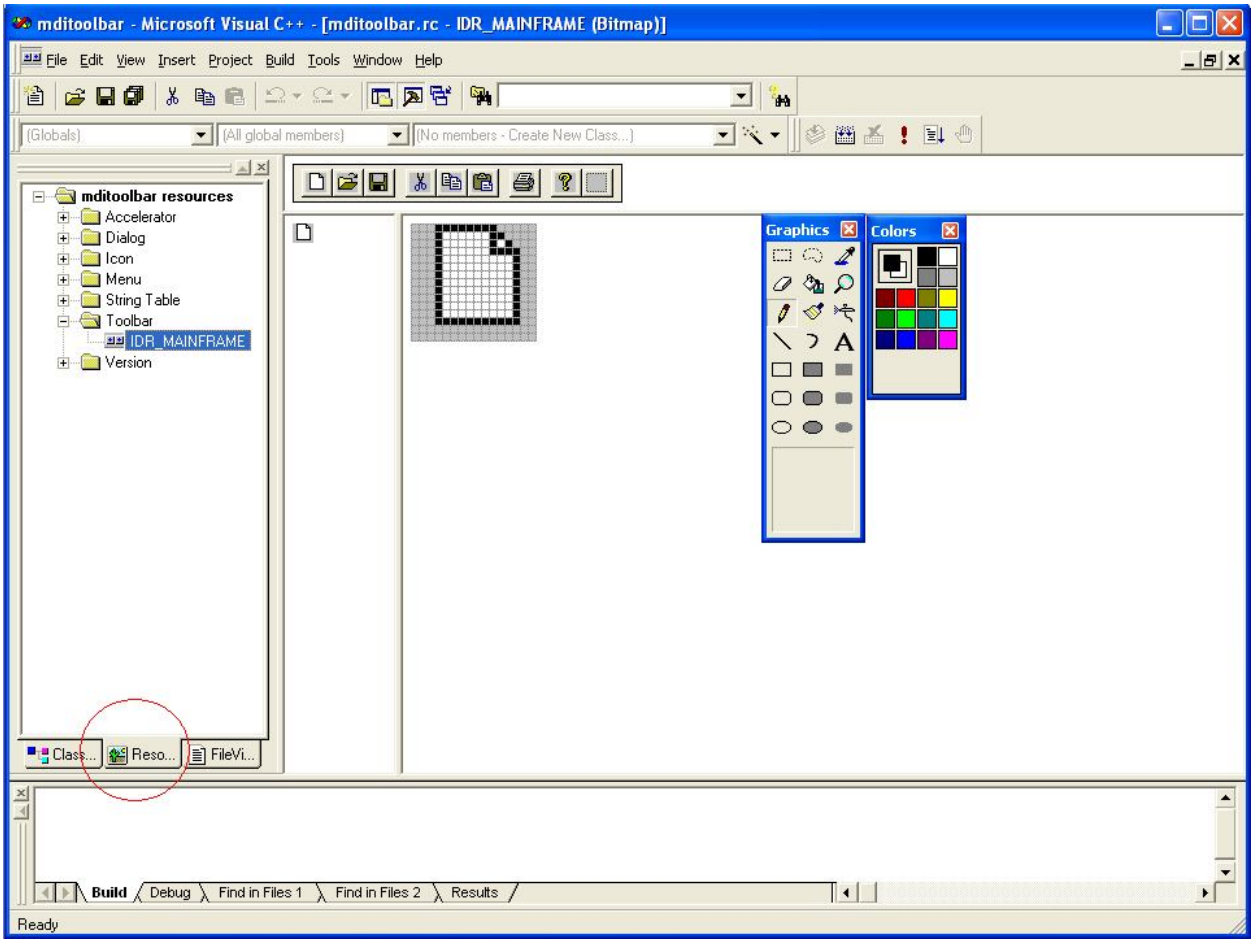
Step 2:


MFC AppWizard - Step 1
Application
File Edit View Window Help
Document 1
Document 2
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

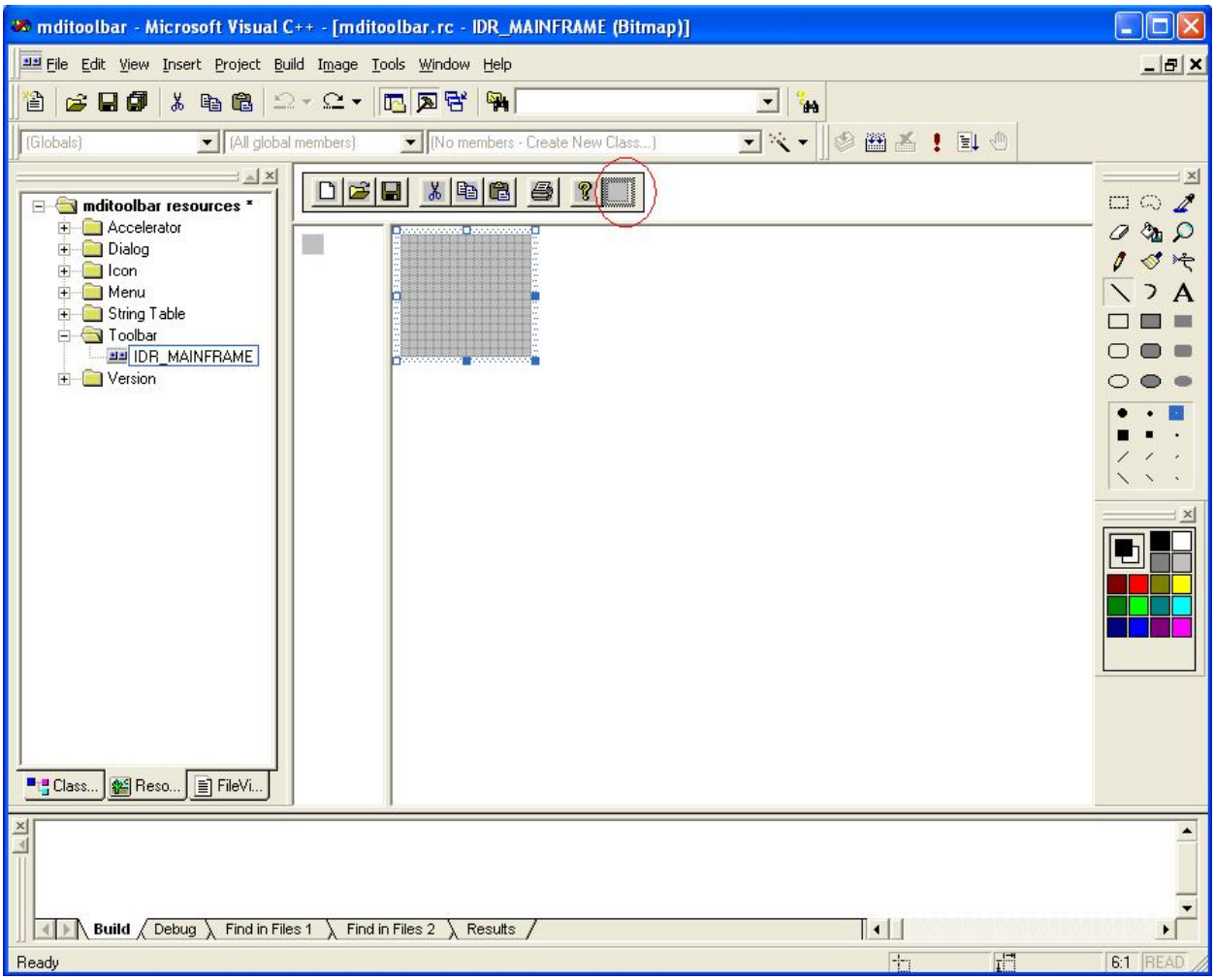
Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of mditoolbar:
Multiple Document Interface Application targeting:
Win32
Classes to be created:
Application: CMditoolbarApp in mditoolbar.h and mditoolbar.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
Document: CMditoolbarDoc in mditoolbarDoc.h and mditoolbarDoc.cpp
View: CMditoolbarView in mditoolbarView.h and mditoolbarView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\SAMPLE PROJECTS\mditoolbar
OK
Cancel

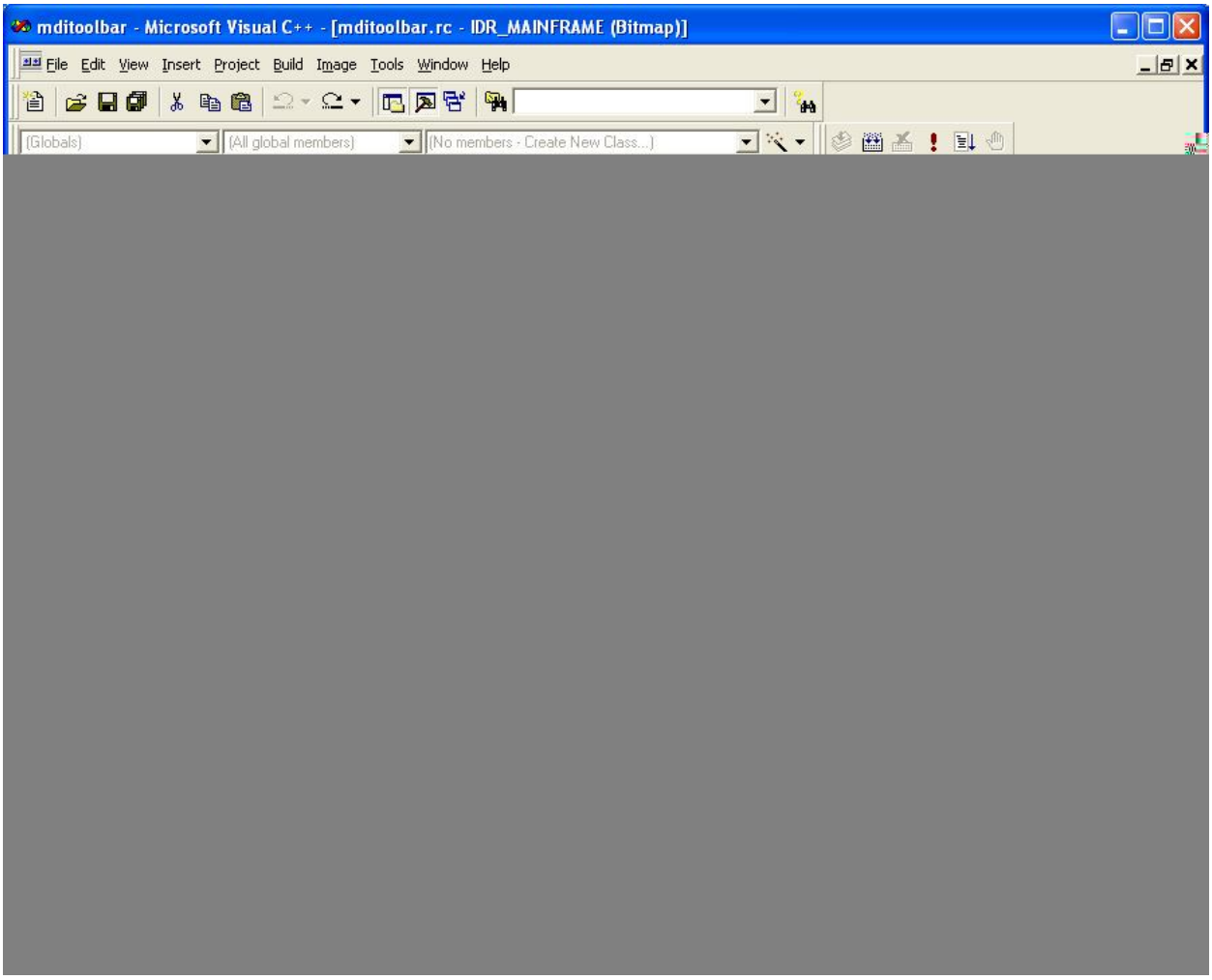
Step 4:


mditoolbar - Microsoft Visual C++ - [mditoolbar.rc - IDR_MAINFRAME (Bitmap)]
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
(No members - Create New Class...)
mditoolbar resources
Accelerator
Dialog
Icon
Menu
String Table
Toolbar
IDR_MAINFRAME
Version
Class...
Reso...
FileVi...
Graphics
Colors
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

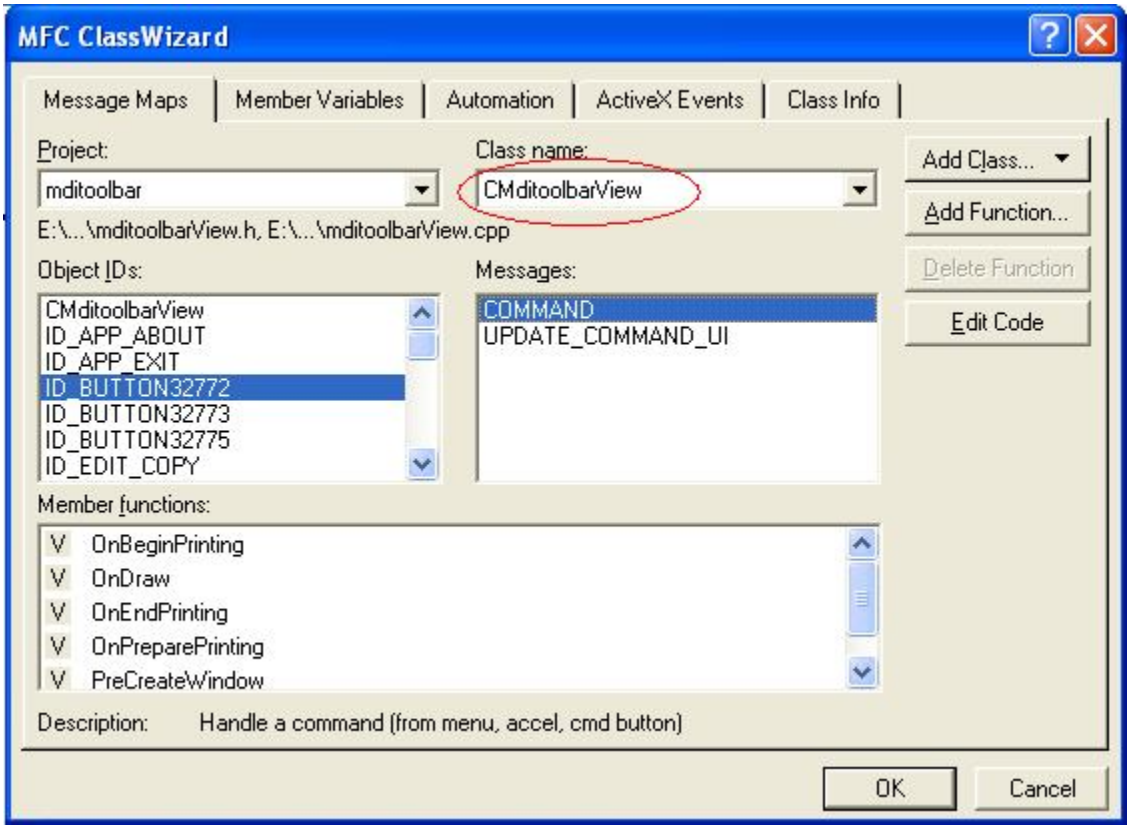
Step 5:


mditoolbar - Microsoft Visual C++ - [mditoolbar.rc - IDR_MAINFRAME (Bitmap)]
File Edit View Insert Project Build Image Tools Window Help
(Globals)
(All global members)
(No members - Create New Class...)
mditoolbar resources *
Accelerator
Dialog
Icon
Menu
String Table
Toolbar
IDR_MAINFRAME
Version
Class...
Reso...
FileVi...
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
6:1
READ

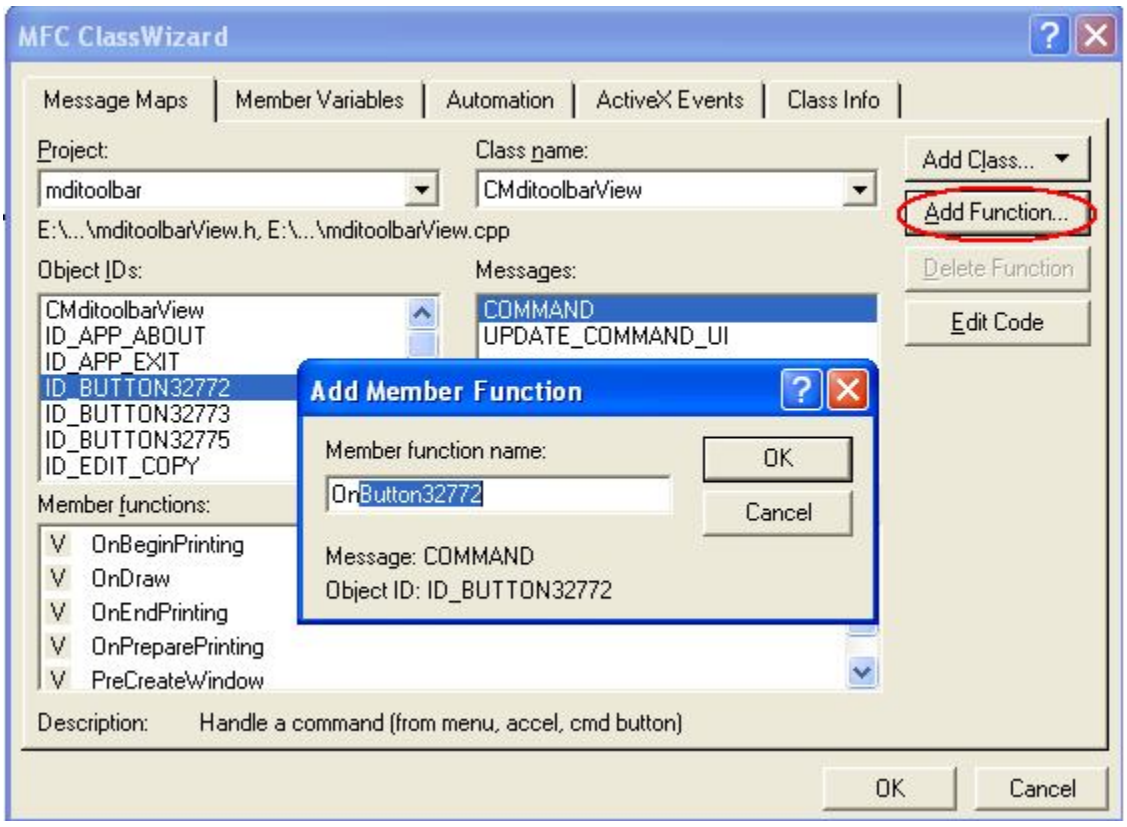
Step 6:


mditoolbar - Microsoft Visual C++ - [mditoolbar.rc - IDR_MAINFRAME (Bitmap)]
File Edit View Insert Project Build Image Tools Window Help
(Globals)
(All global members)
(No members - Create New Class...)

Step 7:

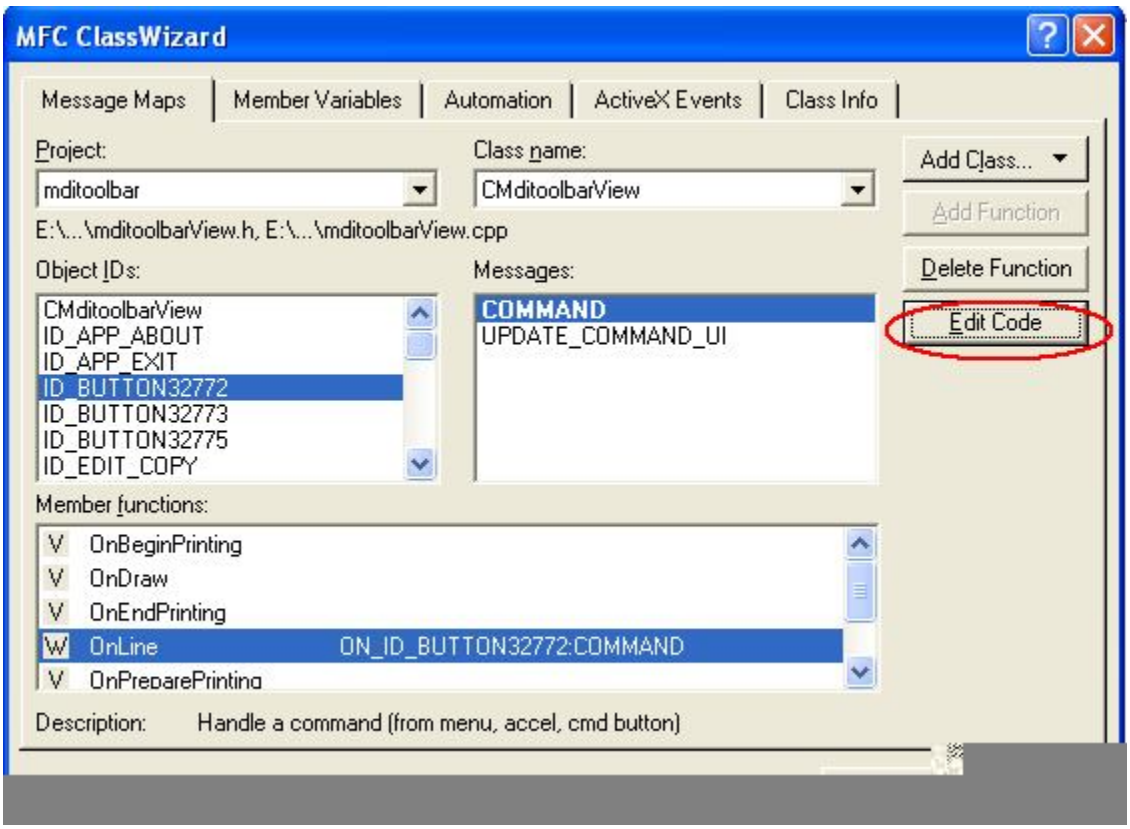

MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
mditoolbar
Class name:
CMditoolbarView
Add Class...
Add Function...
Delete Function
Edit Code
E:\...\mditoolbarView.h, E:\...\mditoolbarView.cpp
Object IDs:
CMditoolbarView
ID_APP_ABOUT
ID_APP_EXIT
ID_BUTTON32772
ID_BUTTON32773
ID_BUTTON32775
ID_EDIT_COPY
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
V OnPreparePrinting
V PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

Step 8:

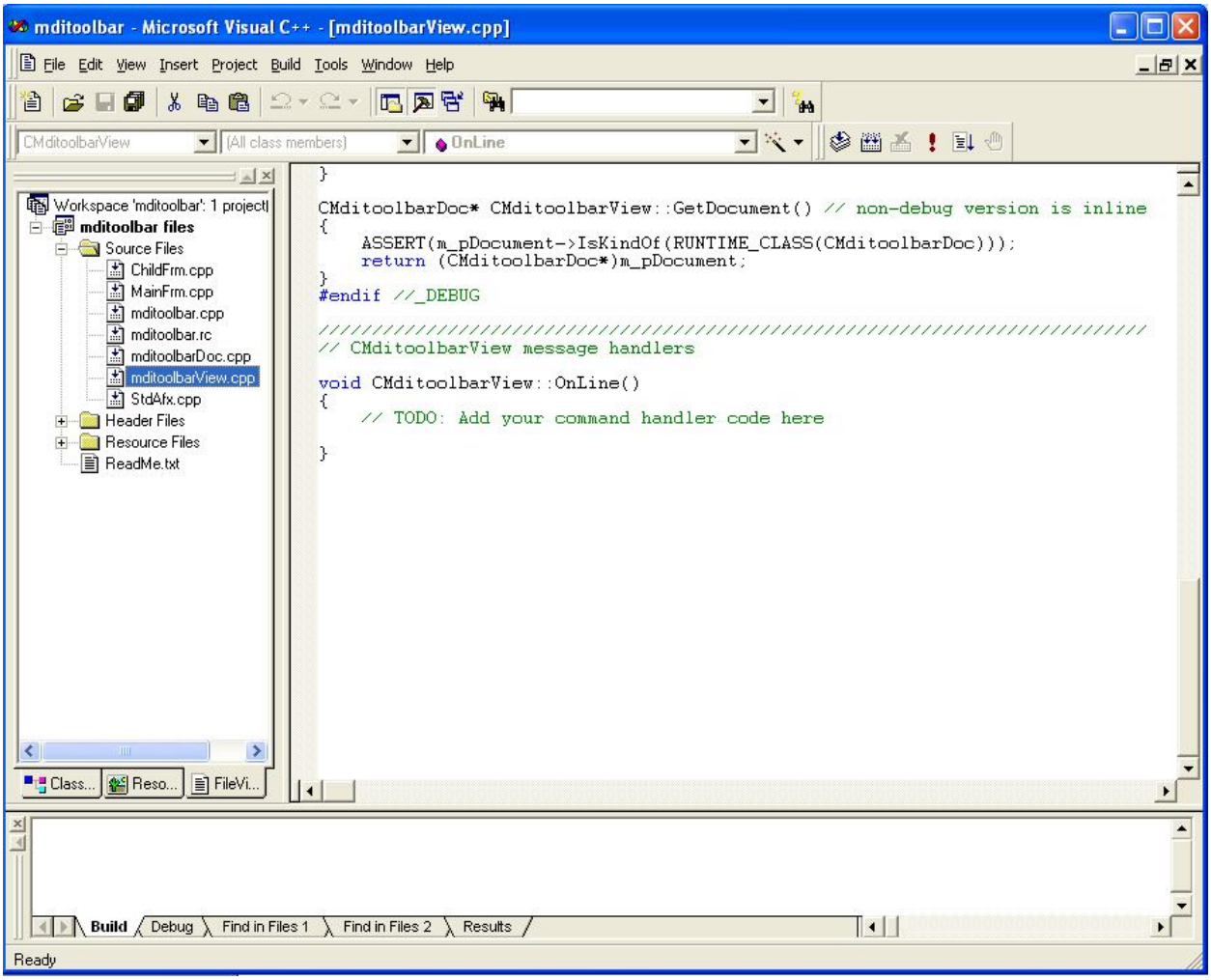


Step 9:

Step 10:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
mditoolbar
Class name:
CMditoolbarView
Add Class...
Add Function
Delete Function
Edit Code
E:\...\mditoolbarView.h, E:\...\mditoolbarView.cpp
Object IDs:
CMditoolbarView
ID_APP_ABOUT
ID_APP_EXIT
ID_BUTTON32772
ID_BUTTON32773
ID_BUTTON32775
ID_EDIT_COPY
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
W OnLine ON_ID_BUTTON32772:COMMAND
V OnPreparePrinting
Description: Handle a command (from menu, accel, cmd button)

Step 11:



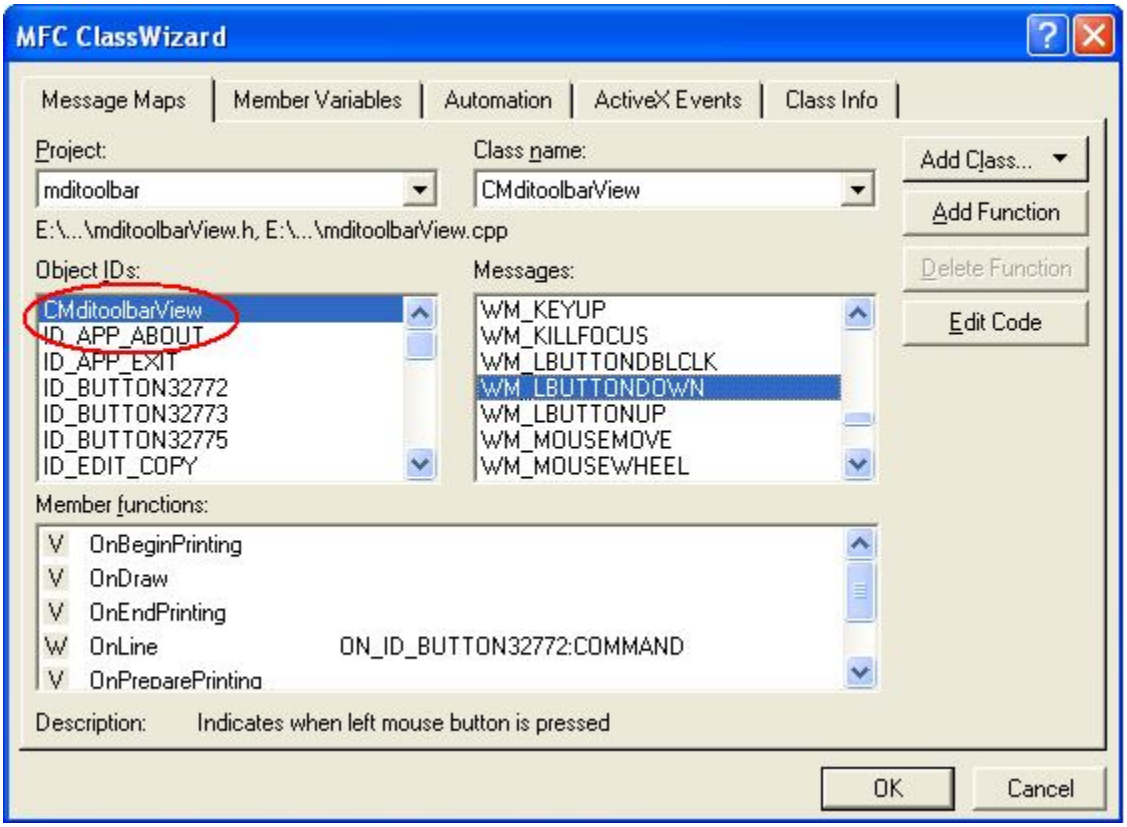
Step 12:

void CMditoolbarView::OnLine() {

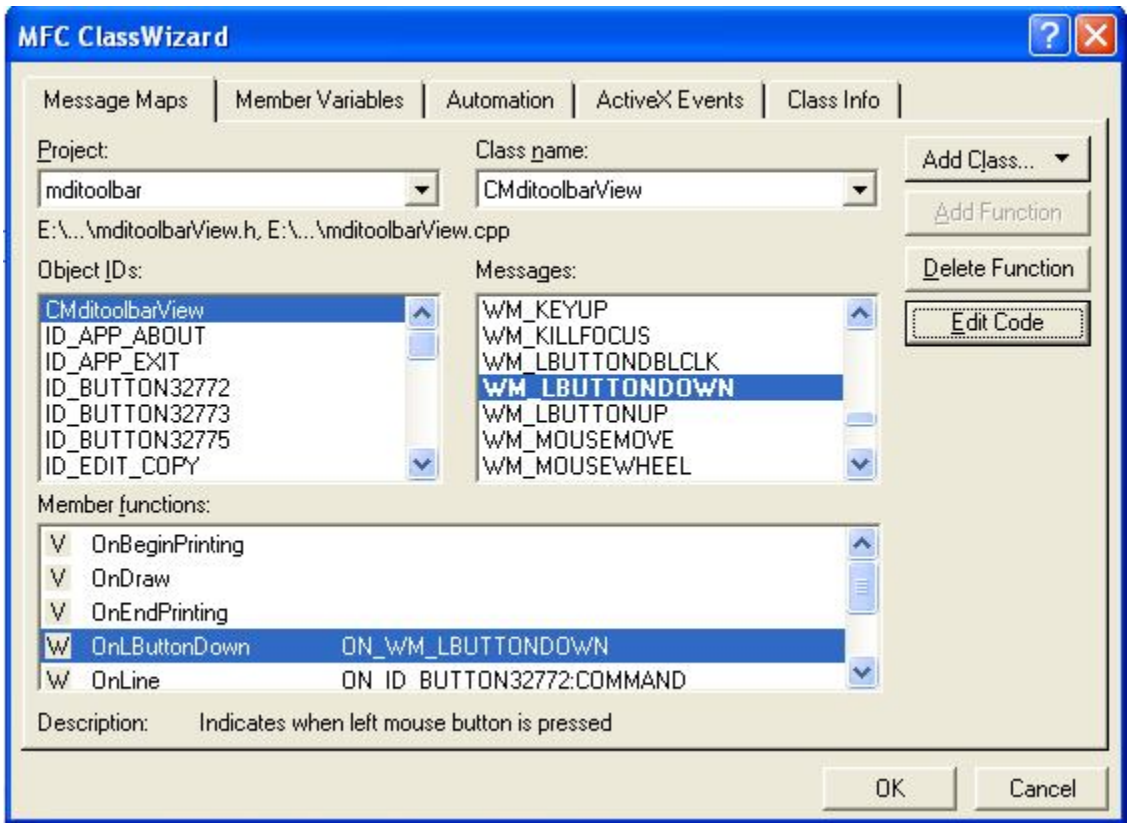
s=2;

}

Step 13:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
mditoolbar
Class name:
CMditoolbarView
Add Class...
Add Function
Delete Function
Edit Code
E:\...\mditoolbarView.h, E:\...\mditoolbarView.cpp
Object IDs:
CMditoolbarView
ID_APP_ABOUT
ID_APP_EXIT
ID_BUTTON32772
ID_BUTTON32773
ID_BUTTON32775
ID_EDIT_COPY
Messages:
WM_KEYUP
WM_KILLFOCUS
WM_LBUTTONDBLCLK
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_MOUSEMOVE
WM_MOUSEWHEEL
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
W OnLine ON_ID_BUTTON32772:COMMAND
V OnPreparePrinting
Description: Indicates when left mouse button is pressed
OK
Cancel

Step 14:

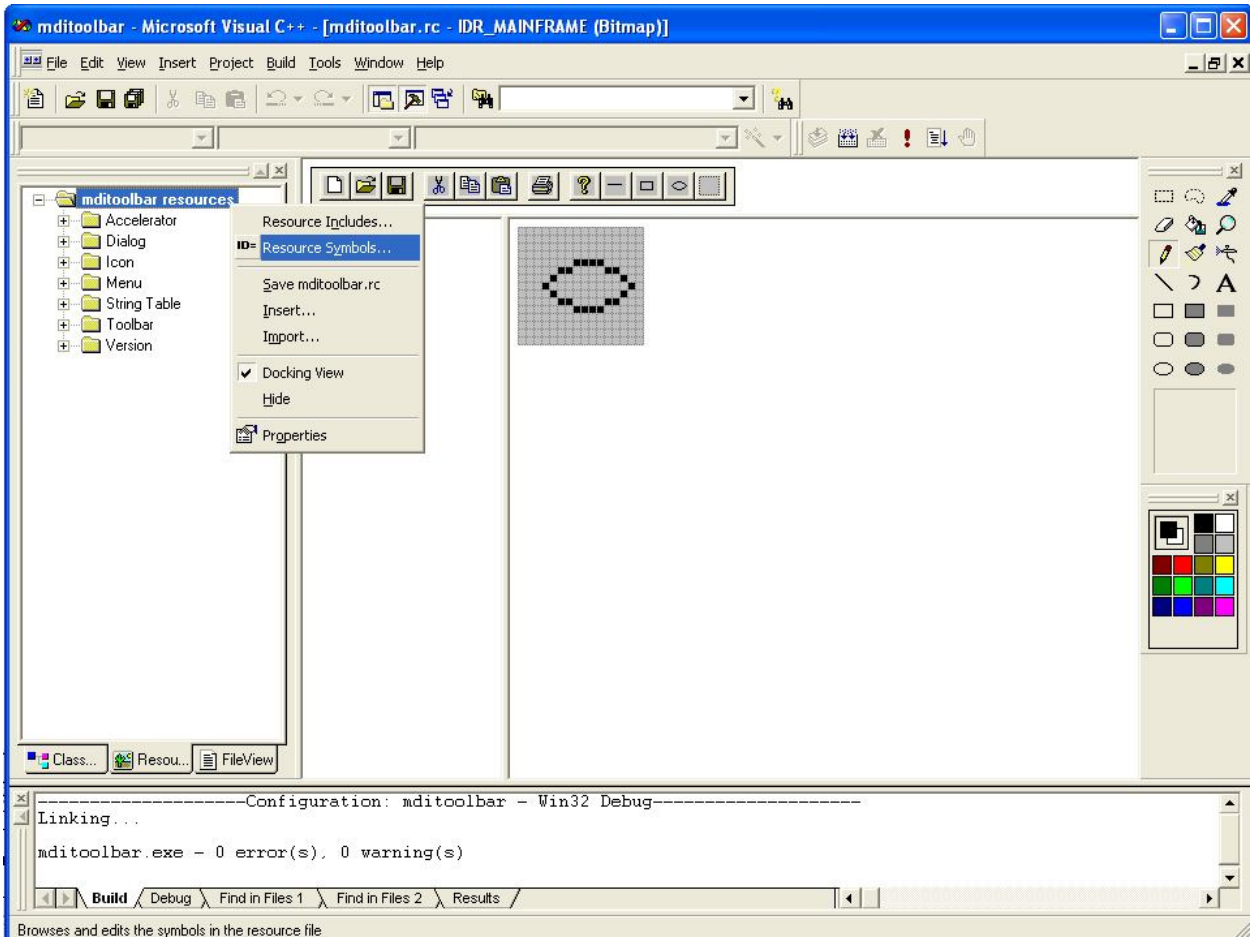


Step 15:

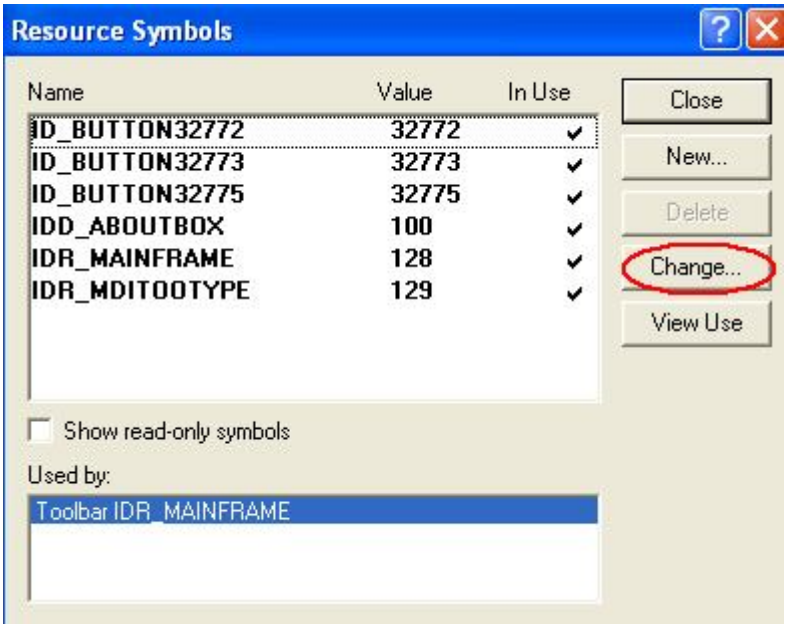
```
void CMditoolbarView::OnLButtonDown(UINT nFlags, CPoint point) {
        st=point;
        CView::OnLButtonDown(nFlags, point);
}
```

Creating Tooltip:

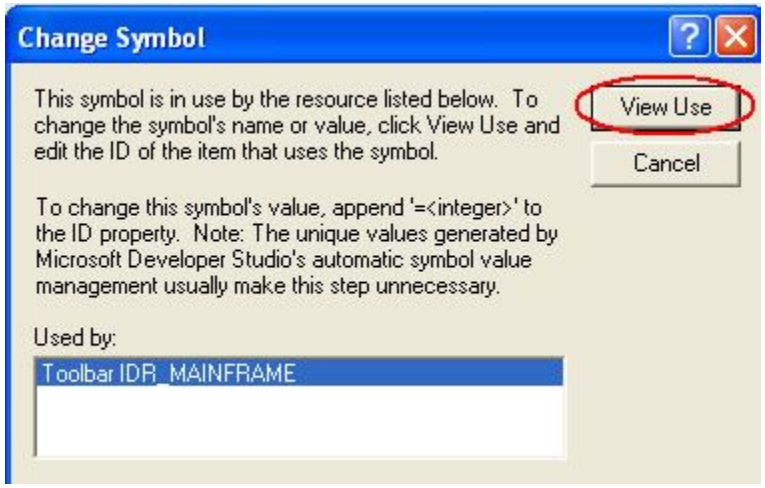
Step 16:


mditoolbar - Microsoft Visual C++ - [mditoolbar.rc - IDR_MAINFRAME (Bitmap)]
File Edit View Insert Project Build Tools Window Help
mditoolbar resources
Accelerator
Dialog
Icon
Menu
String Table
Toolbar
Version
Resource Includes...
Resource Symbols...
Save mditoolbar.rc
Insert...
Import...
Docking View
Hide
Properties
Class...
Resou...
FileView
Configuration: mditoolbar - Win32 Debug
Linking...
mditoolbar.exe - 0 error(s), 0 warning(s)
Build
Debug
Find in Files 1
Find in Files 2
Results
Browses and edits the symbols in the resource file

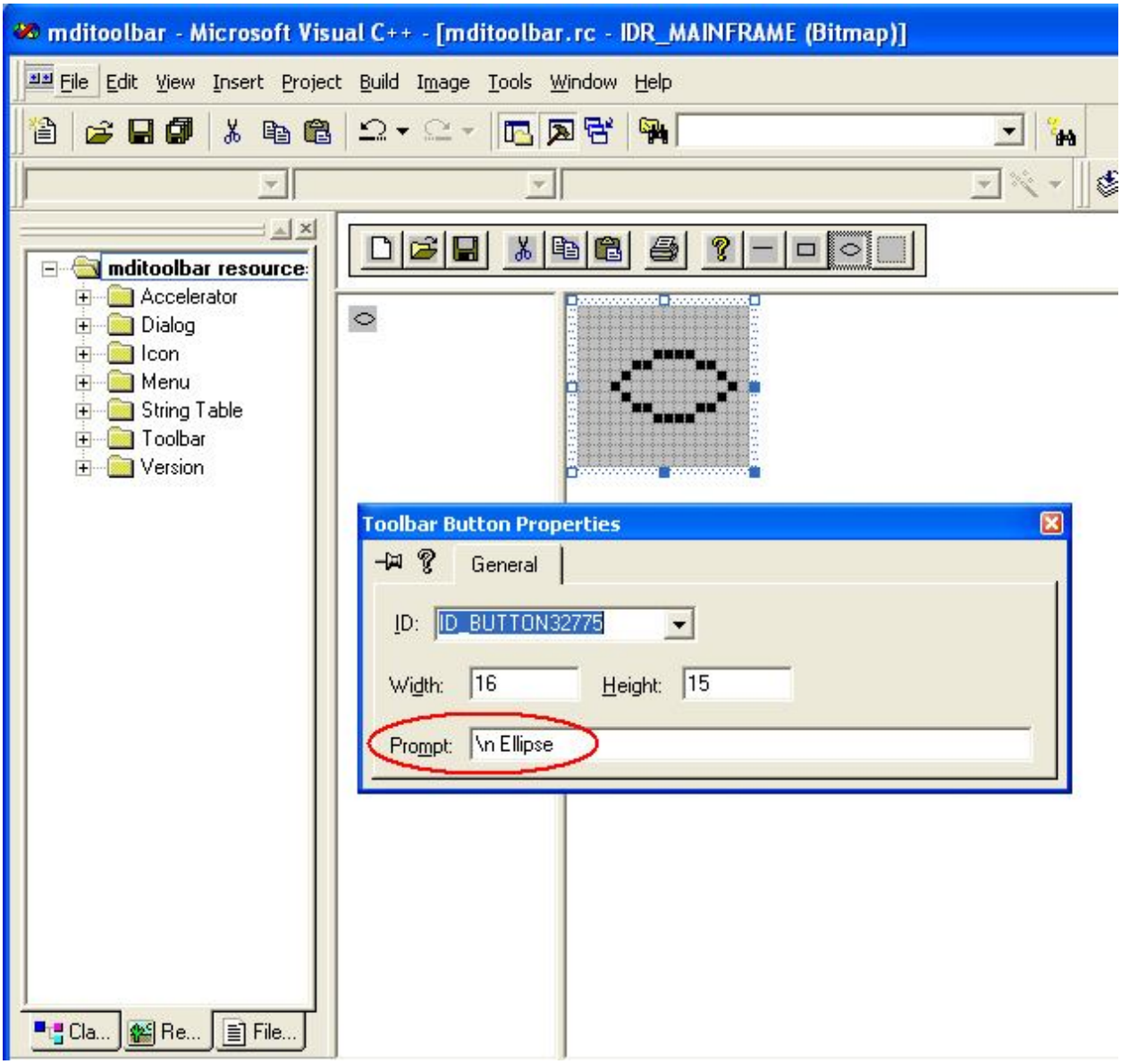
Step 17:


Resource Symbols
Name
Value
In Use
ID_BUTTON32772 32772
ID_BUTTON32773 32773
ID_BUTTON32775 32775
IDD_ABOUTBOX 100
IDR_MAINFRAME 128
IDR_MDITOOTYPE 129
Close
New...
Delete
Change...
View Use
Show read-only symbols
Used by:
Toolbar IDR_MAINFRAME

Step 18:



Step 19:



Program:

// View file: mditoolbarView.cpp

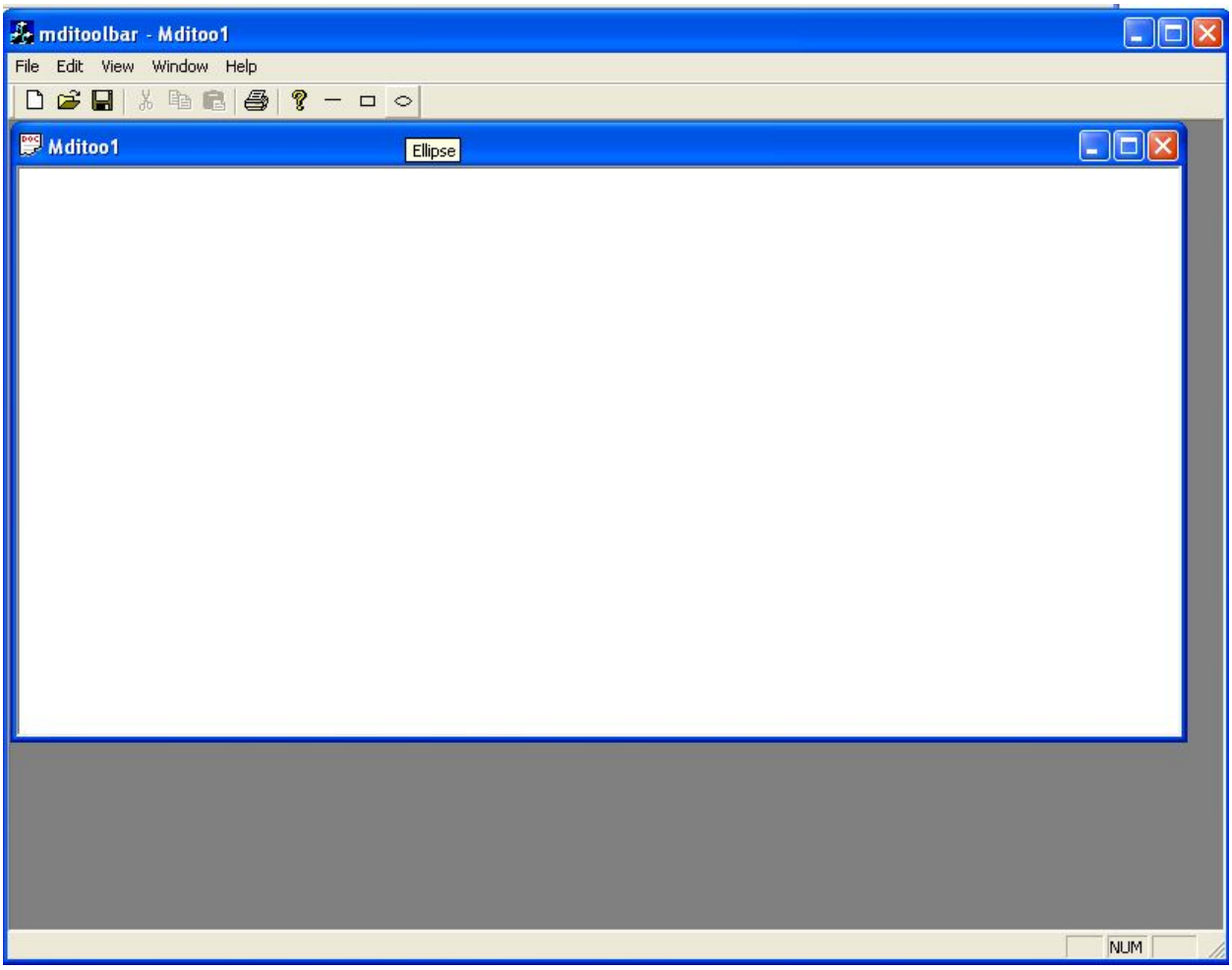
```
int s;
CPoint st;
void CMditoolbarView::OnEclipse() {
```

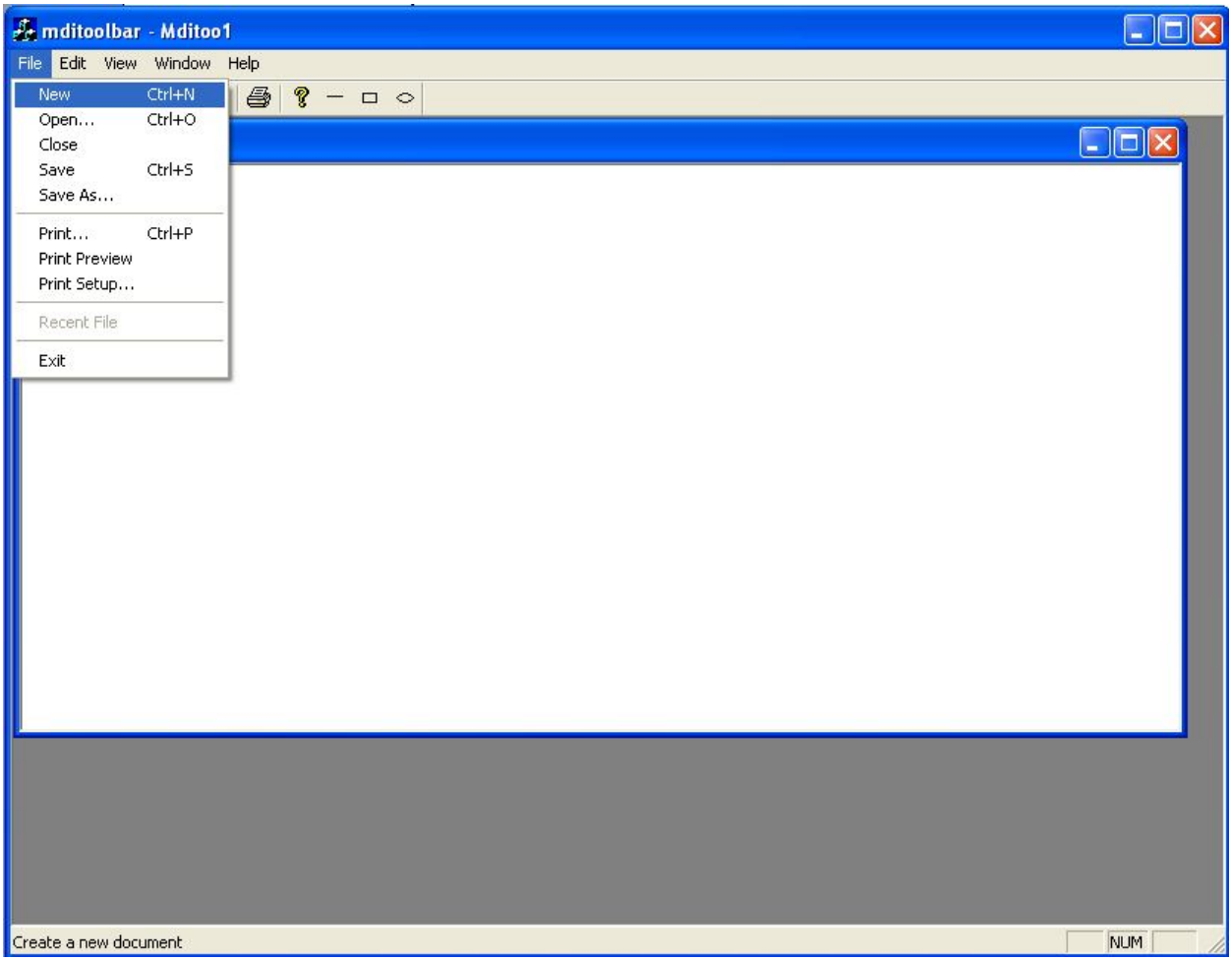
```
		s=0;
}

void CMditoolbarView::OnRectangle() {
		s=1;
}
void CMditoolbarView::OnLine() {
	s=2;
}

void CMditoolbarView::OnLButtonDown(UINT nFlags, CPoint point) {
	st=point;
	CView::OnLButtonDown(nFlags, point);
}

void CMditoolbarView::OnLButtonUp(UINT nFlags, CPoint point) {
	CClientDC *dc=new CClientDC(this);
	if (s==0)
	{
		dc->Ellipse(st.x,st.y,point.x,point.y);
	}
	if (s==1)
	{
		dc->Rectangle(st.x,st.y,point.x,point.y);
	}
```

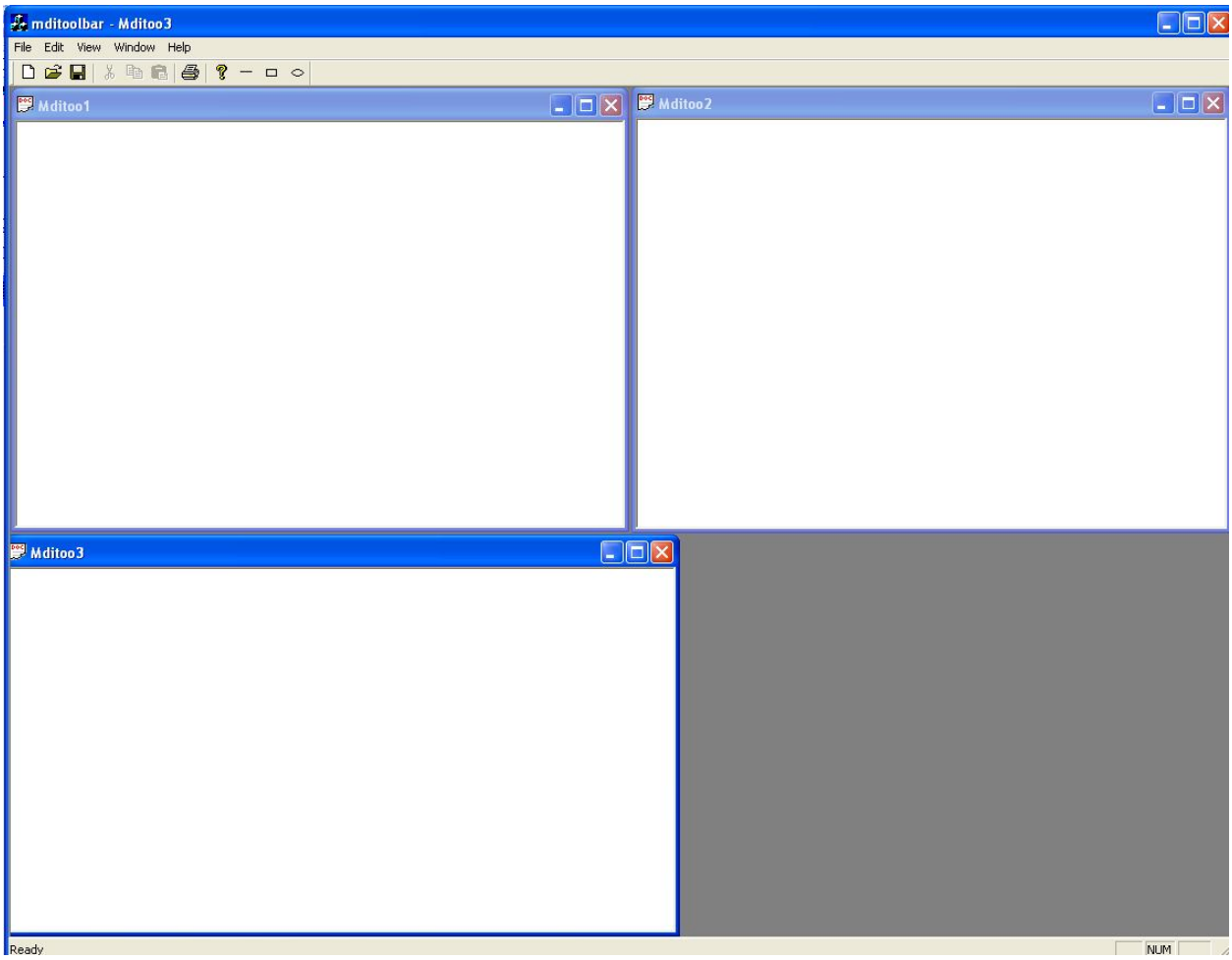
```
        if (s==2)
        {
                dc->MoveTo(st.x,st.y);
                dc->LineTo(point.x,point.y);
        }
        CView::OnLButtonUp(nFlags, point);
}
```

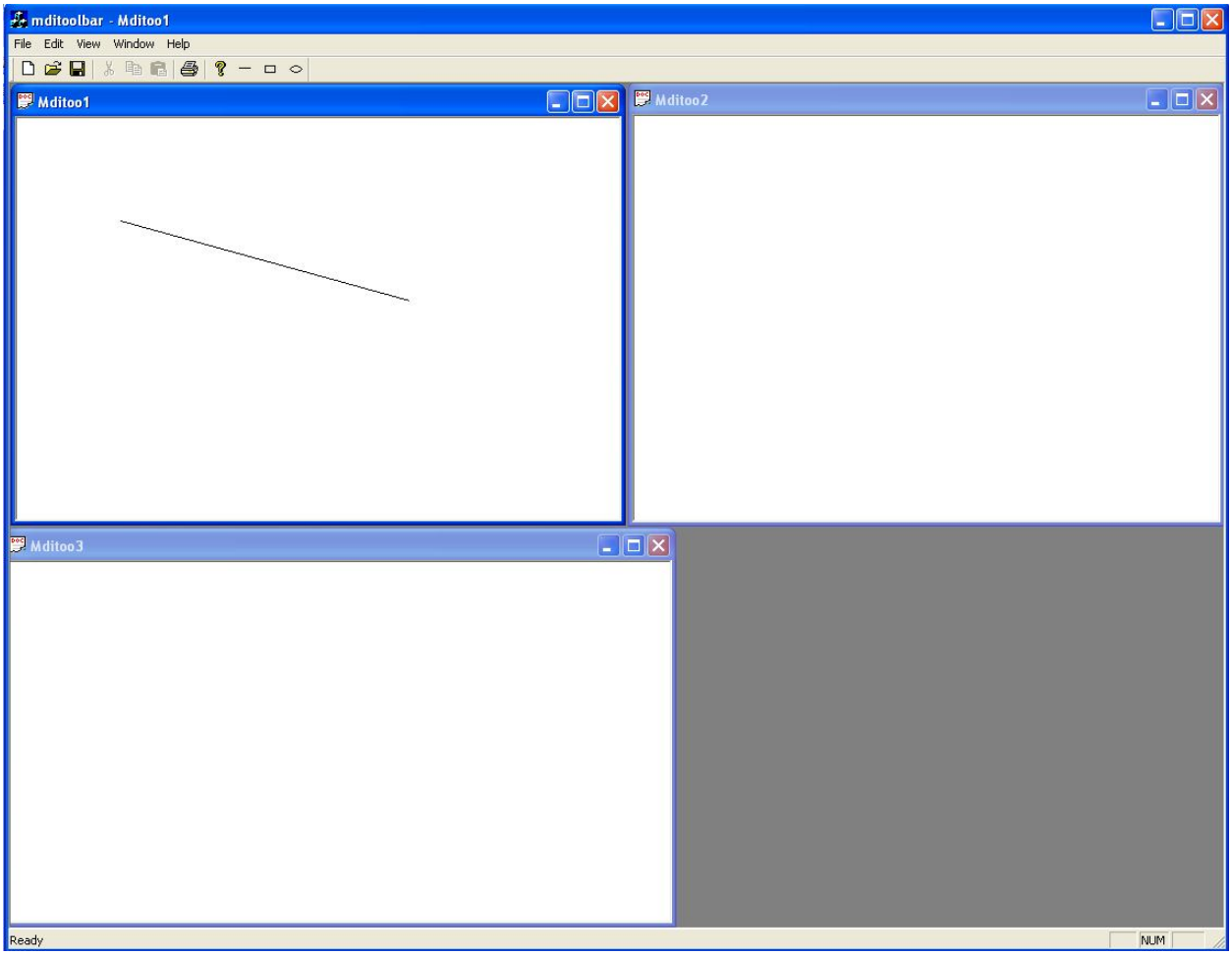
Description:

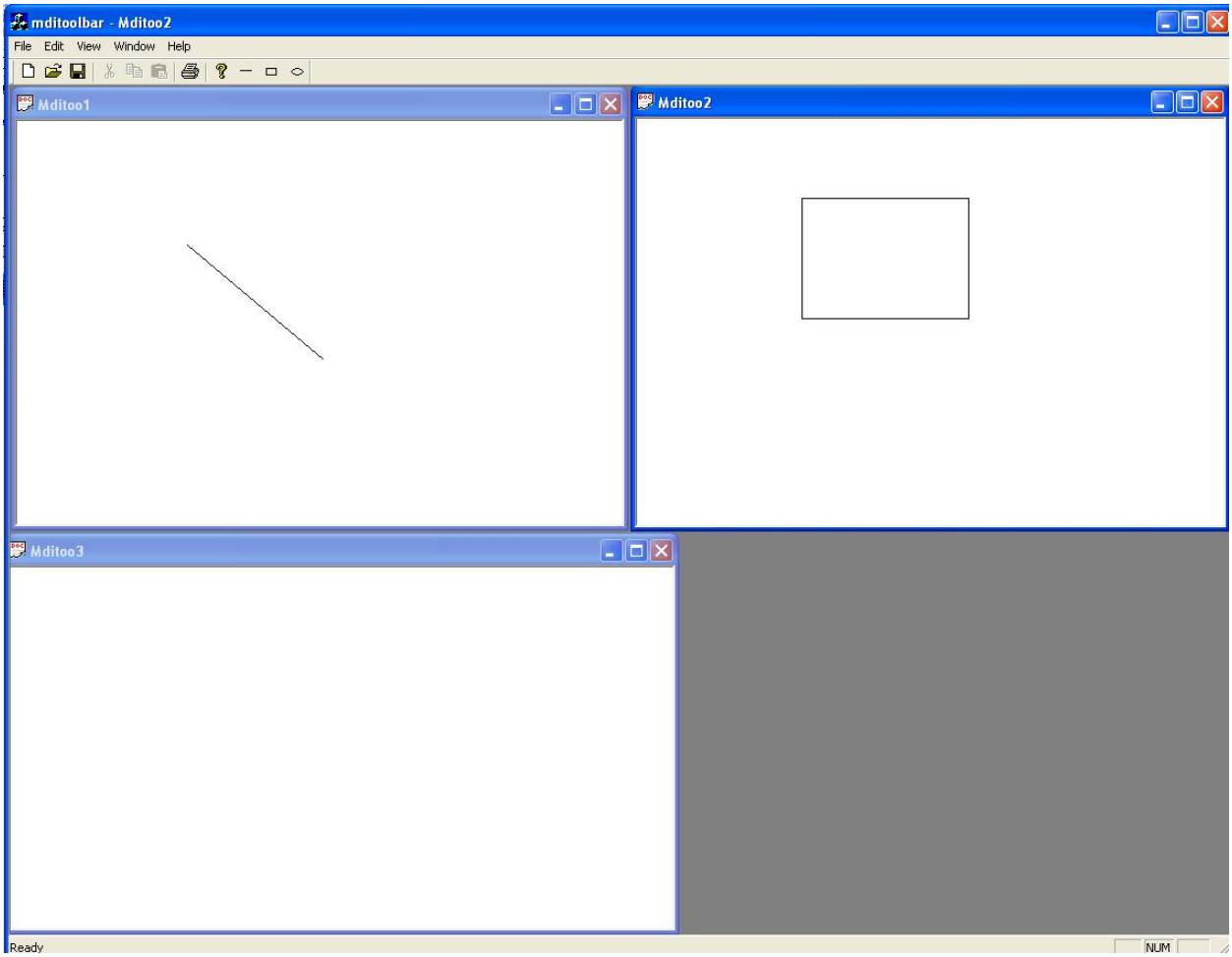
Output:

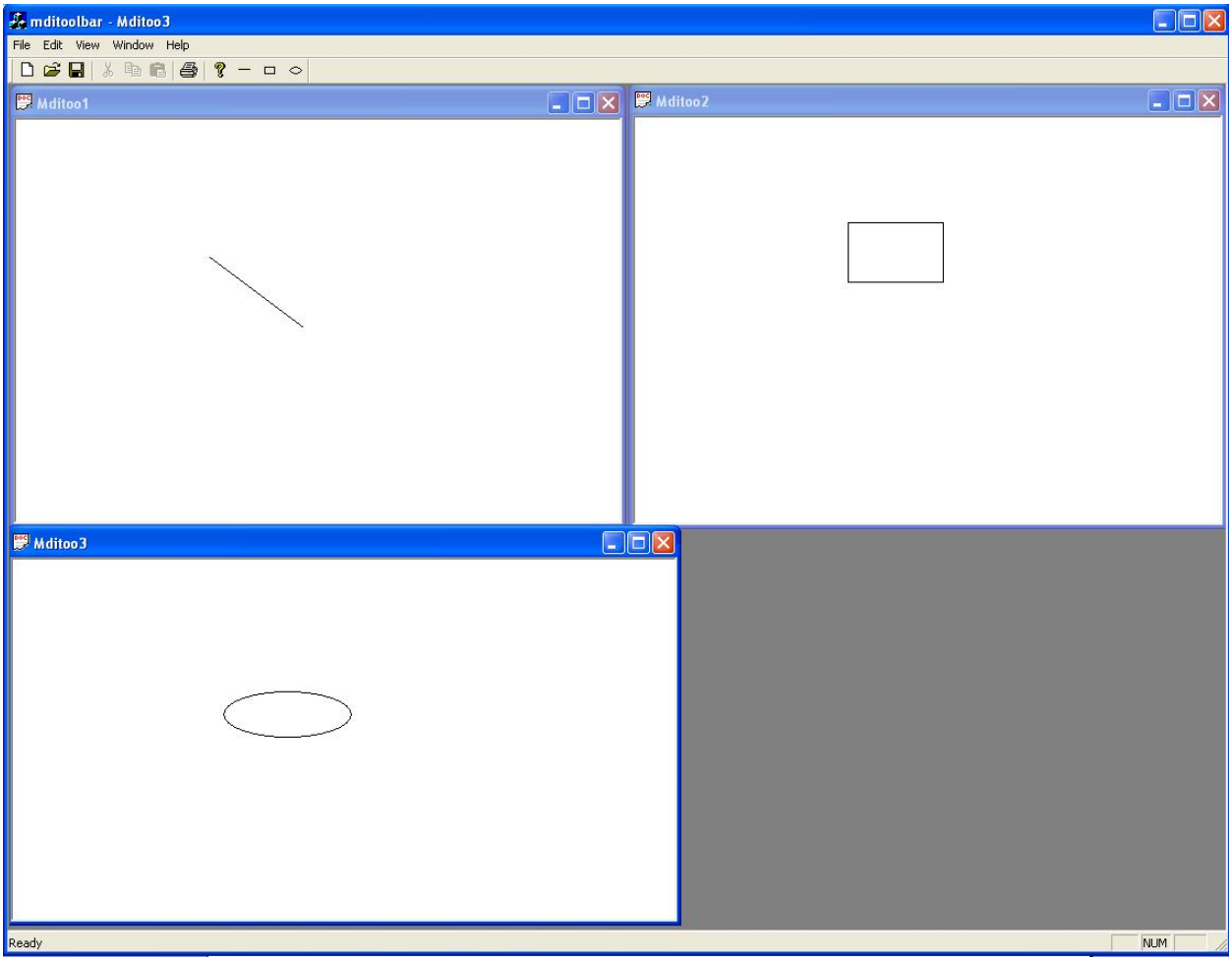



mditoolbar - Mditoo1
File
Edit
View
Window
Help
New Ctrl+N
Open... Ctrl+O
Close
Save Ctrl+S
Save As...
Print... Ctrl+P
Print Preview
Print Setup...
Recent File
Exit
Create a new document
NUM


mditoolbar - Mditoo3
File Edit View Window Help
Mditoo1
Mditoo2
Mditoo3
Ready
NUM


mditoolbar - Mditoo1
File Edit View Window Help
Mditoo1
Mditoo2
Mditoo3
Ready
NUM


mditoolbar - Mditoo2
File Edit View Window Help
Mditoo1
Mditoo2
Mditoo3
Ready
NUM



Result:

Thus the program for creating MDI application using MFC appwizard in VC++ was done successfully.

8. Menu Creation and Keyboard Accelerator

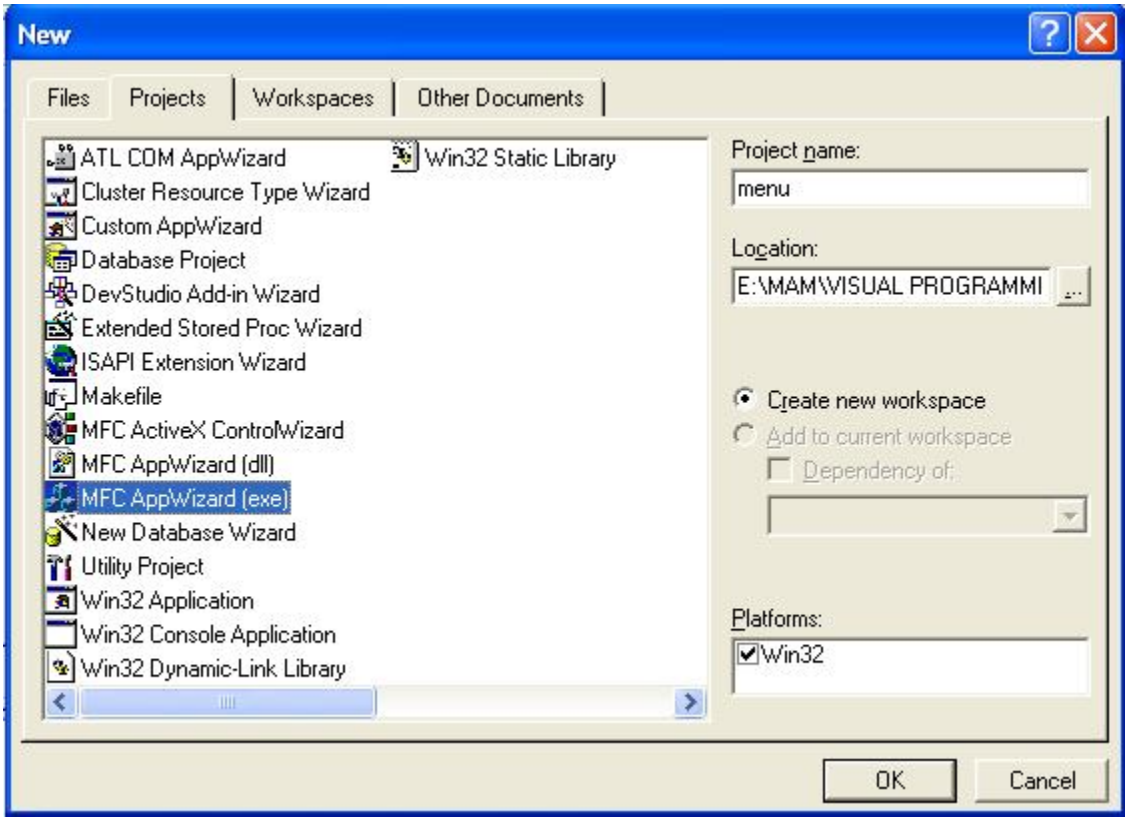
Aim:

To write a VC++ program to handle menus and keyboard accelerator on the window.

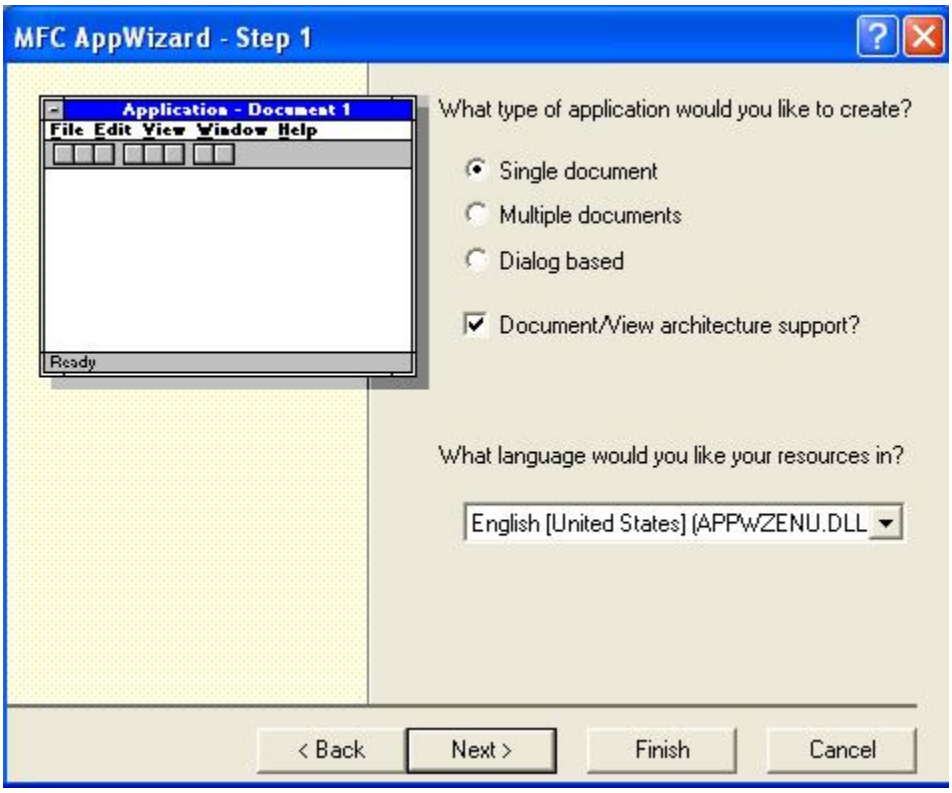
Concept:

Procedure:

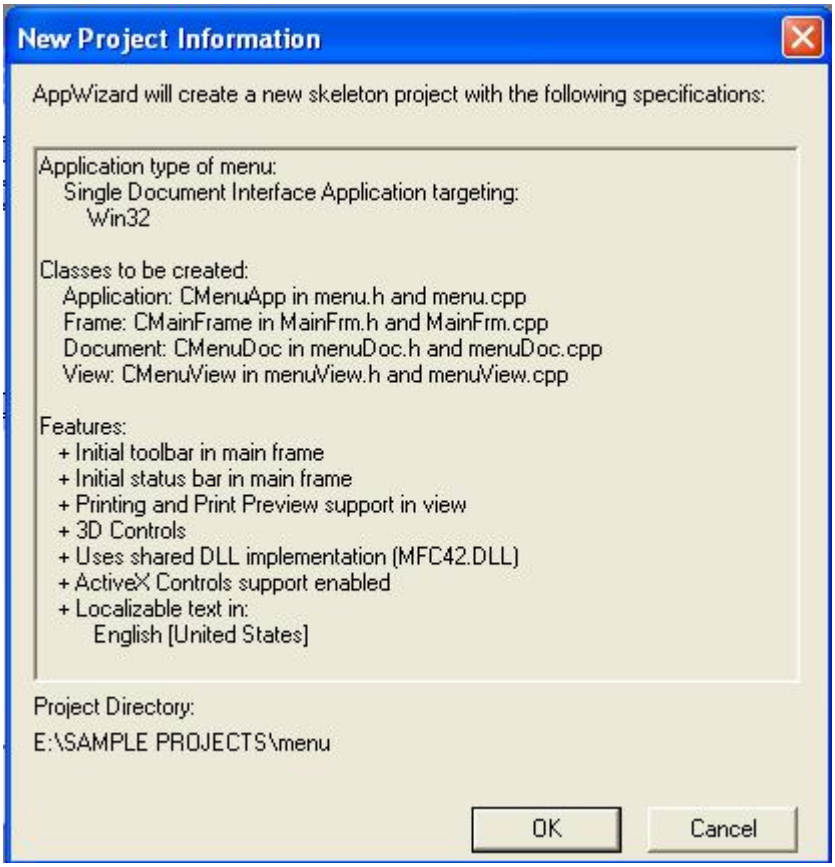
Step 1:



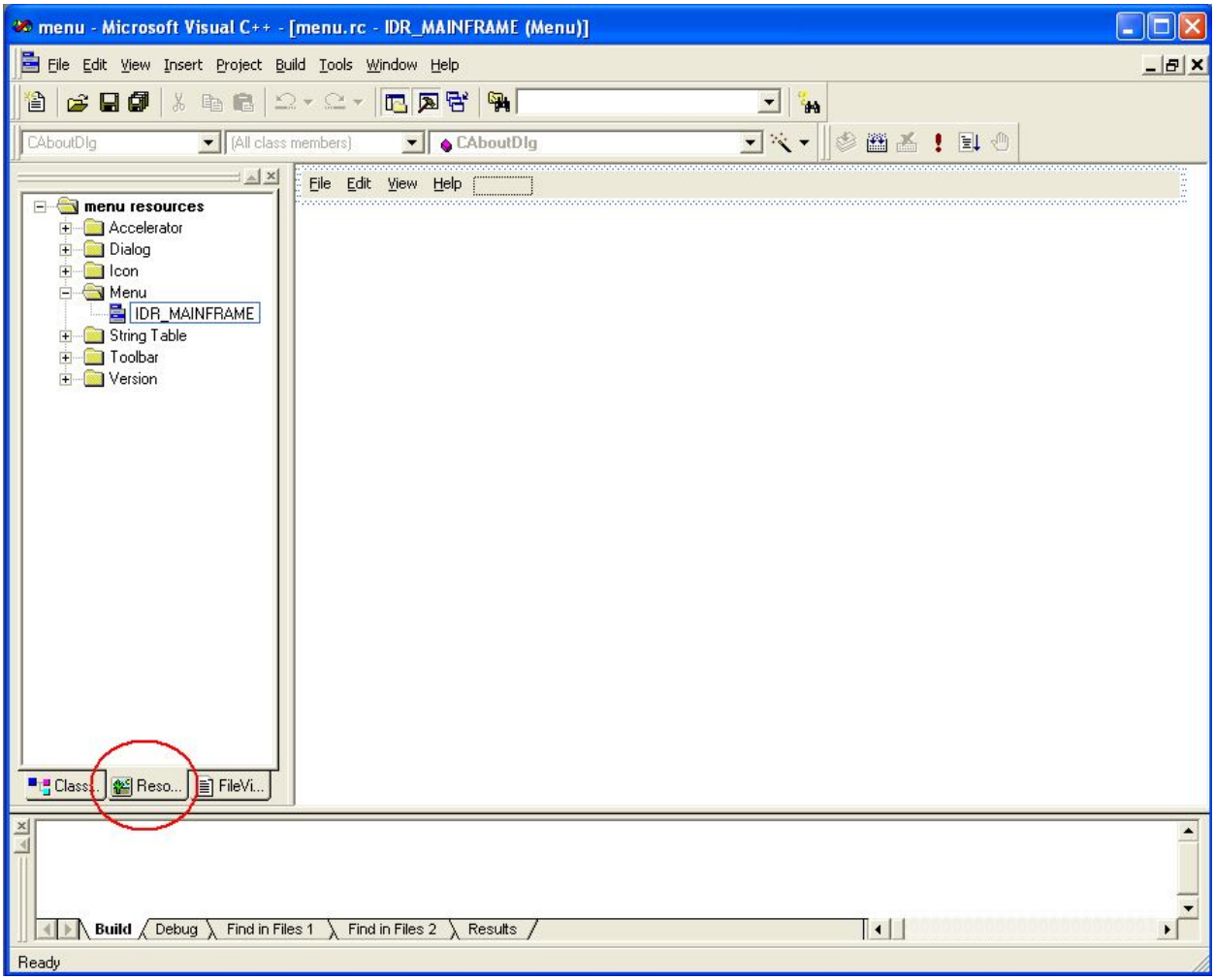
Step 2:


MFC AppWizard - Step 1
Application - Document 1
File Edit View Window Help
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

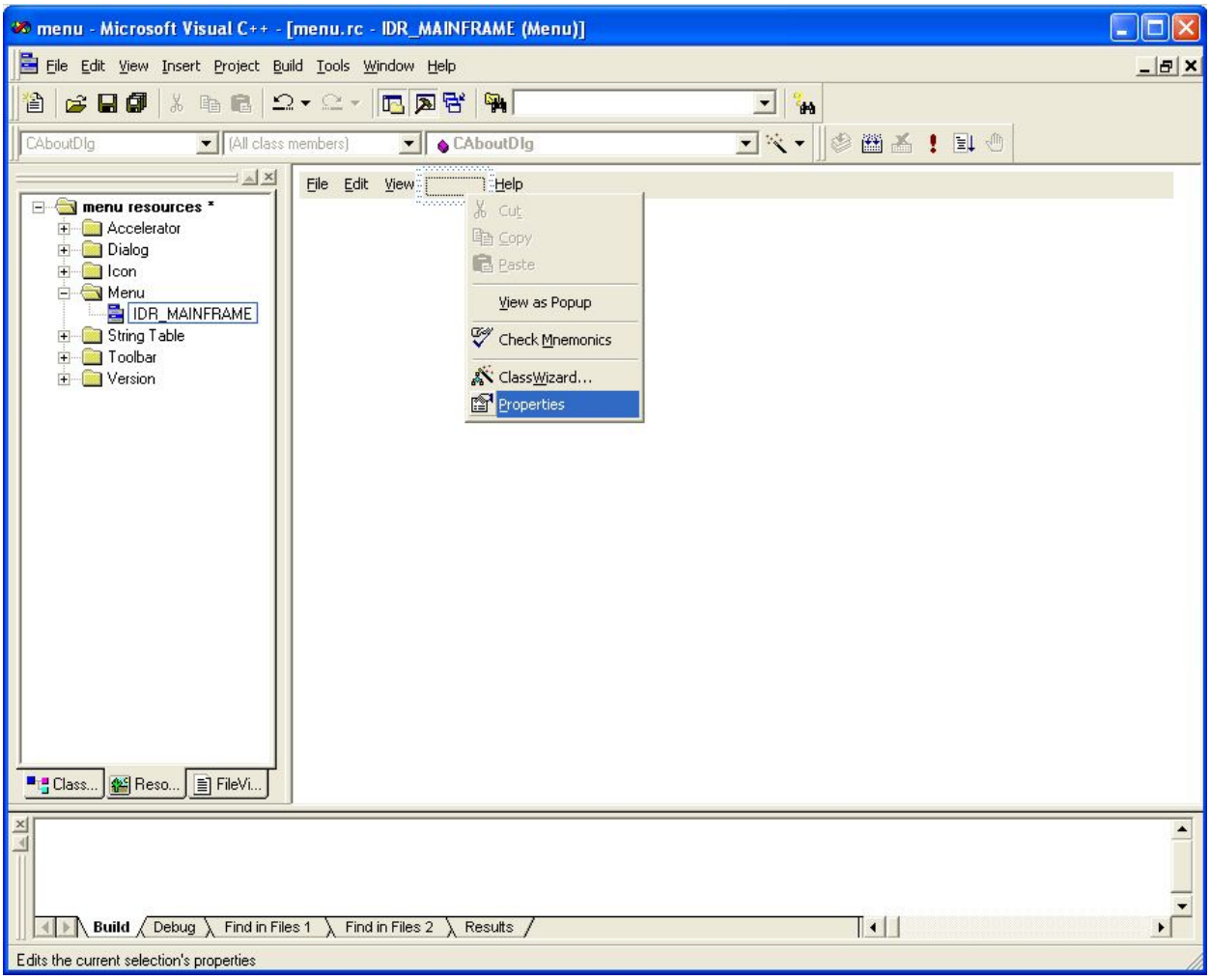
Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of menu:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CMenuApp in menu.h and menu.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CMenuDoc in menuDoc.h and menuDoc.cpp
View: CMenuView in menuView.h and menuView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\SAMPLE PROJECTS\menu
OK
Cancel

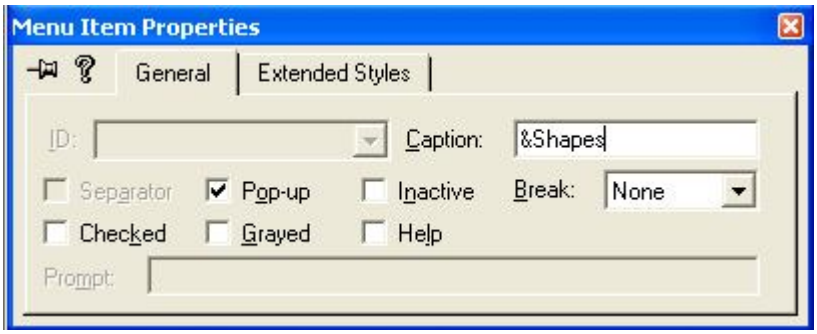
Step 4:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
menu resources
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
File Edit View Help
Class...
Reso...
FileVi...
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

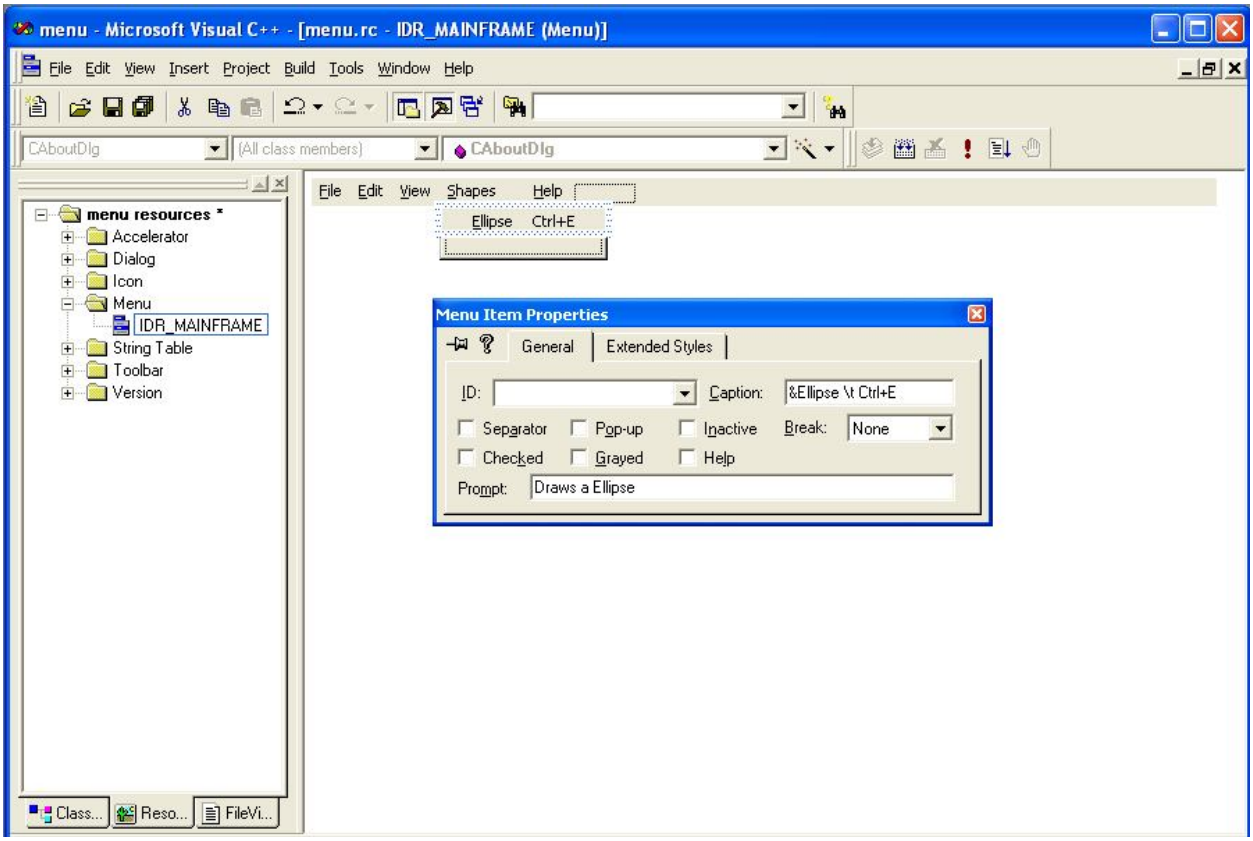
Step 5:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
menu resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Help
Cut
Copy
Paste
View as Popup
Check Mnemonics
ClassWizard...
Properties
Build
Debug
Find in Files 1
Find in Files 2
Results
Edits the current selection's properties

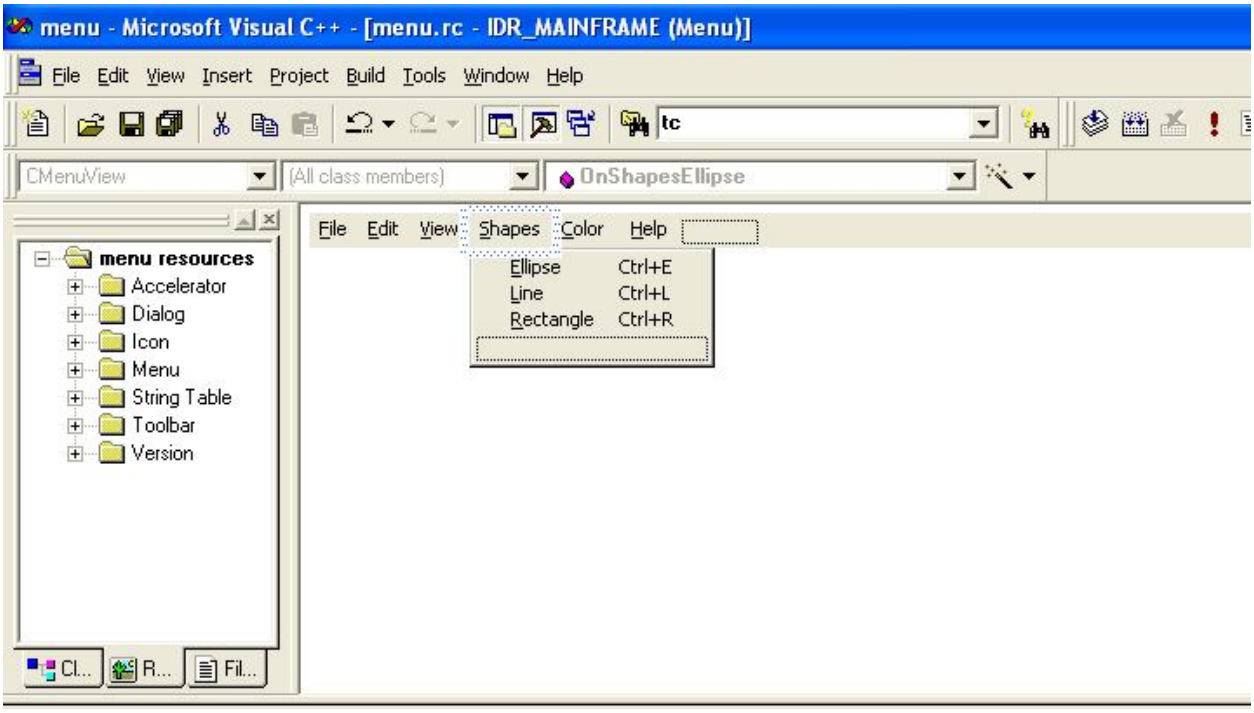
Step 6:


Menu Item Properties
General
Extended Styles
ID:
Caption:
&Shapes
Separator
Pop-up
Inactive
Break:
None
Checked
Grayed
Help
Prompt:

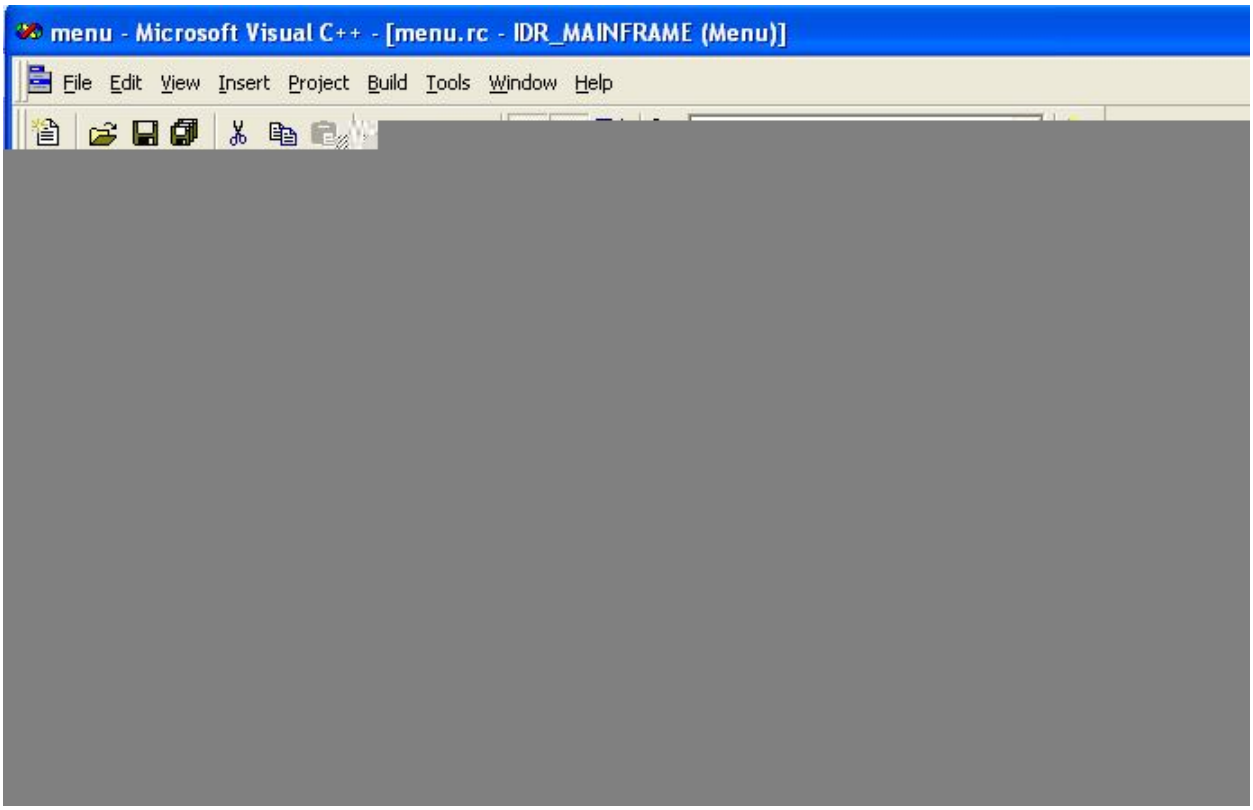
Step 7:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
menu resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Shapes Help
Ellipse Ctrl+E
Menu Item Properties
General
Extended Styles
ID:
Caption:
&Ellipse \t Ctrl+E
Separator
Pop-up
Inactive
Break:
None
Checked
Grayed
Help
Prompt:
Draws a Ellipse

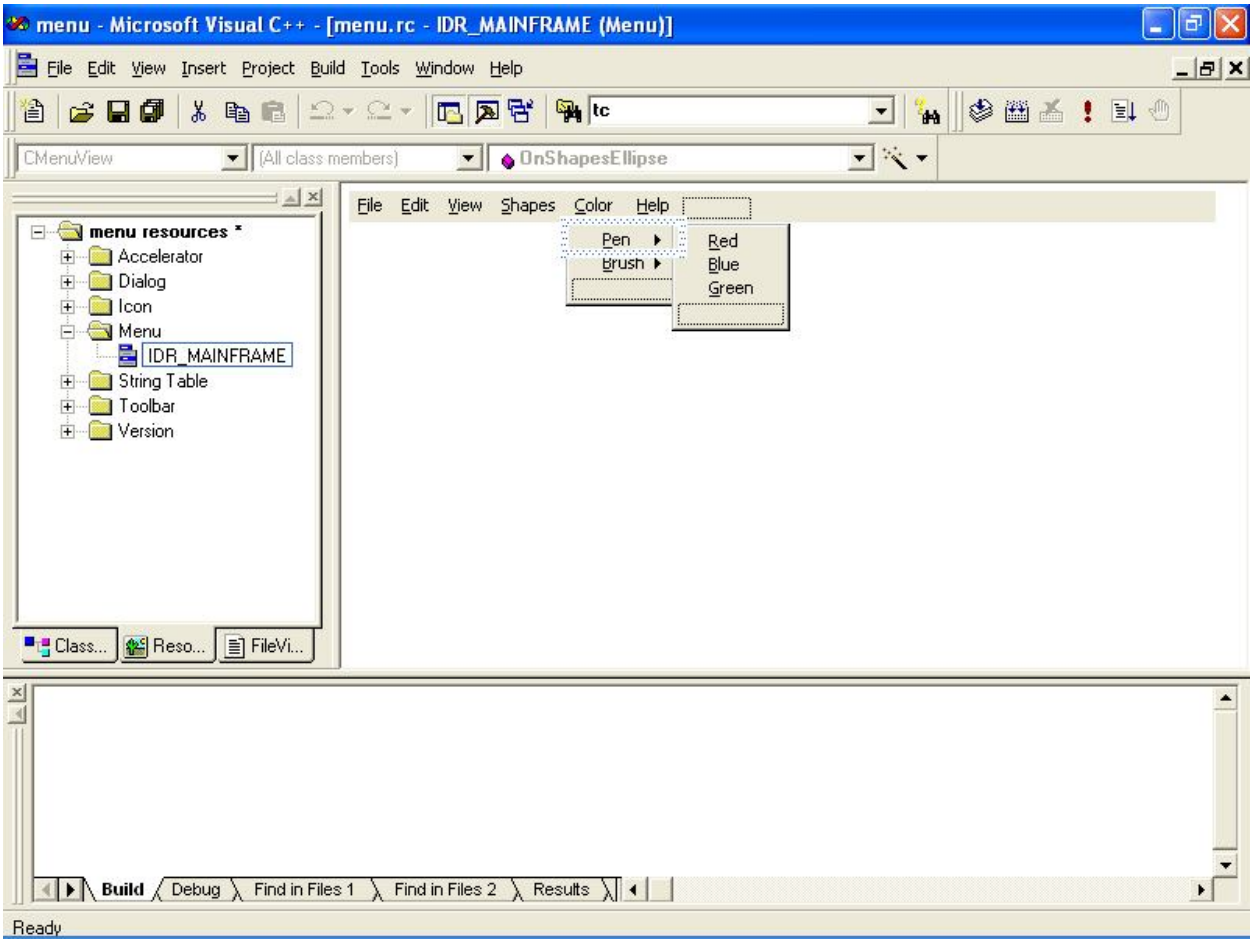
Step 8:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
tc
CMenuView
(All class members)
OnShapesEllipse
menu resources
Accelerator
Dialog
Icon
Menu
String Table
Toolbar
Version
File Edit View Shapes Color Help
Ellipse Ctrl+E
Line Ctrl+L
Rectangle Ctrl+R

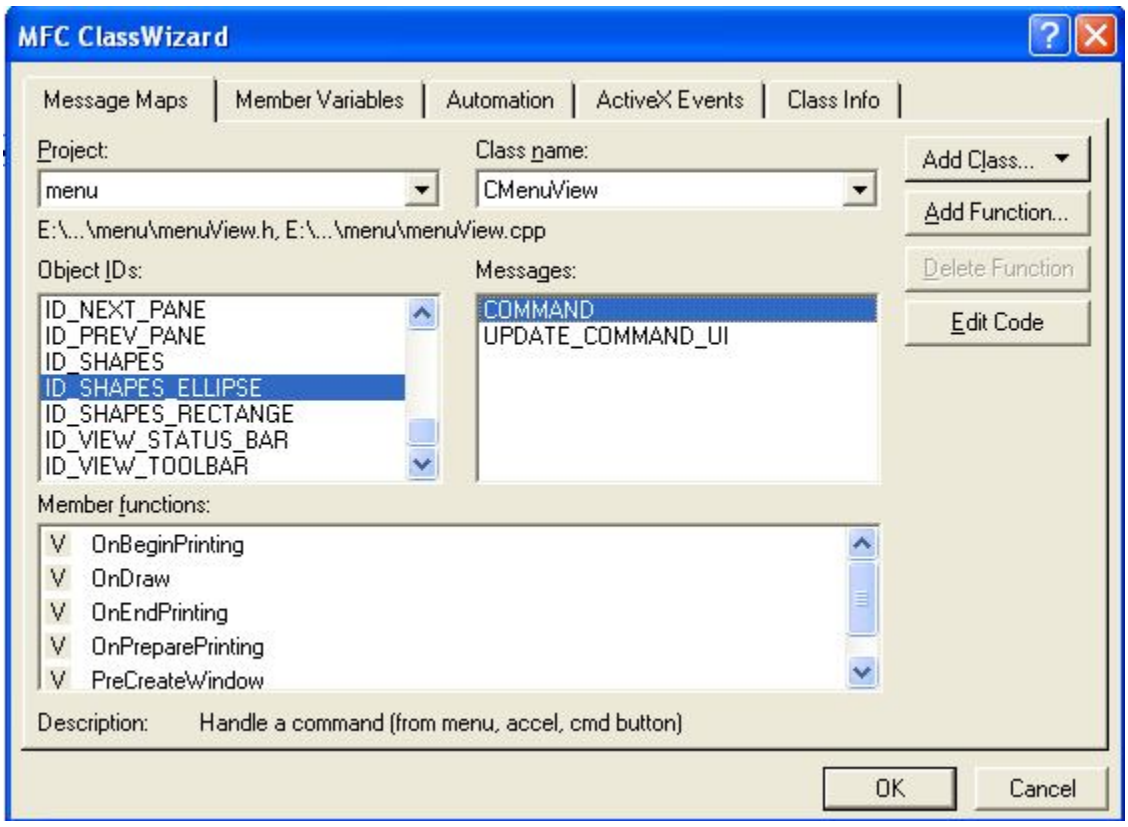
Step 9:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help

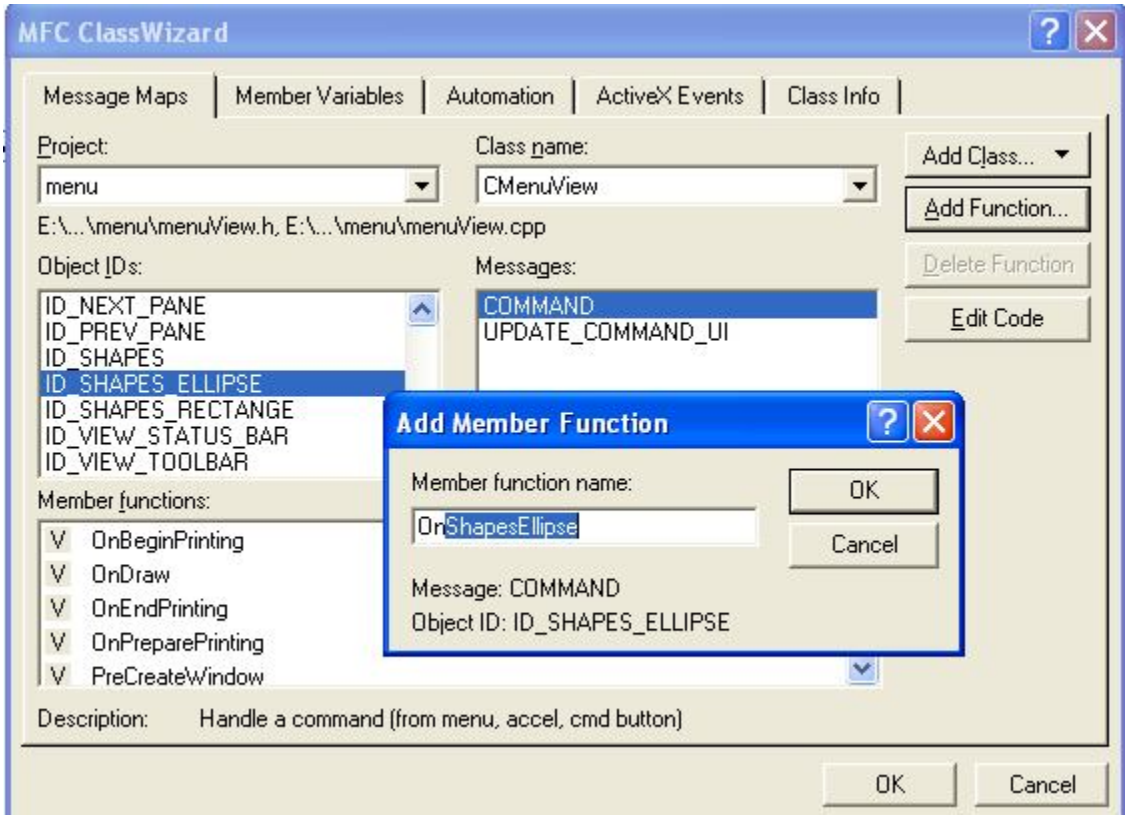
Step 9:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
tc
CMenuView
(All class members)
OnShapesEllipse
menu resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Shapes Color Help
Pen
Brush
Red
Blue
Green
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

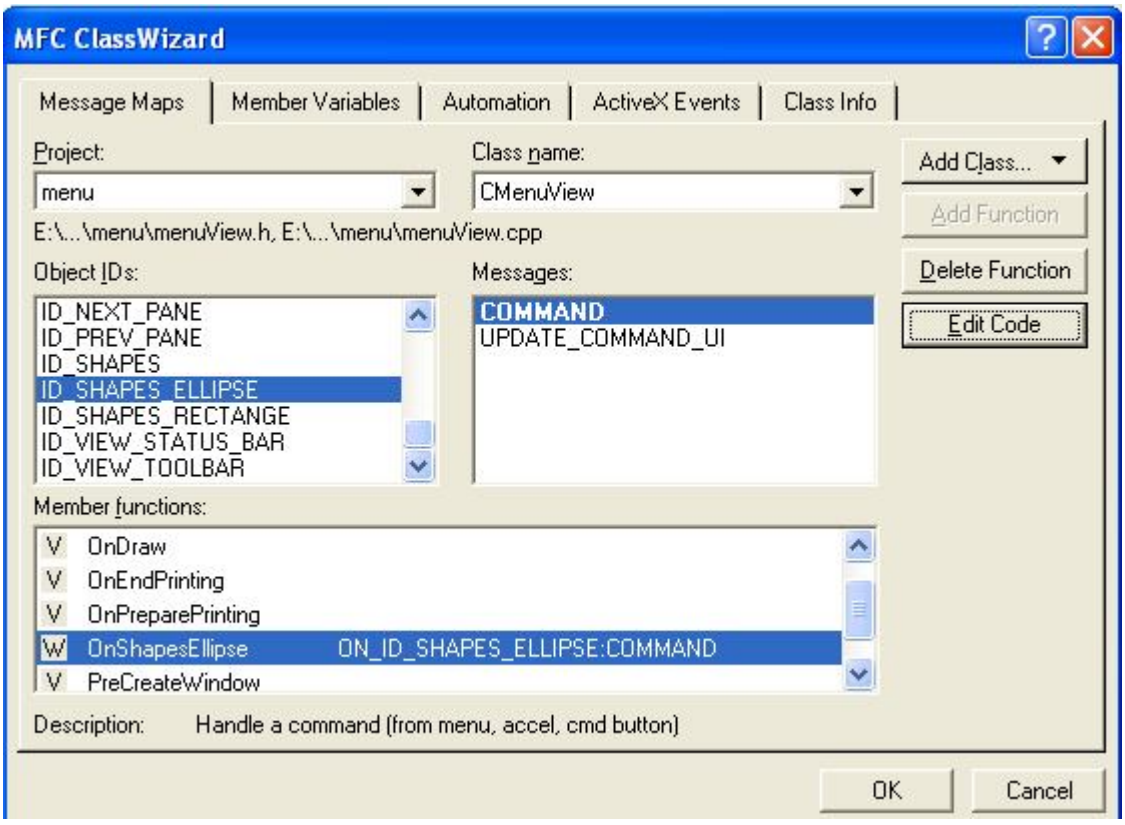
Step 10:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
menu
Class name:
CMenuView
Add Class...
Add Function...
Delete Function
Edit Code
E:\...\menu\menuView.h, E:\...\menu\menuView.cpp
Object IDs:
ID_NEXT_PANE
ID_PREV_PANE
ID_SHAPES
ID_SHAPES_ELLIPSE
ID_SHAPES_RECTANGE
ID_VIEW_STATUS_BAR
ID_VIEW_TOOLBAR
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
V OnPreparePrinting
V PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

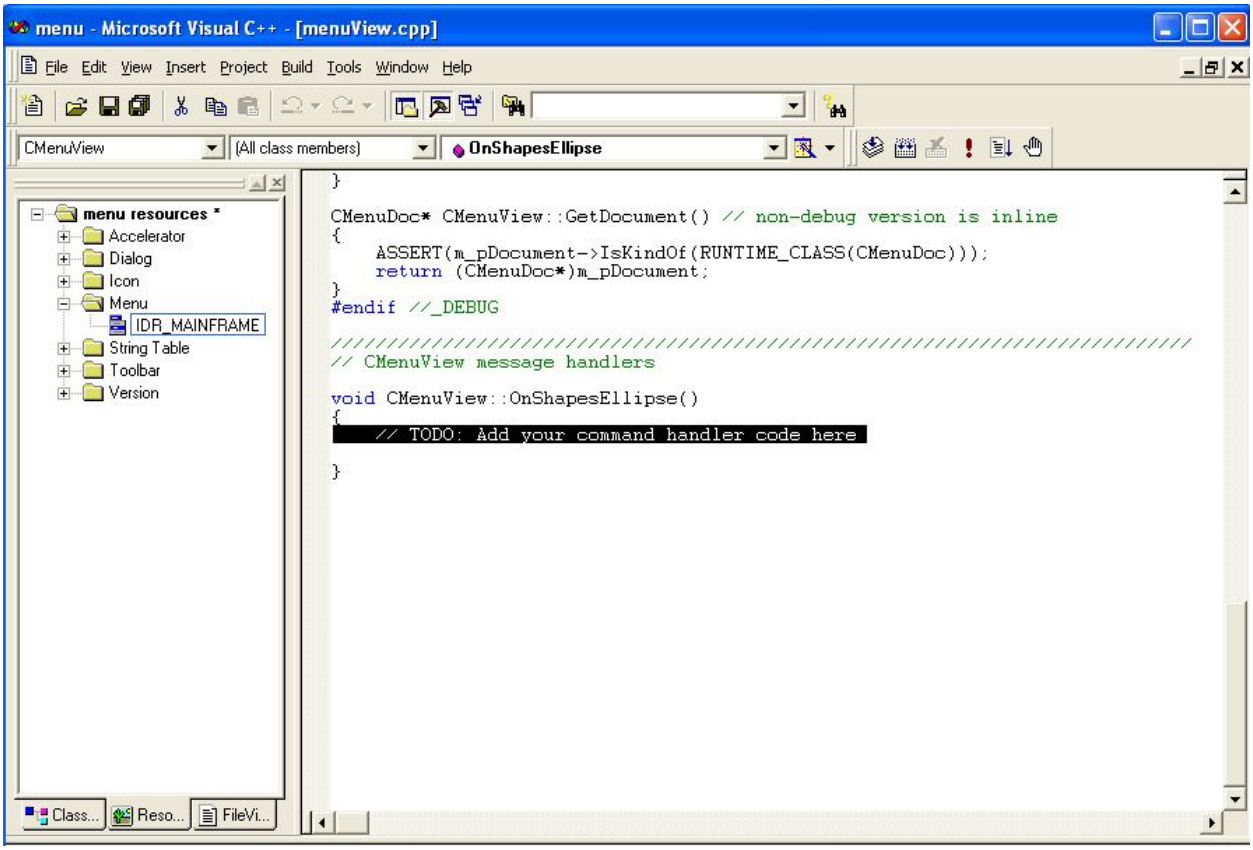
Step 11:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
menu
Class name:
CMenuView
Add Class...
Add Function...
Delete Function
Edit Code
E:\...\menu\menuView.h, E:\...\menu\menuView.cpp
Object IDs:
ID_NEXT_PANE
ID_PREV_PANE
ID_SHAPES
ID_SHAPES_ELLIPSE
ID_SHAPES_RECTANGE
ID_VIEW_STATUS_BAR
ID_VIEW_TOOLBAR
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
V OnPreparePrinting
V PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel
Add Member Function
Member function name:
OnShapesEllipse
OK
Cancel
Message: COMMAND
Object ID: ID_SHAPES_ELLIPSE

Step 12:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
menu
Class name:
CMenuView
Add Class...
Add Function
Delete Function
Edit Code
E:\...\menu\menuView.h, E:\...\menu\menuView.cpp
Object IDs:
ID_NEXT_PANE
ID_PREV_PANE
ID_SHAPES
ID_SHAPES_ELLIPSE
ID_SHAPES_RECTANGE
ID_VIEW_STATUS_BAR
ID_VIEW_TOOLBAR
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
OnDraw
OnEndPrinting
OnPreparePrinting
OnShapesEllipse ON_ID_SHAPES_ELLIPSE:COMMAND
PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

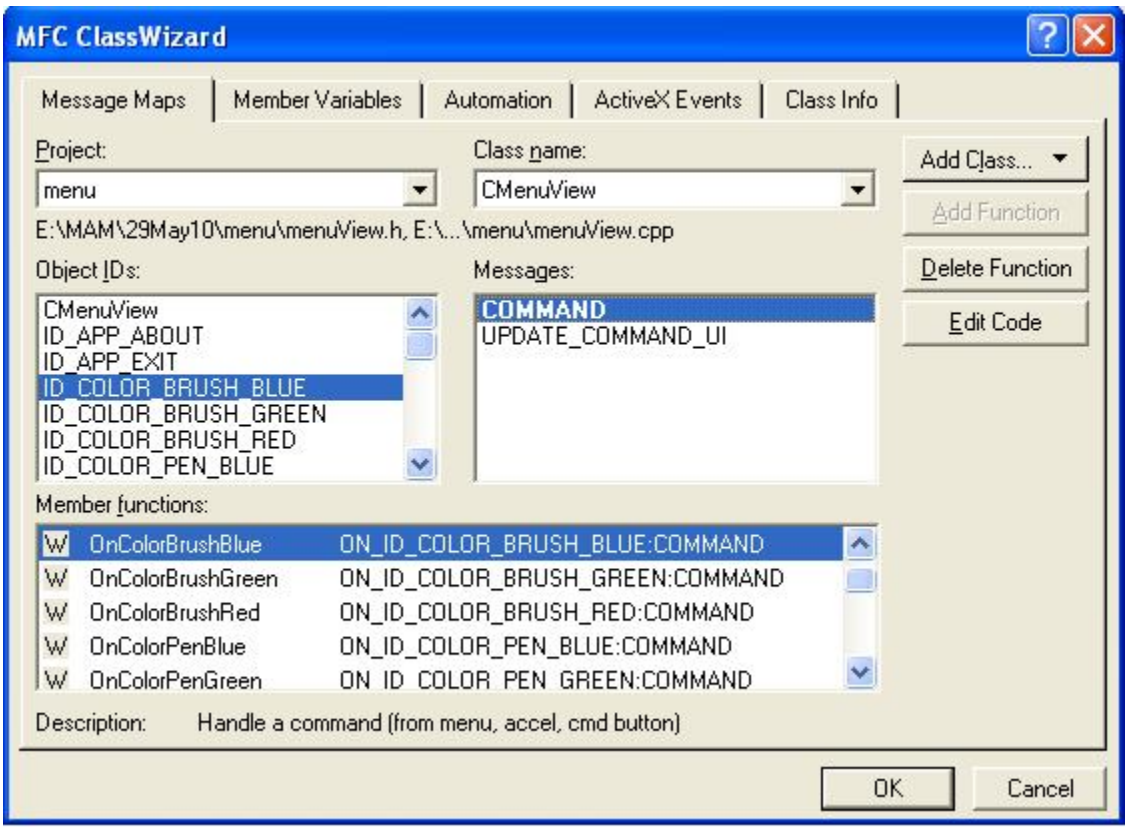
Step 13:



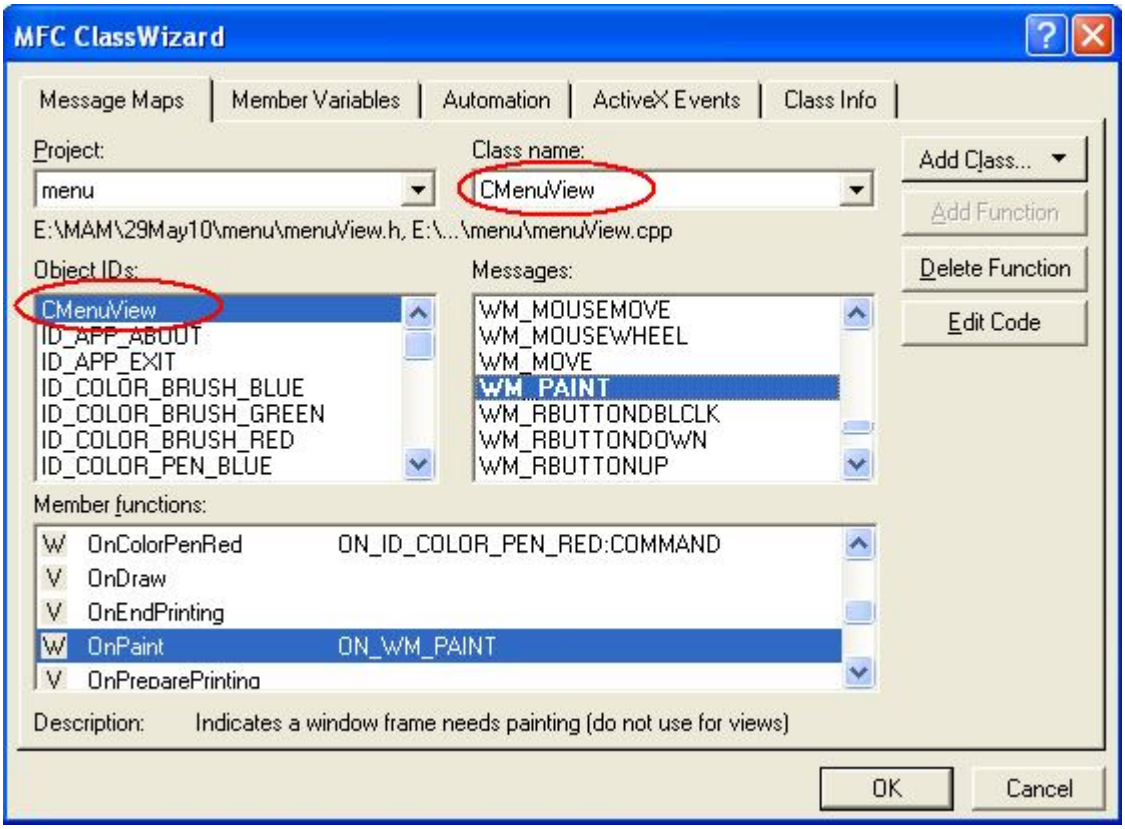
Step 14:

```
void CMenuView::OnShapesEllipse()
{
        s=3;
        Invalidate();
}
```

Step 15:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
menu
Class name:
CMenuView
Add Class...
Add Function
Delete Function
Edit Code
E:\MAM\29May10\menu\menuView.h, E:\...\menu\menuView.cpp
Object IDs:
CMenuView
ID_APP_ABOUT
ID_APP_EXIT
ID_COLOR_BRUSH_BLUE
ID_COLOR_BRUSH_GREEN
ID_COLOR_BRUSH_RED
ID_COLOR_PEN_BLUE
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
OnColorBrushBlue ON_ID_COLOR_BRUSH_BLUE:COMMAND
OnColorBrushGreen ON_ID_COLOR_BRUSH_GREEN:COMMAND
OnColorBrushRed ON_ID_COLOR_BRUSH_RED:COMMAND
OnColorPenBlue ON_ID_COLOR_PEN_BLUE:COMMAND
OnColorPenGreen ON_ID_COLOR_PEN_GREEN:COMMAND
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

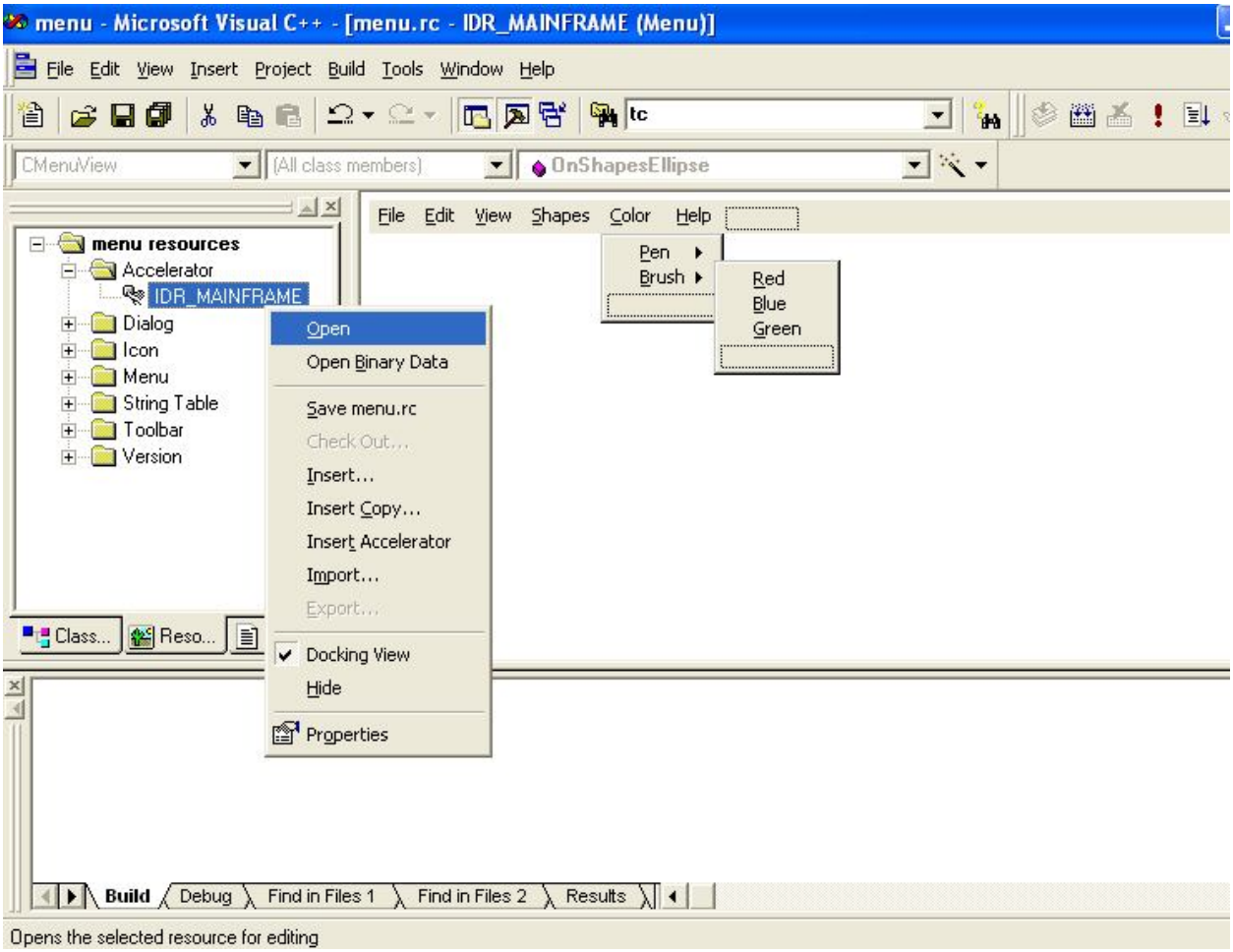
Step 16:



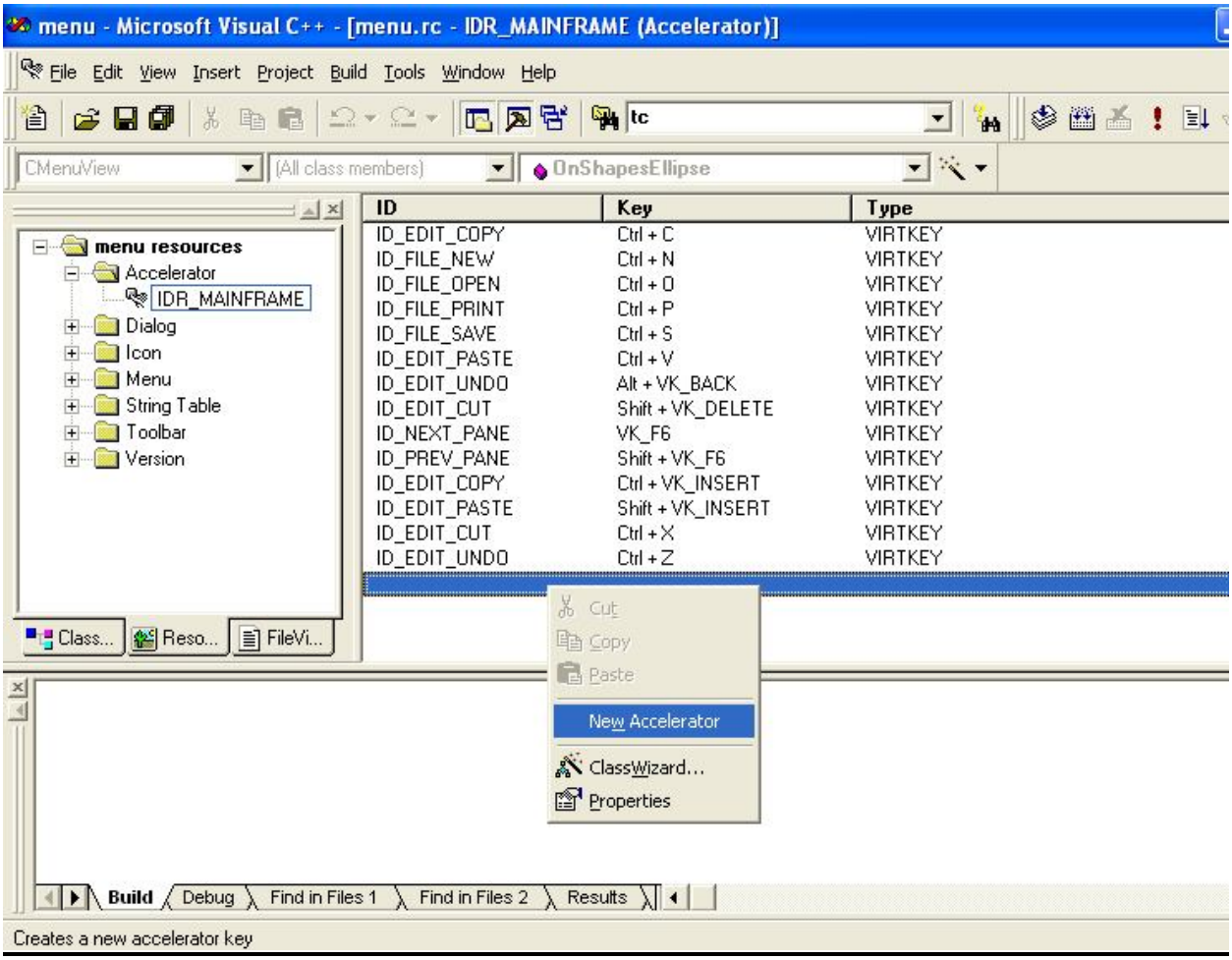
Step 17:

Keyboard Accelerator:

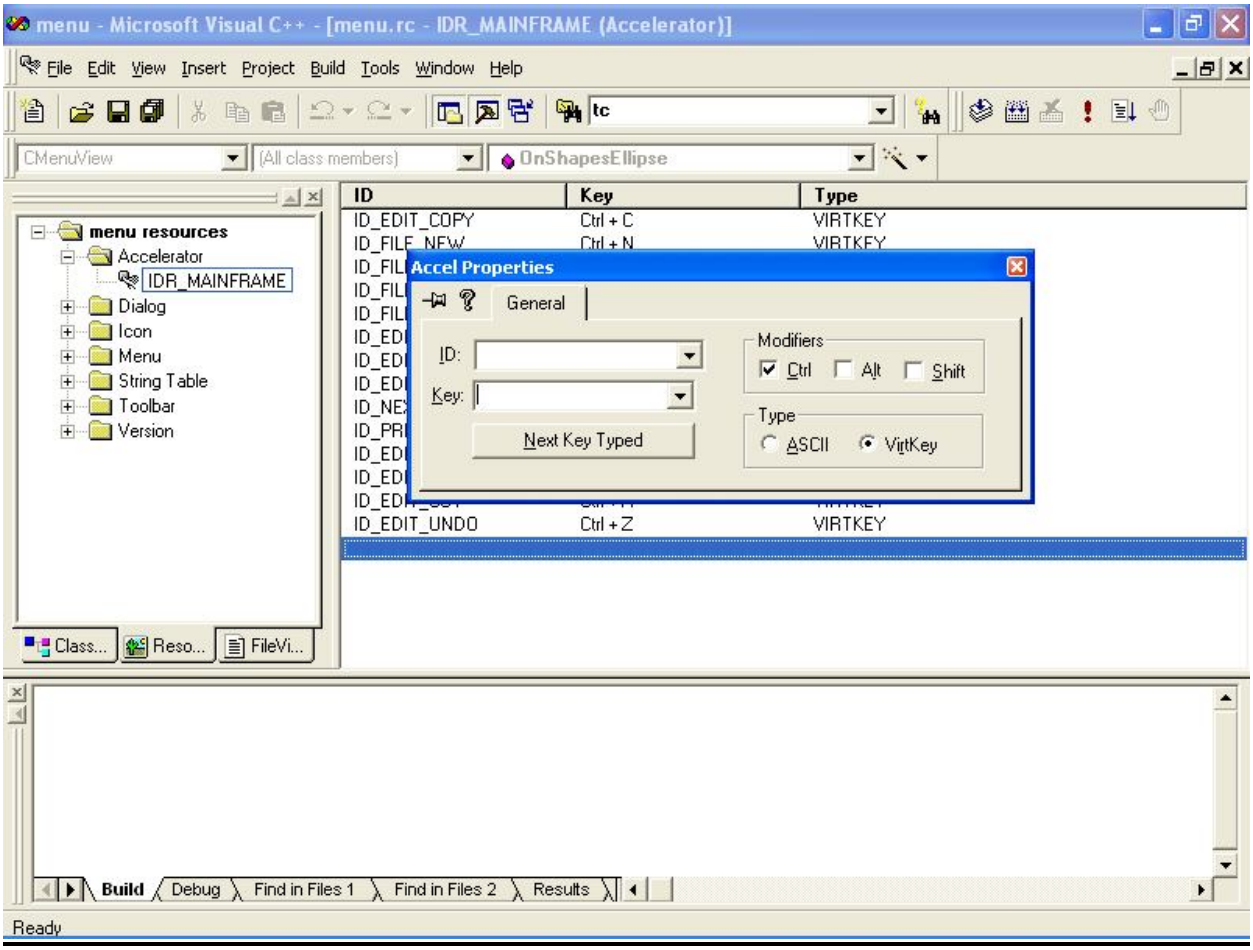
Step 13:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
tc
CMenuView
(All class members)
OnShapesEllipse
menu resources
Accelerator
IDR_MAINFRAME
Dialog
Icon
Menu
String Table
Toolbar
Version
File Edit View Shapes Color Help
Pen
Brush
Red
Blue
Green
Open
Open Binary Data
Save menu.rc
Check Out...
Insert...
Insert Copy...
Insert Accelerator
Import...
Export...
Docking View
Hide
Properties
Class...
Reso...
Build
Debug
Find in Files 1
Find in Files 2
Results
Opens the selected resource for editing

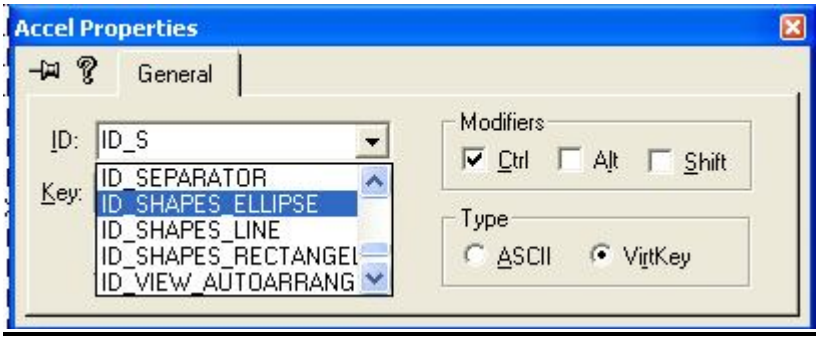
Step 14:


menu - Microsoft Visual C++ - [menu.rc - IDR_MAINFRAME (Accelerator)]
File Edit View Insert Project Build Tools Window Help
tc
CMenuView
(All class members)
OnShapesEllipse
menu resources
Accelerator
IDR_MAINFRAME
Dialog
Icon
Menu
String Table
Toolbar
Version
Class...
Reso...
FileVi...
ID
Key
Type
ID_EDIT_COPY Ctrl + C VIRTKEY
ID_FILE_NEW Ctrl + N VIRTKEY
ID_FILE_OPEN Ctrl + O VIRTKEY
ID_FILE_PRINT Ctrl + P VIRTKEY
ID_FILE_SAVE Ctrl + S VIRTKEY
ID_EDIT_PASTE Ctrl + V VIRTKEY
ID_EDIT_UNDO Alt + VK_BACK VIRTKEY
ID_EDIT_CUT Shift + VK_DELETE VIRTKEY
ID_NEXT_PANE VK_F6 VIRTKEY
ID_PREV_PANE Shift + VK_F6 VIRTKEY
ID_EDIT_COPY Ctrl + VK_INSERT VIRTKEY
ID_EDIT_PASTE Shift + VK_INSERT VIRTKEY
ID_EDIT_CUT Ctrl + X VIRTKEY
ID_EDIT_UNDO Ctrl + Z VIRTKEY
Cut
Copy
Paste
New Accelerator
ClassWizard...
Properties
Build
Debug
Find in Files 1
Find in Files 2
Results
Creates a new accelerator key

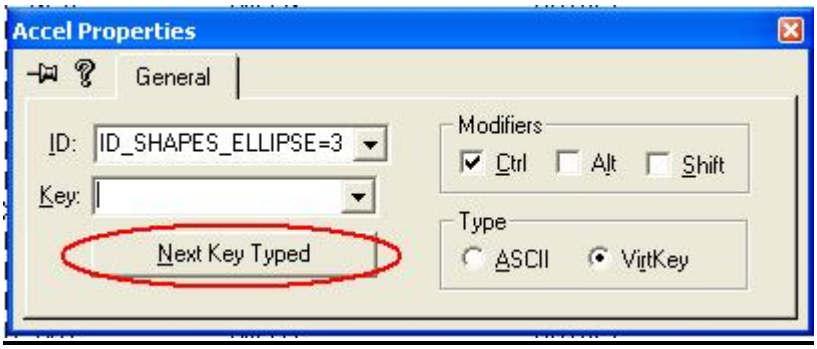
Step 15:



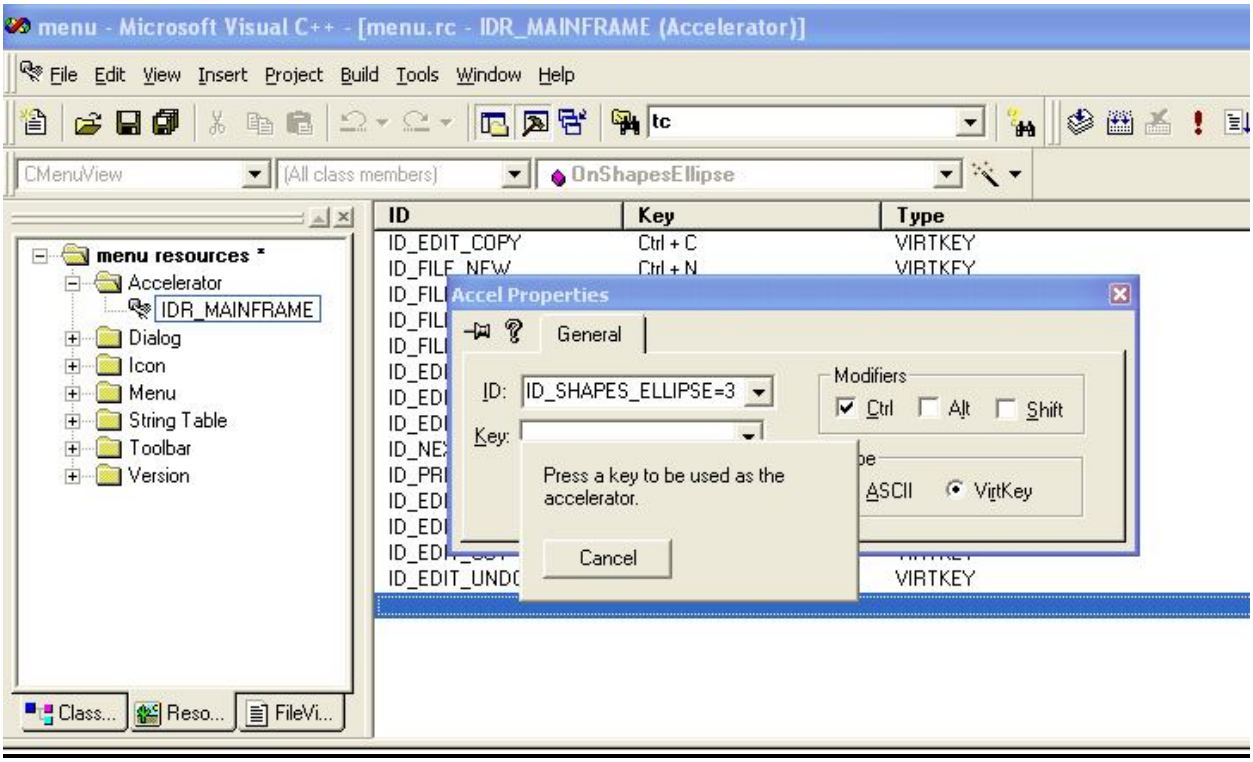
Step 16:



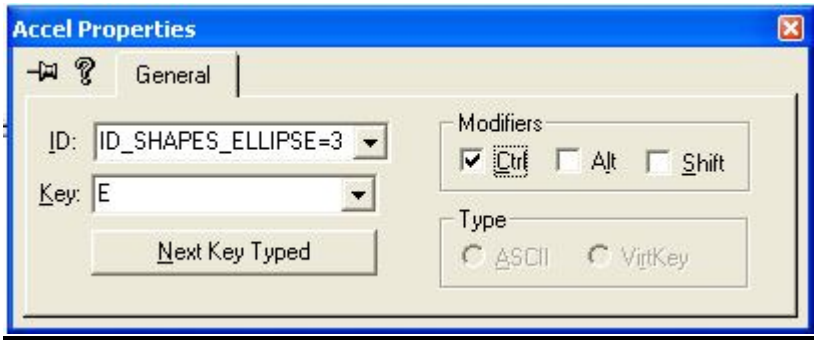
Step 17:



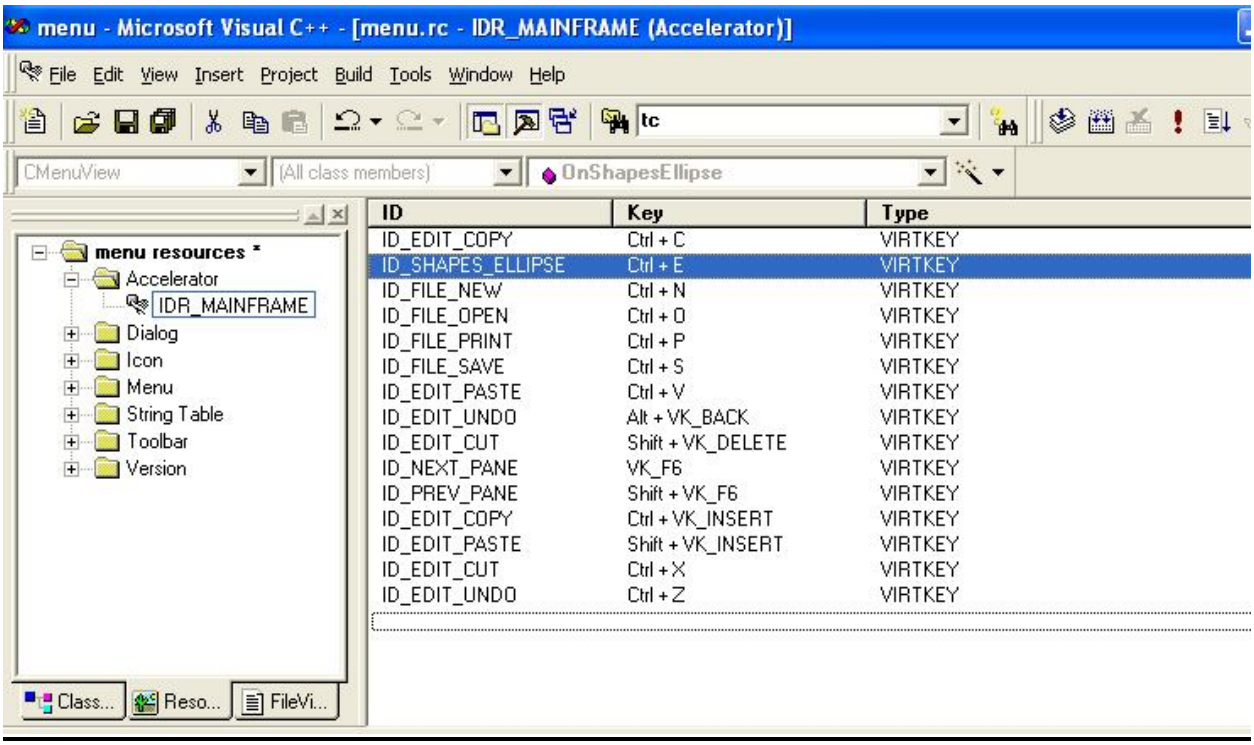
Step 18:



Step 19:



Step 20:



Program:

//menuView.cpp

```
int s,b,p;


void CMenuView::OnShapesEllipse()
{
        s=3;
        Invalidate();
}
void CMenuView::OnShapesLine()
{
```

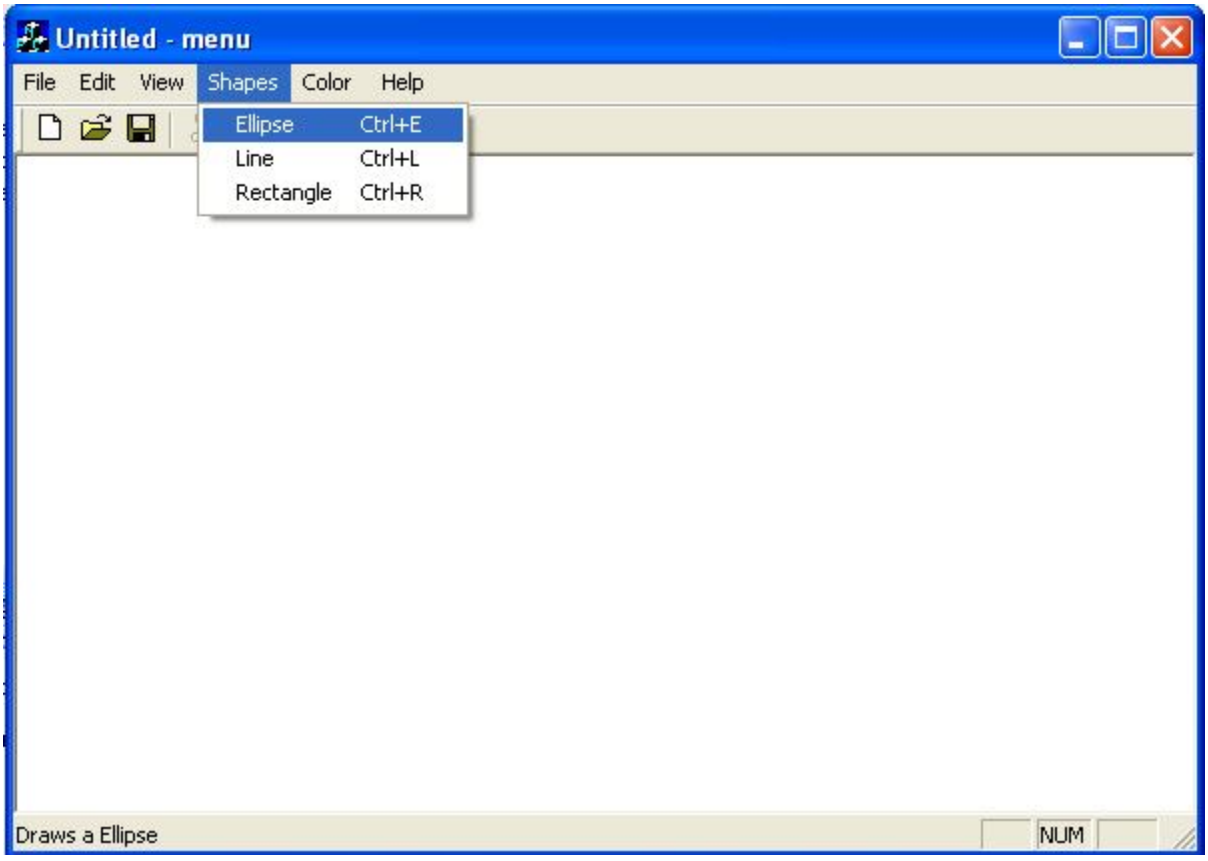
```
        s=1;
        Invalidate();
}

void CMenuView::OnShapesRectangle()
{
        s=2;
        Invalidate();
}

void CMenuView::OnColorPenRed()
{
        p=1;
        Invalidate();
}

void CMenuView::OnColorPenGreen()
{
        p=2;
        Invalidate();
}

void CMenuView::OnColorPenBlue()
{
        p=3;
```

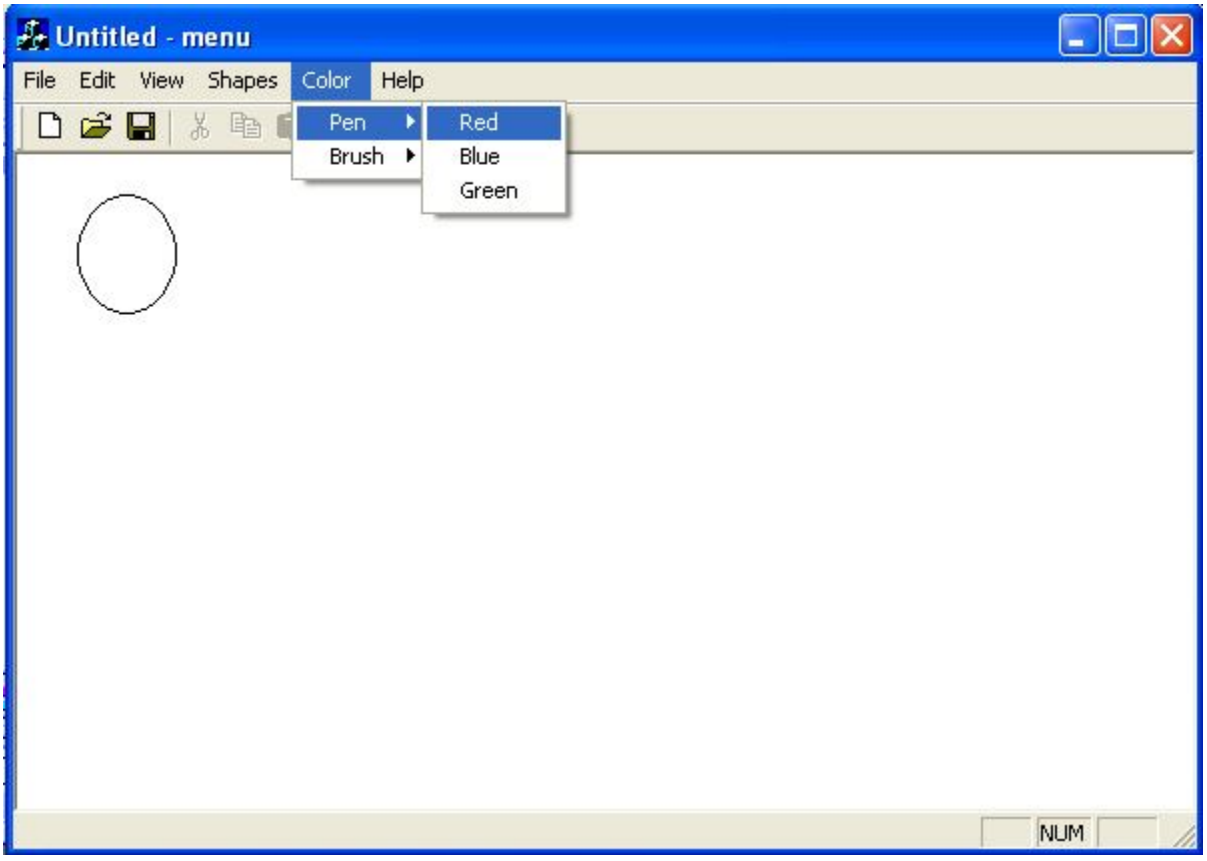
```
        Invalidate();
}

void CMenuView::OnColorBrushRed()
{
        b=1;
        Invalidate();
}

void CMenuView::OnColorBrushGreen()
{
        b=2;
        Invalidate();
}

void CMenuView::OnColorBrushBlue()
{
        b=3;
        Invalidate();
}

void CMenuView::OnPaint()
{
        CPaintDC dc(this); // device context for painting
        CPen P;
```

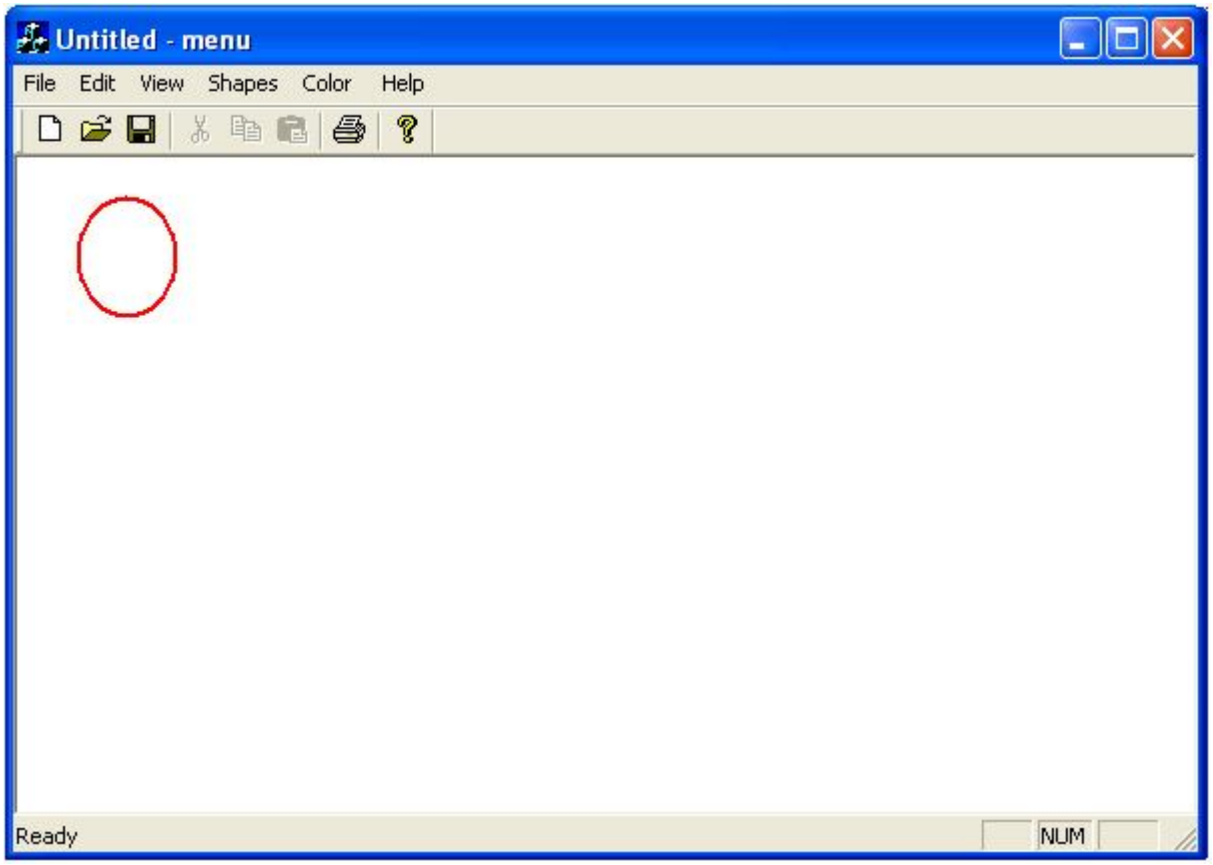
```
CBrush B;

switch(p)
{
case 1:
        P.CreatePen(PS_SOLID,2,RGB(255,0,0));   // for Red color
        break;
case 2:
        P.CreatePen(PS_SOLID,2,RGB(0,255,0)); // for Green color
        break;
case 3:
        P.CreatePen(PS_SOLID,2,RGB(0,0,255)); // for Blue color
        break;
}

switch(b)
{
case 1:
        B.CreateSolidBrush(RGB(255,0,0));  // For Red Color
        break;
case 2:
        B.CreateSolidBrush(RGB(0,255,0)); // For Green Color
        break;
case 3:
        B.CreateSolidBrush(RGB(0,0,255)); // For Blue Color
```

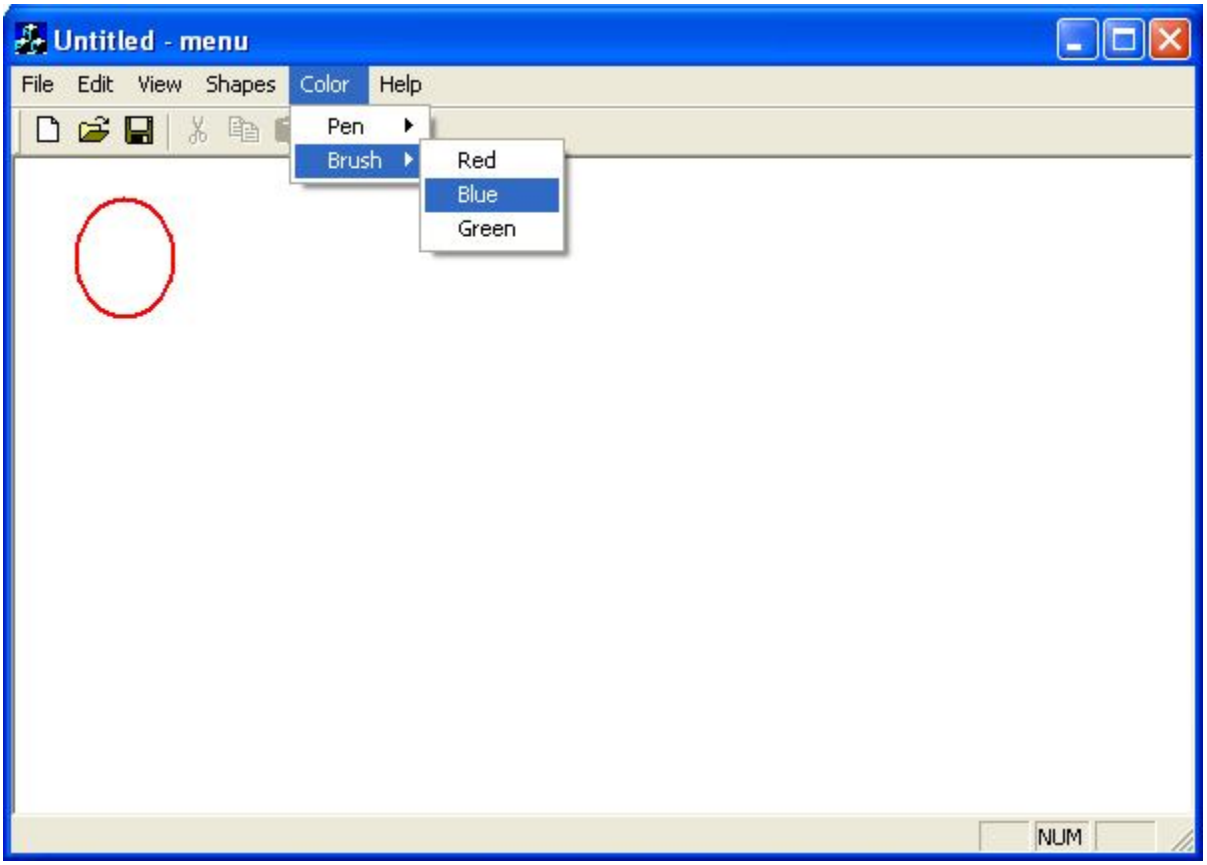
```
            break;
        }
        dc.SelectObject(&P);
        dc.SelectObject(&B);

        switch(s)
        {
        case 1:
            dc.MoveTo(100,100);
            dc.LineTo(200,200);
            break;
        case 2:
            dc.Rectangle(80,80,160,160);
            break;
        case 3:
            dc.Ellipse(30,20,80,80);
            break;
        }
}
```

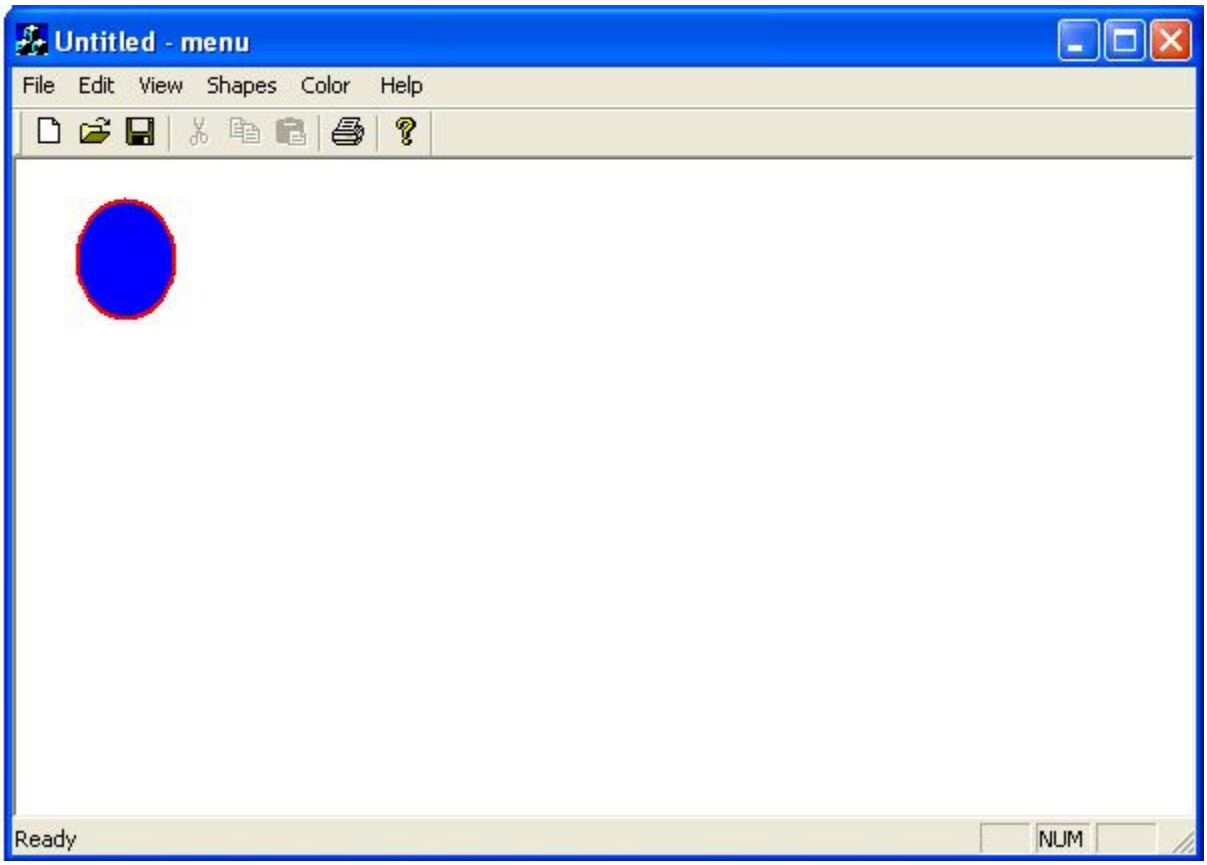
Output:

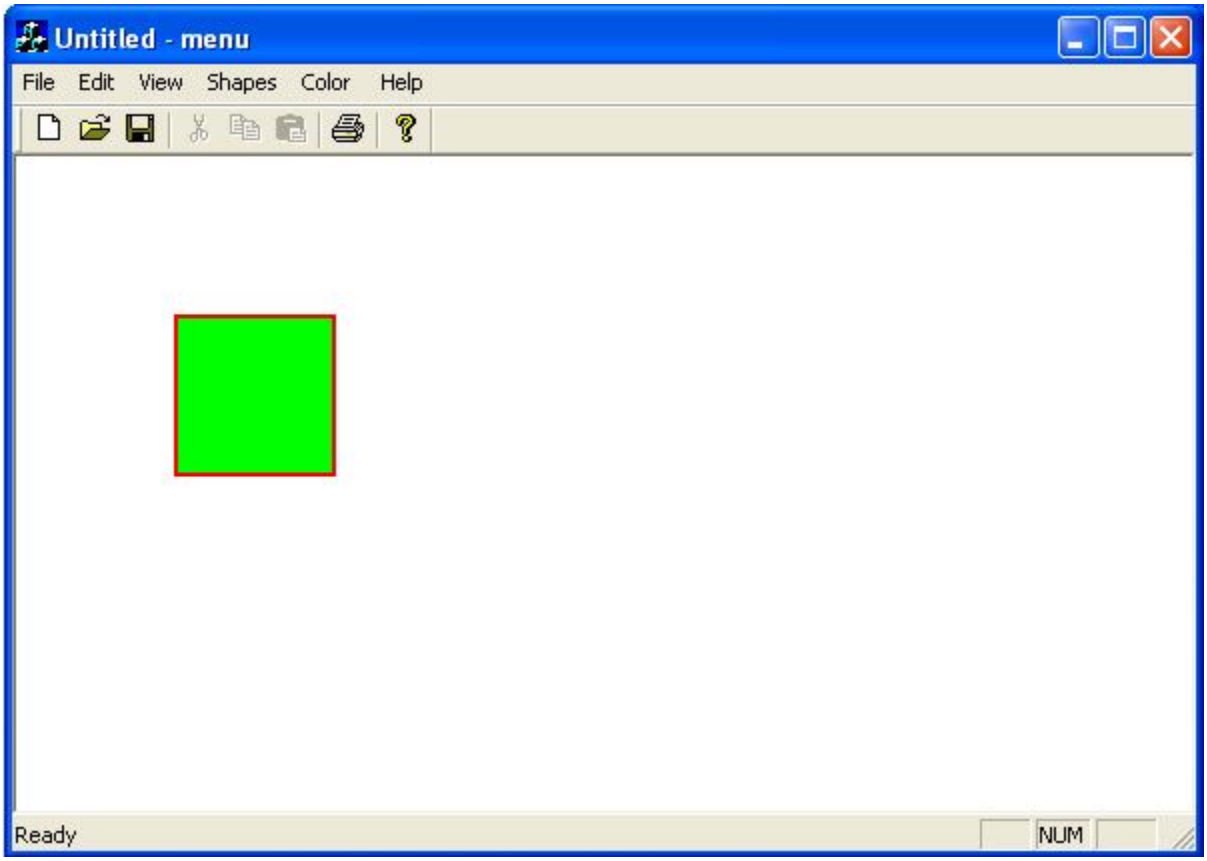

Untitled - menu
File
Edit
View
Shapes
Color
Help
Ellipse Ctrl+E
Line Ctrl+L
Rectangle Ctrl+R
Draws a Ellipse
NUM

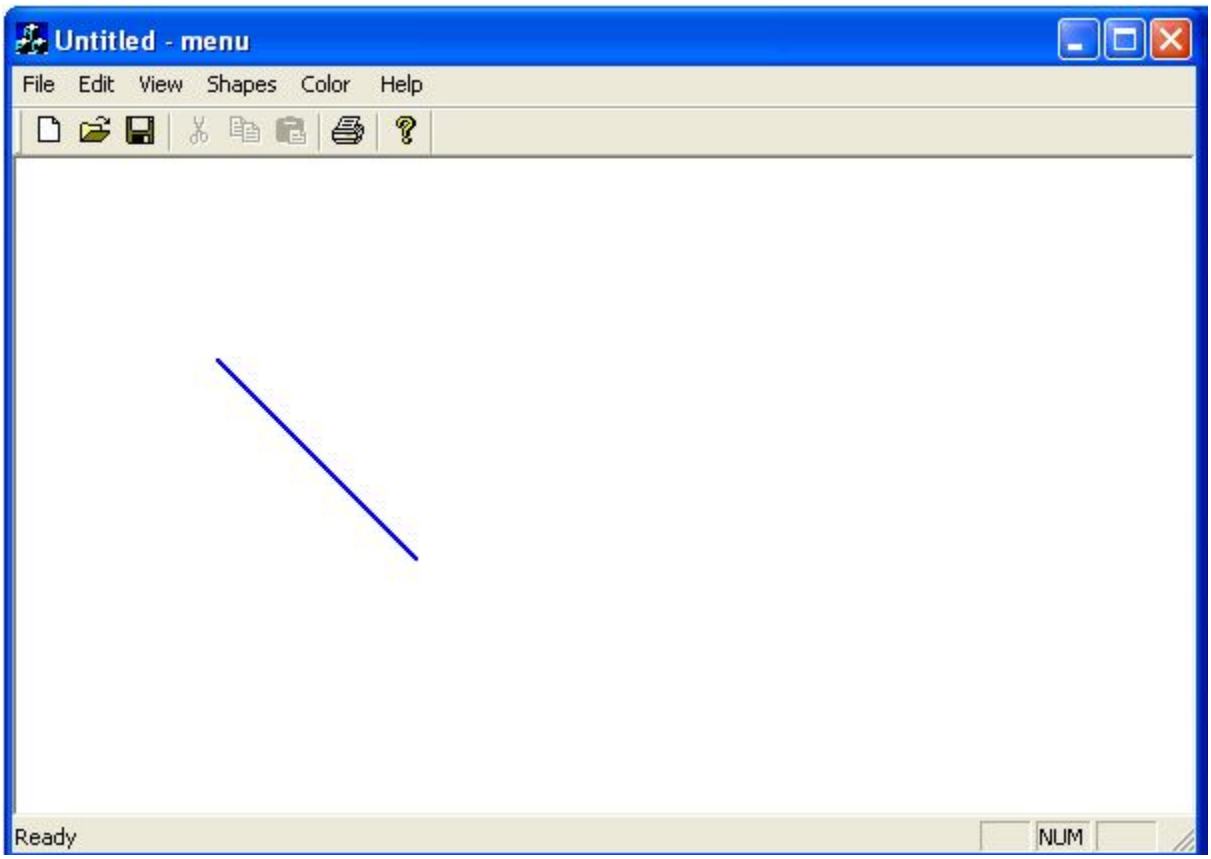

Untitled - menu
File
Edit
View
Shapes
Color
Help
Pen
Brush
Red
Blue
Green
NUM


Untitled - menu
File Edit View Shapes Color Help
Ready
NUM


Untitled - menu
File
Edit
View
Shapes
Color
Help
Pen
Brush
Red
Blue
Green
NUM


Untitled - menu
File Edit View Shapes Color Help
Ready
NUM


Untitled - menu
File Edit View Shapes Color Help
Ready
NUM



Result:

Thus the program to handle menus, views and accelerator has been developed, built and executed using MFC application.

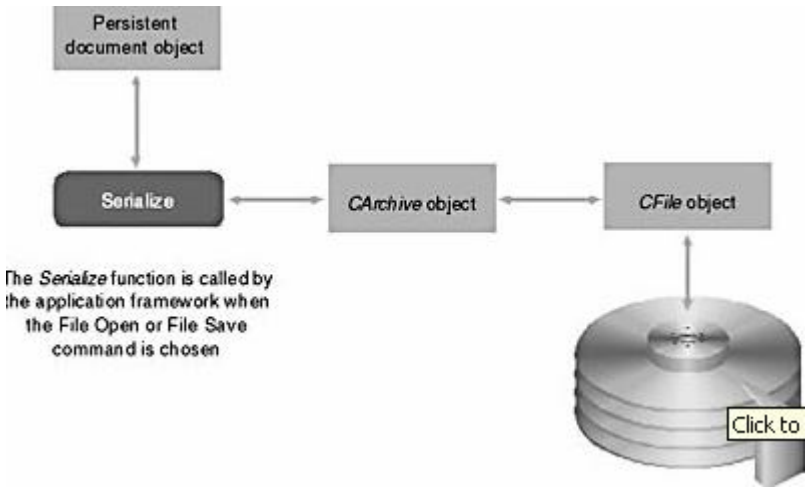
9. SERIALIZATION

Aim:

To write a VC++ program to develop an application and serialize the content inside the document using MFC.

Concept:

The objects can be persistent, which means that they can be saved on disk when a program exits and then can be restored when the program is restarted. This process of saving and restoring objects is called serialization. In the MFC library, designated classes have a member function named Serialize( ). When the application framework calls Serialize( ) for a particular object, then the data for that object is either saved on the disk or read from the disk as shown below

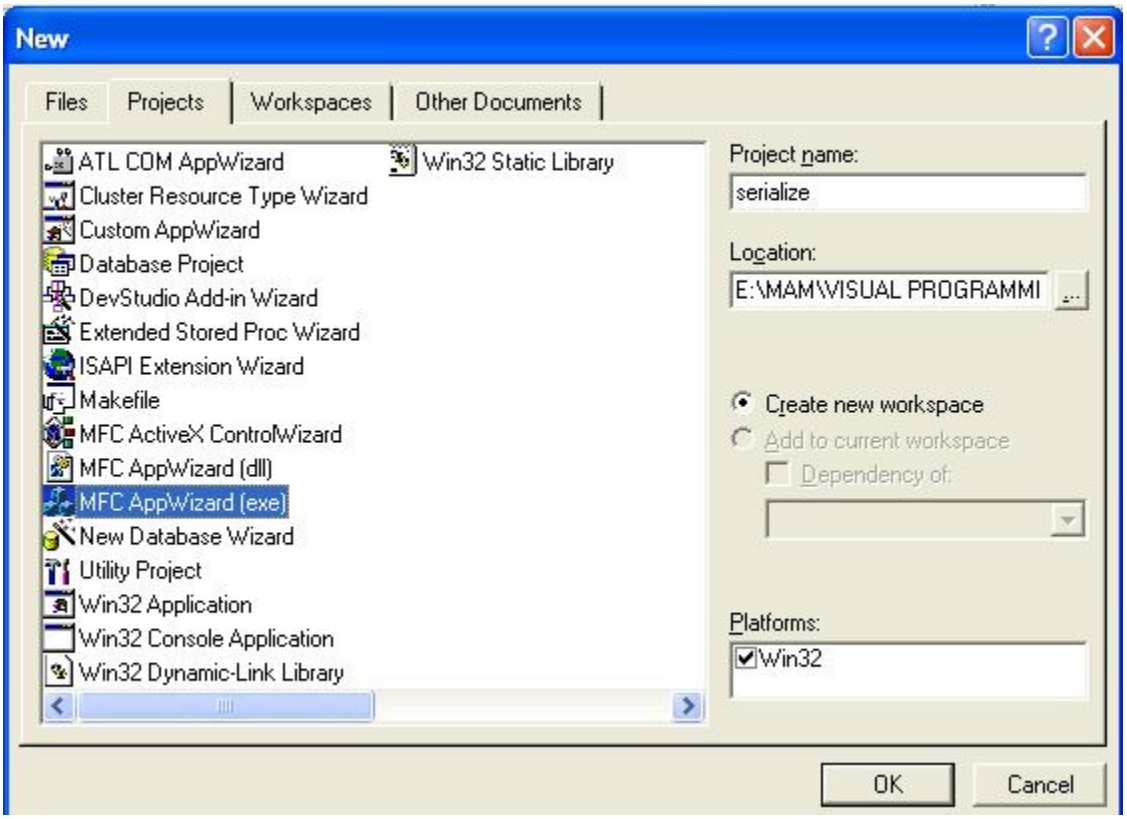


Serialization – Disk Files and Archives

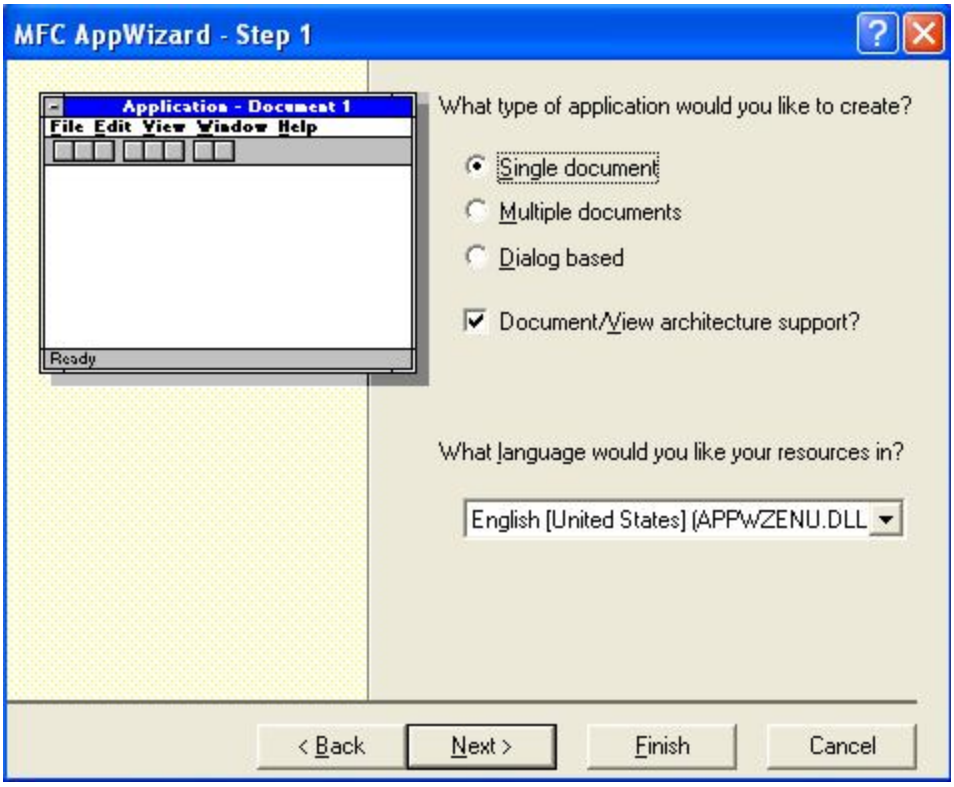
In this program, user can type the text and save it in the system and again open any file and view the content which is serialized inside the document using the above concept.

Procedure:

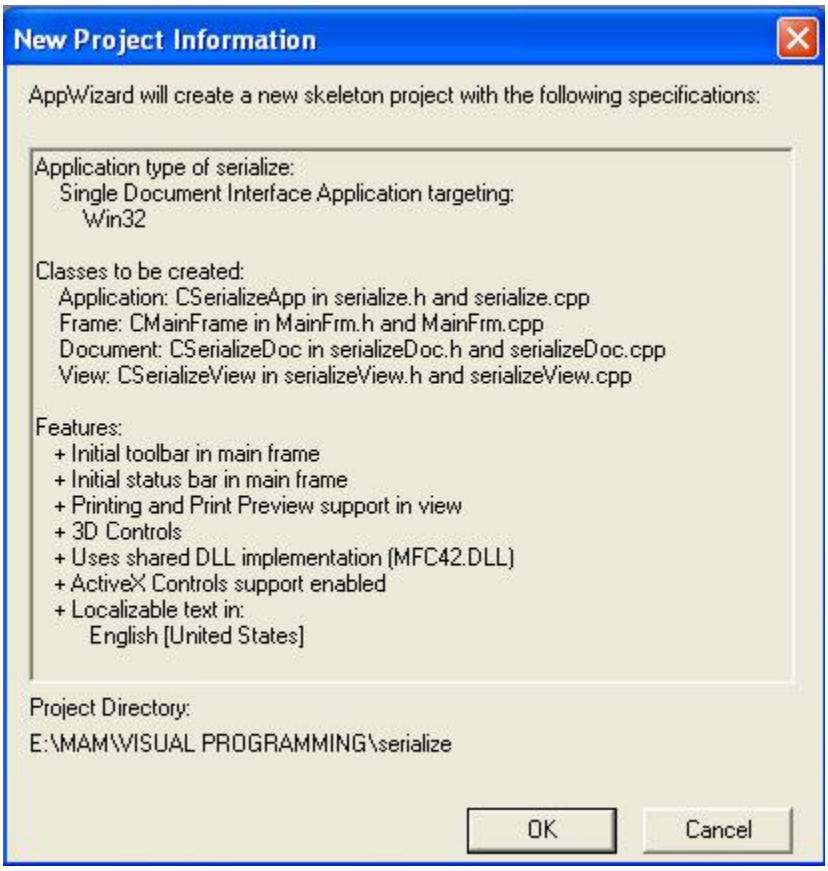
Step 1:



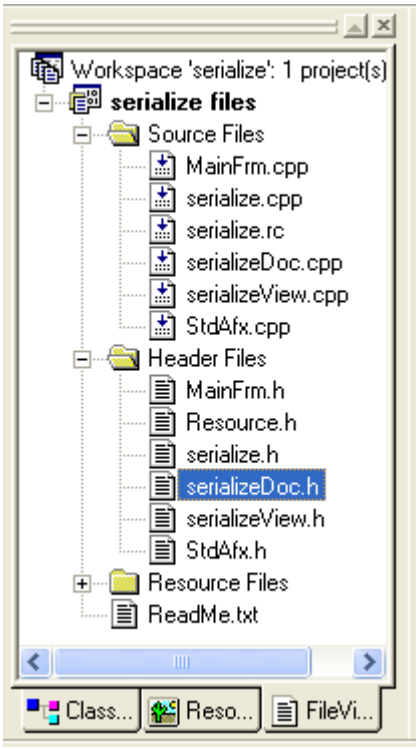
Step 2:


MFC AppWizard - Step 1
Application - Document 1
File Edit View Window Help
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of serialize:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CSerializeApp in serialize.h and serialize.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CSerializeDoc in serializeDoc.h and serializeDoc.cpp
View: CSerializeView in serializeView.h and serializeView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\MAM\VISUAL PROGRAMMING\serialize
OK
Cancel

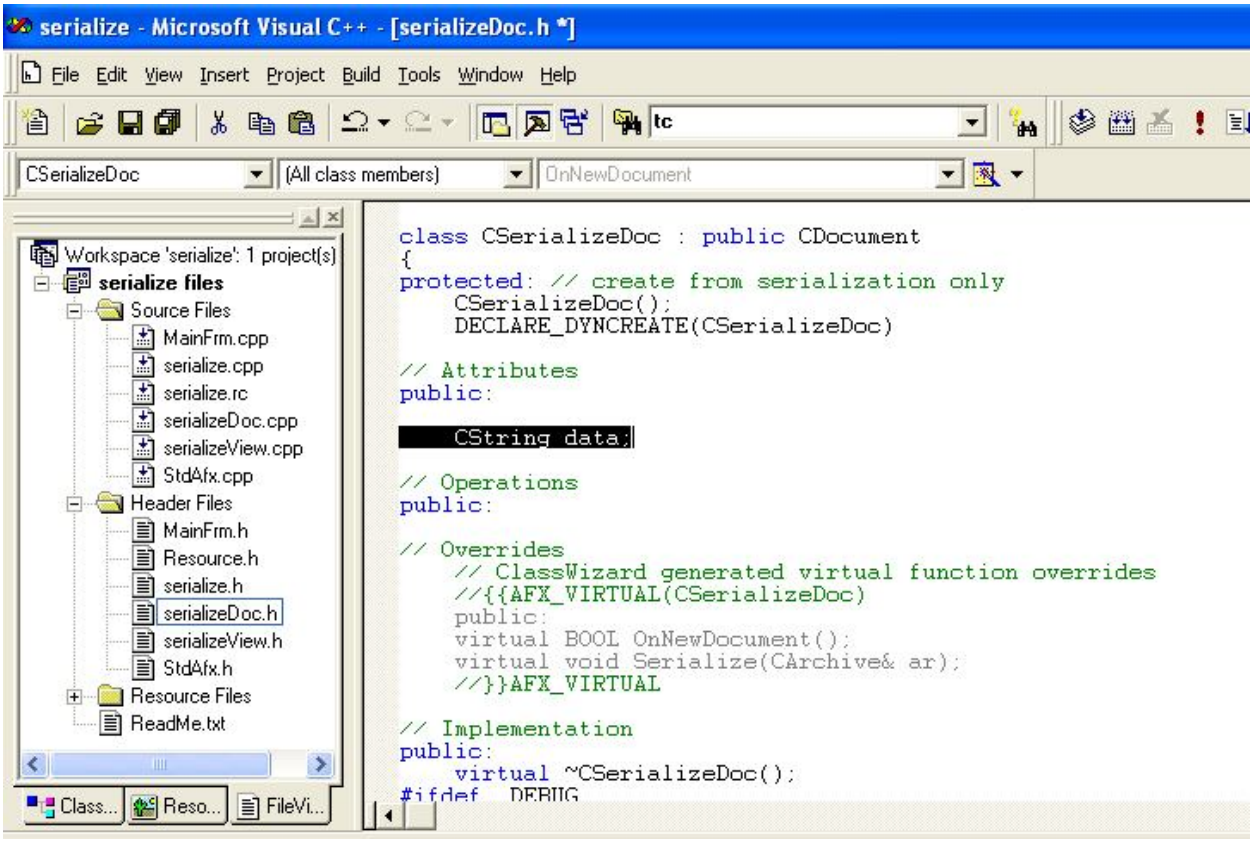
Step 4:



Step 5:

Declare the string data member in **serialize*Doc.h*** file as,

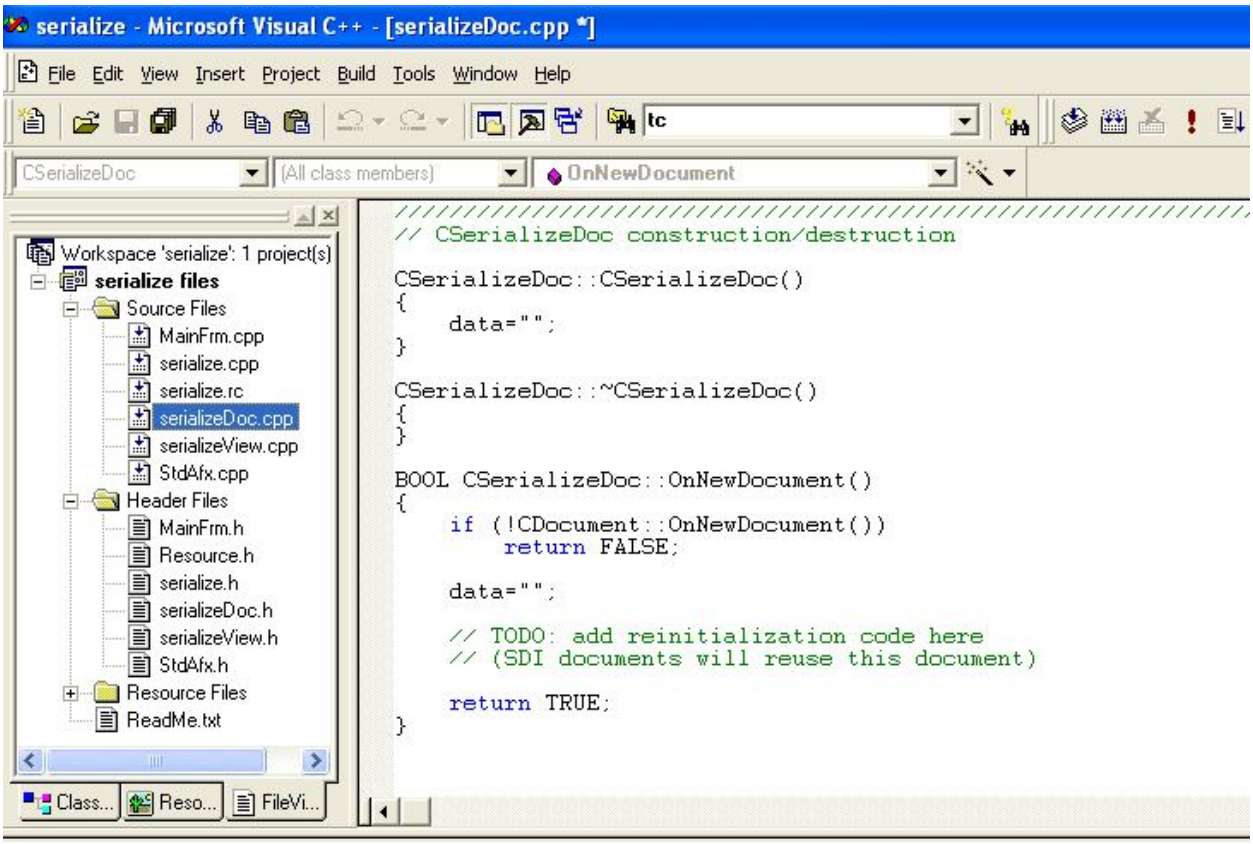
CString data;



Step 6:

Initialize the data member by adding following **boldface** code in **serialize*Doc.cpp*** file in both constructor and onNewDocument function as,

```
CSerializeDoc::CSerializeDoc()
{
        data="";
}

BOOL CSerializeDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        data="";
        return TRUE;
}
```



Step 7:

Edit the Serialize() function to add following **boldface** code in ***serializeDoc.cpp*** as

```
void CSerializeDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                ar<<data;
        }
        else
        {
                ar>>data;
        }
}
```

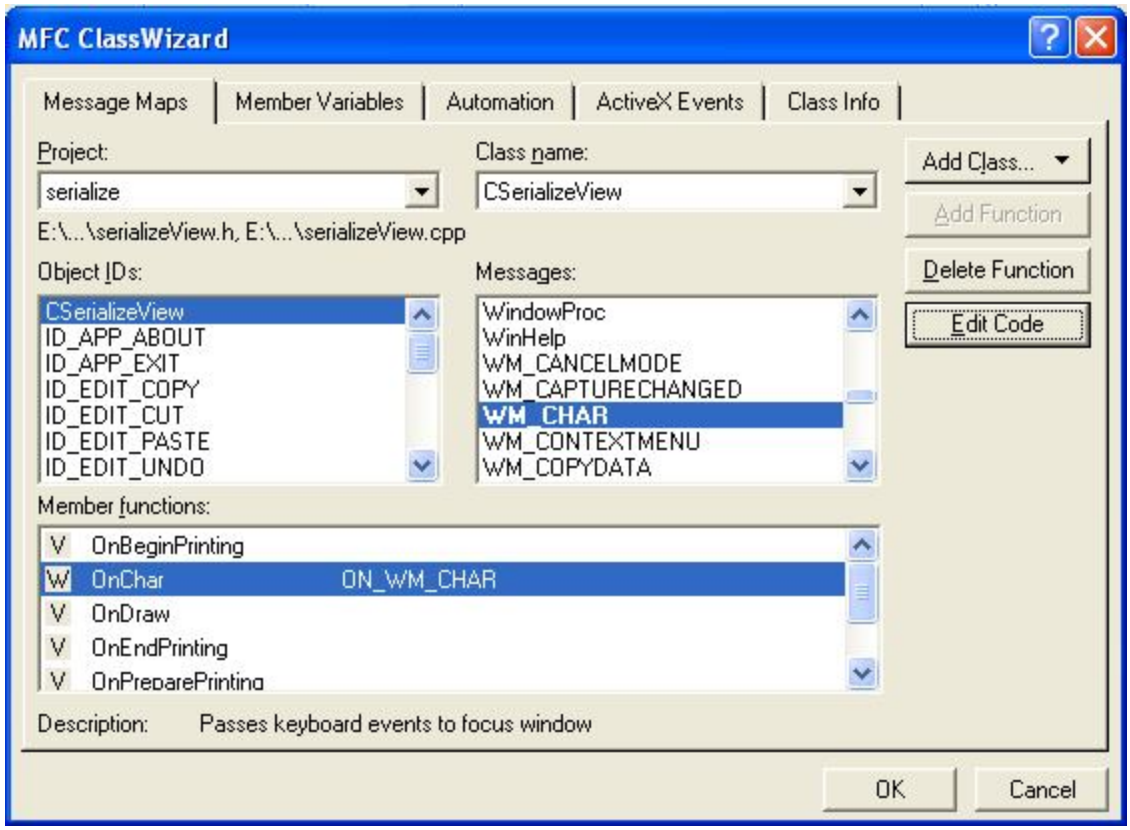
Step 8:

Edit the **OnDraw( )** in **serialize*View.cpp*** file as to add following **boldface** code,

```
void CSerializeView::OnDraw(CDC* pDC)
{
        CSerializeDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDC->TextOut(0,0,pDoc->data);

}
```

Step 9:

Use classwizard to connect the **WM_CHAR** message to ***CserializeView*** class



Step 10:

Edit the OnChar**( )** message handler in ***serializeView.cpp*** file as to add following **boldface** code,

```
void CSerializeView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        CSerializeDoc* pDoc=GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->data +=nChar;
```

```
        Invalidate();
        pDoc->SetModifiedFlag();

        CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Step 11:

Build the program and execute it

Program:

// serializeDoc.h

```
class CSerializeDoc : public CDocument
{
protected: // create from serialization only
        CSerializeDoc();
        DECLARE_DYNCREATE(CSerializeDoc)

// Attributes
public:

        CString data;
```

// serializeDoc.cpp

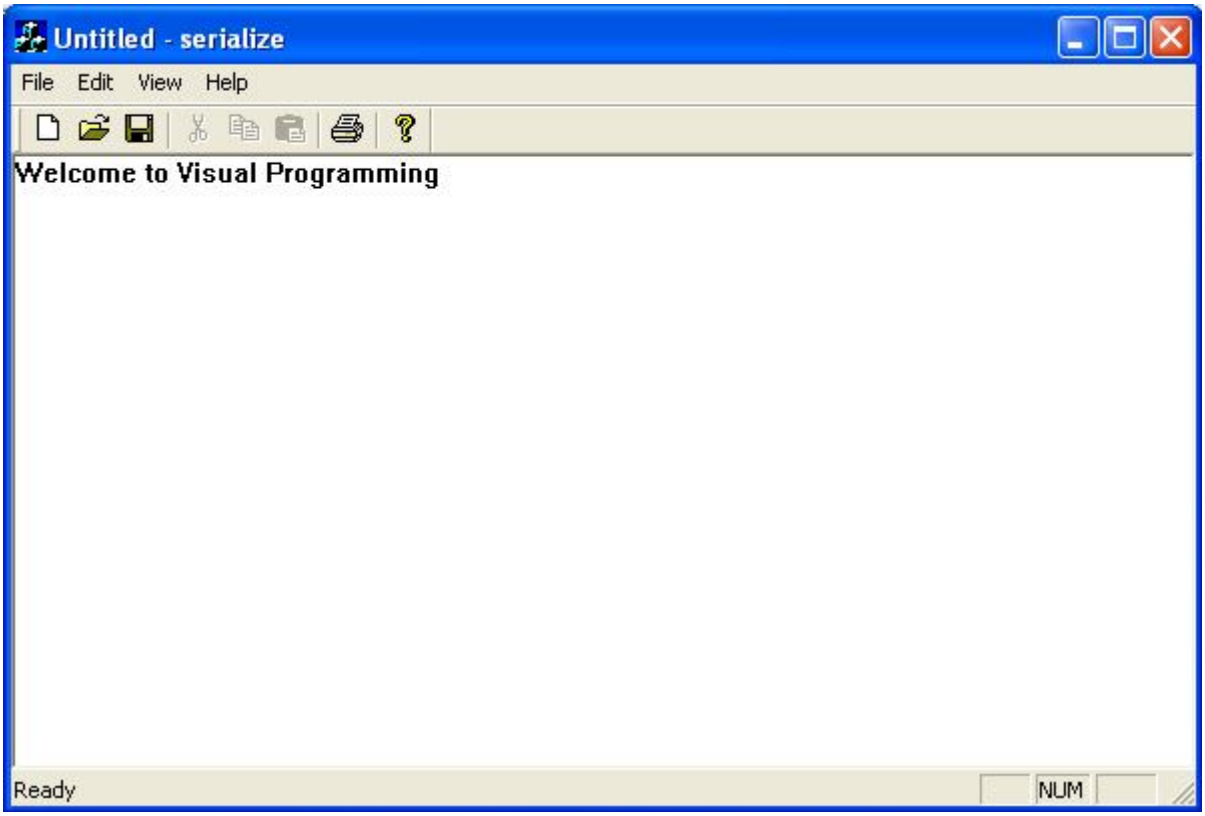
```
// CSerializeDoc construction/destruction

CSerializeDoc::CSerializeDoc()
{
        data="";
}


BOOL CSerializeDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        data="";
```

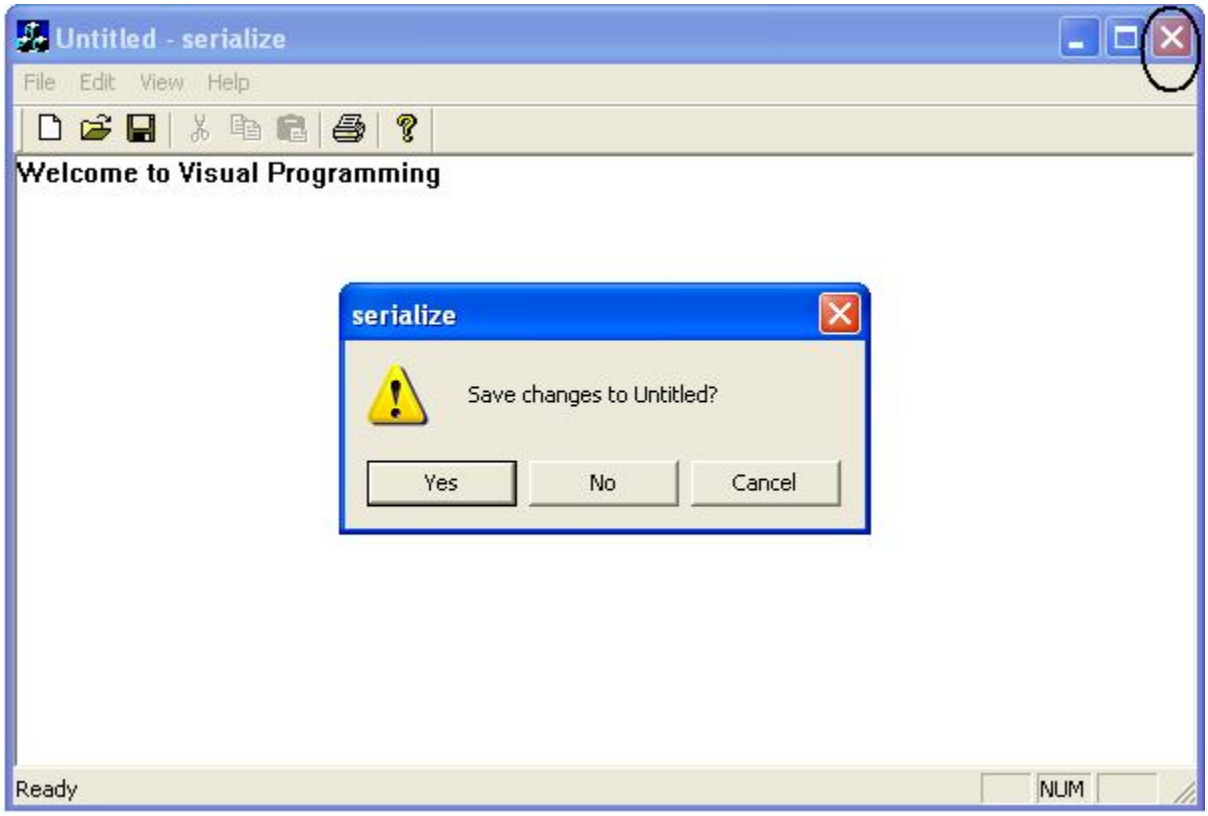
```
	return TRUE;
}


// CSerializeDoc serialization

void CSerializeDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		ar<<data;
	}
	else
	{
		ar>>data;
	}
}
```

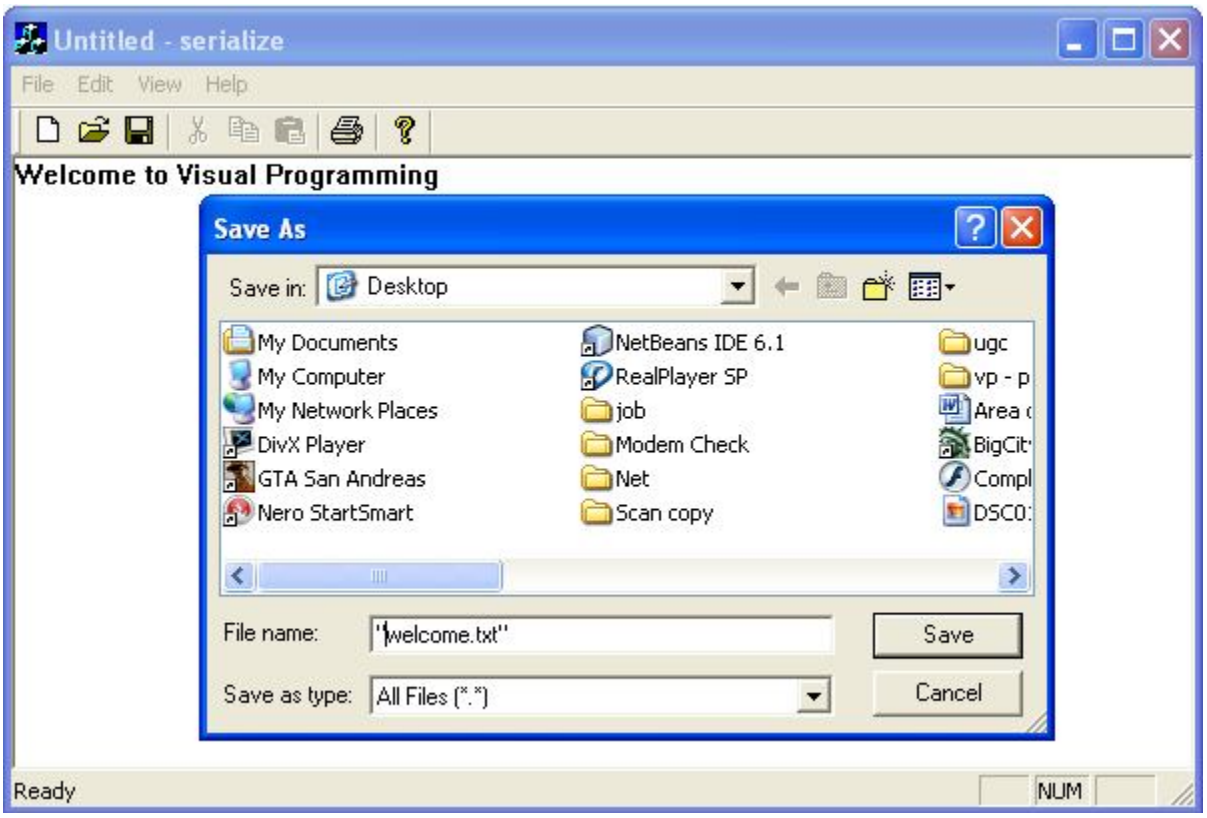
// serializeView.cpp

```
// CSerializeView drawing

void CSerializeView::OnDraw(CDC* pDC)
{
	CSerializeDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	pDC->TextOut(0,0,pDoc->data);

}

// CSerializeView message handlers

void CSerializeView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	CSerializeDoc* pDoc=GetDocument();
	ASSERT_VALID(pDoc);
	pDoc->data +=nChar;
	Invalidate();
	pDoc->SetModifiedFlag();

	CView::OnChar(nChar, nRepCnt, nFlags);
}
```

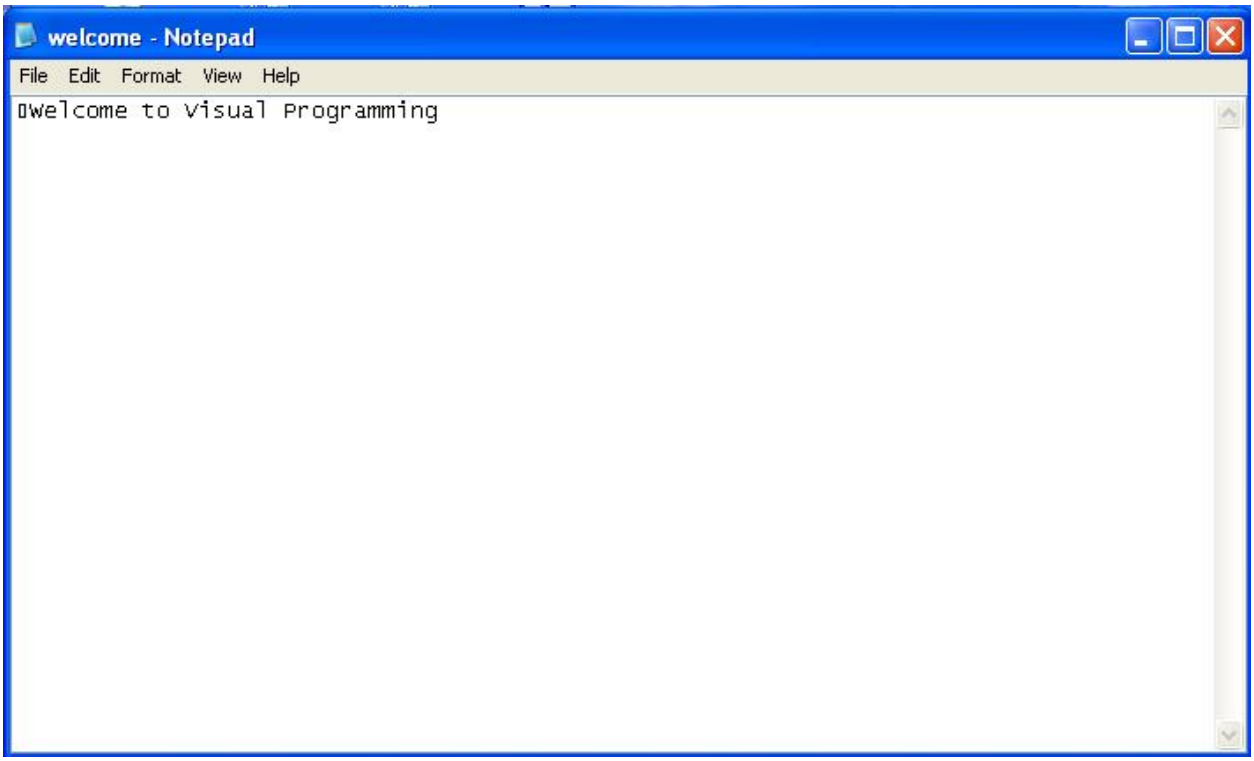
Output:


Untitled - serialize
File Edit View Help
Welcome to Visual Programming
Ready
NUM


Untitled - serialize
File Edit View Help
Welcome to Visual Programming
serialize
Save changes to Untitled?
Yes
No
Cancel
Ready
NUM



Save the document as “Welcome.txt” in some folder and open the same file as below and view the content which is serialized inside the document.



Result:

Thus the VC++ program to serialize the content inside the document has been developed, builds and verified.

10. ACTIVEX CONTROL

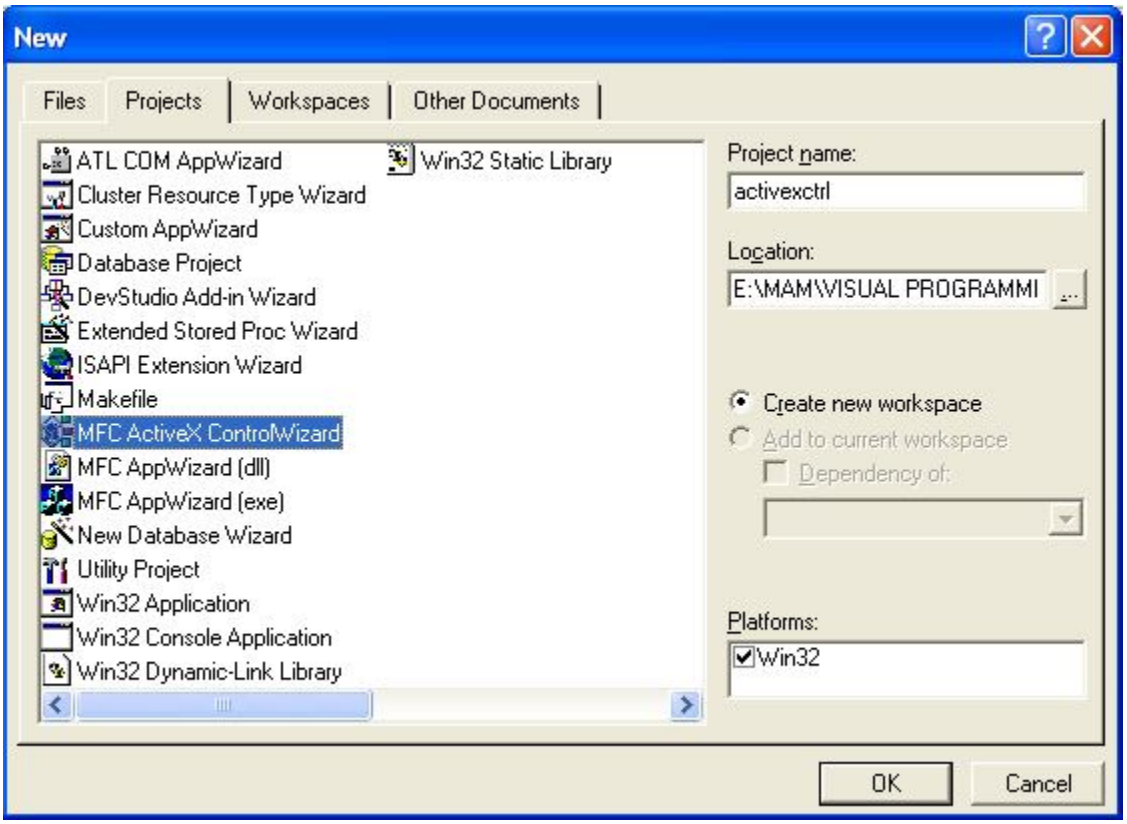
Aim:

To write a VC++ program to build an application that uses ActiveX control

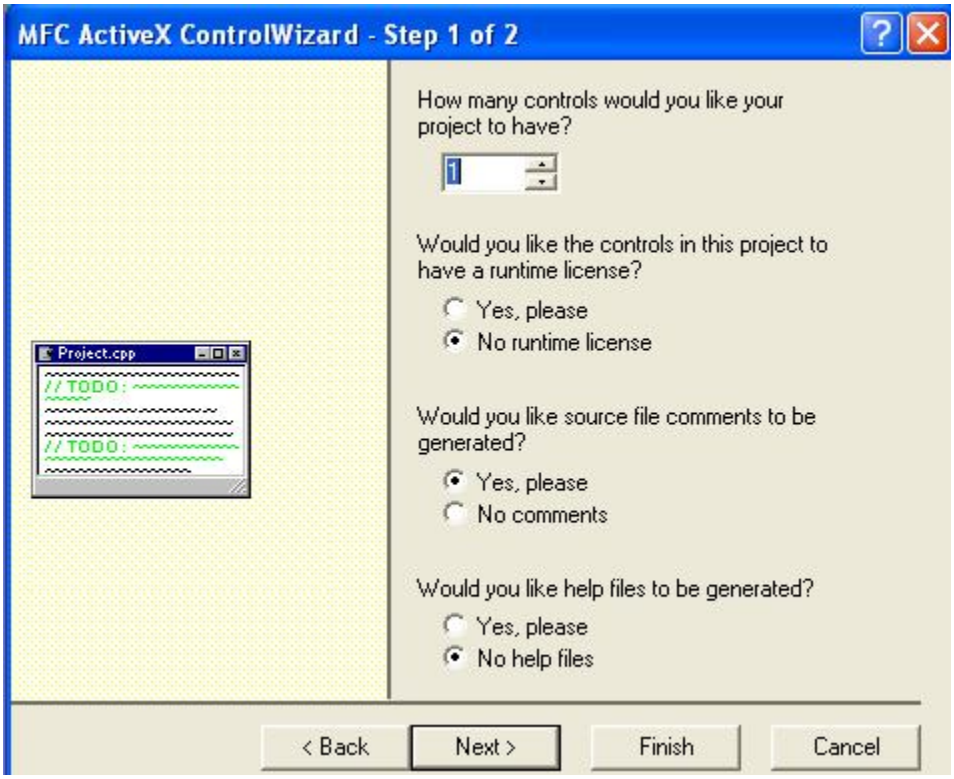
Concept:

Procedure:

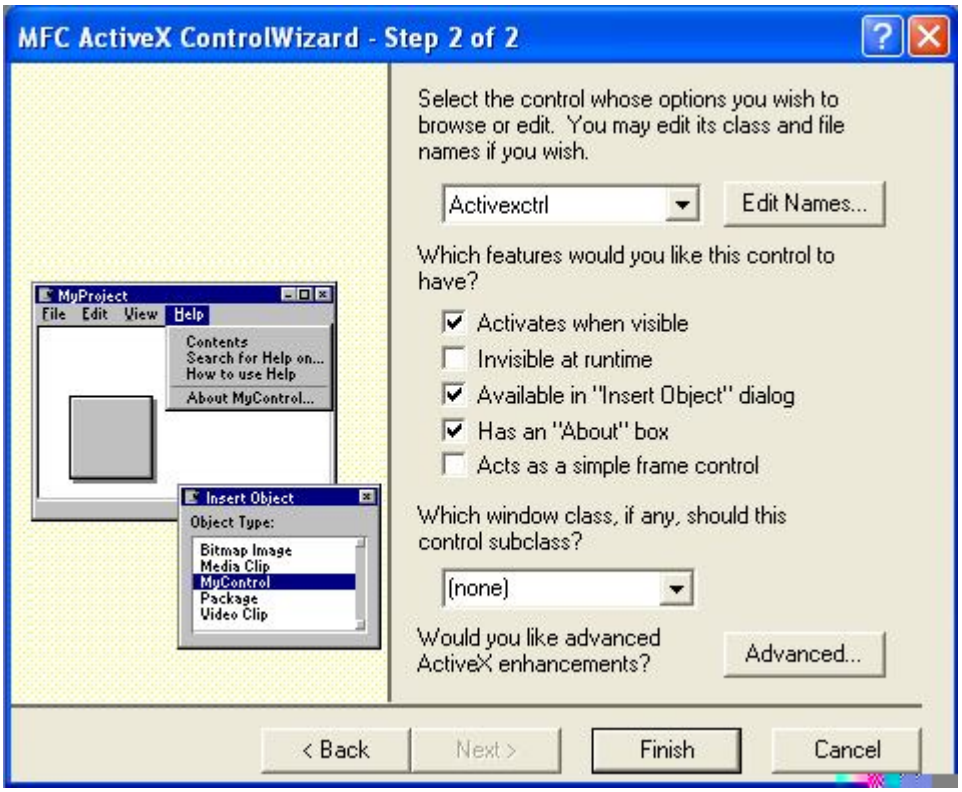
Step 1:



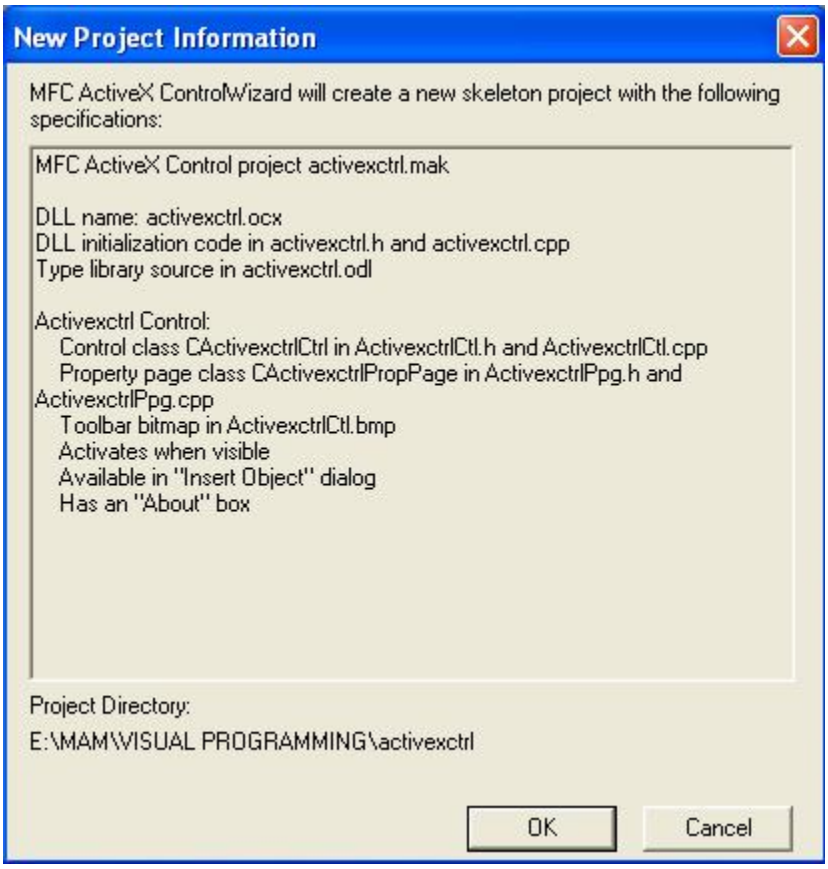
Step 2:


MFC ActiveX ControlWizard - Step 1 of 2
How many controls would you like your project to have?
1
Would you like the controls in this project to have a runtime license?
Yes, please
No runtime license
Would you like source file comments to be generated?
Yes, please
No comments
Would you like help files to be generated?
Yes, please
No help files
Project.cpp
// TODO:
// TODO:
< Back
Next >
Finish
Cancel

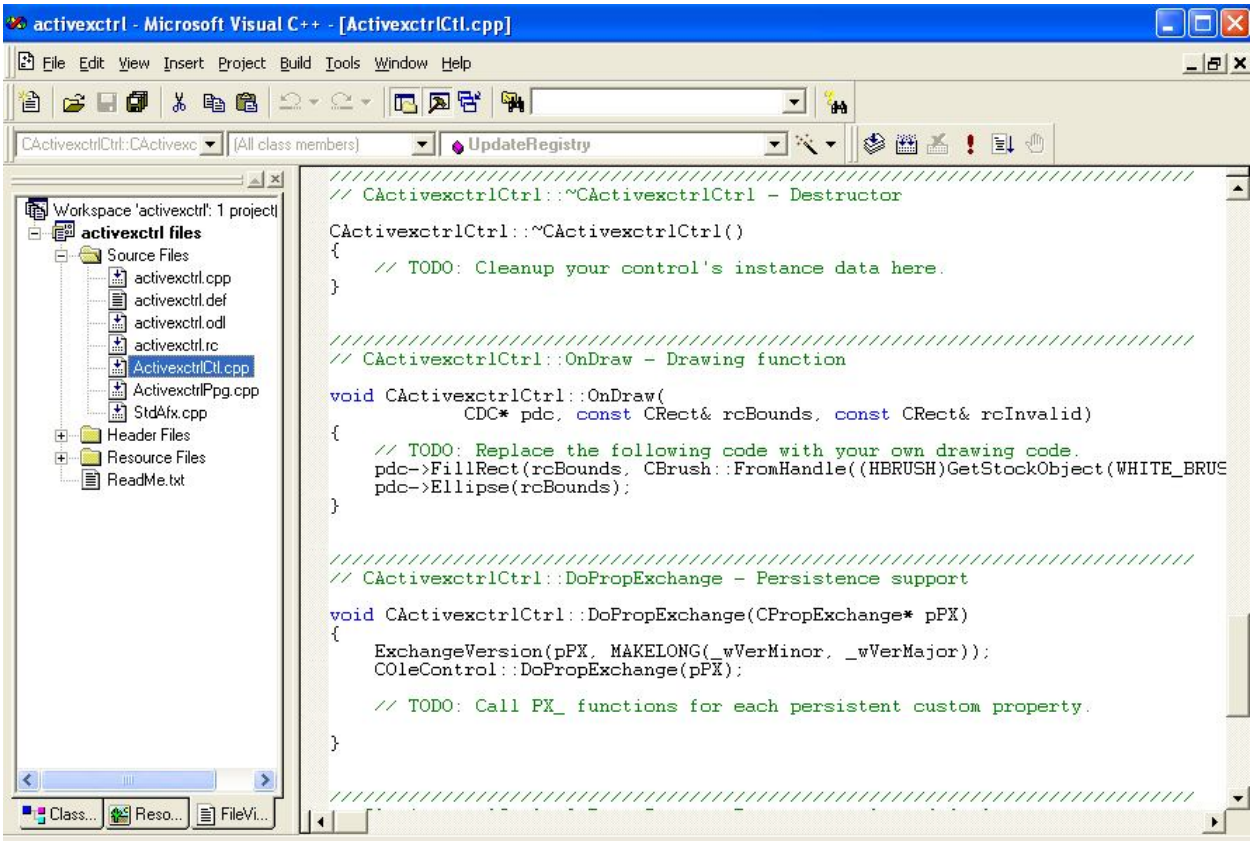
Step 3:


MFC ActiveX ControlWizard - Step 2 of 2
Select the control whose options you wish to browse or edit. You may edit its class and file names if you wish.
Activexctrl
Edit Names...
Which features would you like this control to have?
Activates when visible
Invisible at runtime
Available in "Insert Object" dialog
Has an "About" box
Acts as a simple frame control
Which window class, if any, should this control subclass?
(none)
Would you like advanced ActiveX enhancements?
Advanced...
MyProject
File Edit View Help
Contents
Search for Help on...
How to use Help
About MyControl...
Insert Object
Object Type:
Bitmap Image
Media Clip
MyControl
Package
Video Clip
< Back
Next >
Finish
Cancel

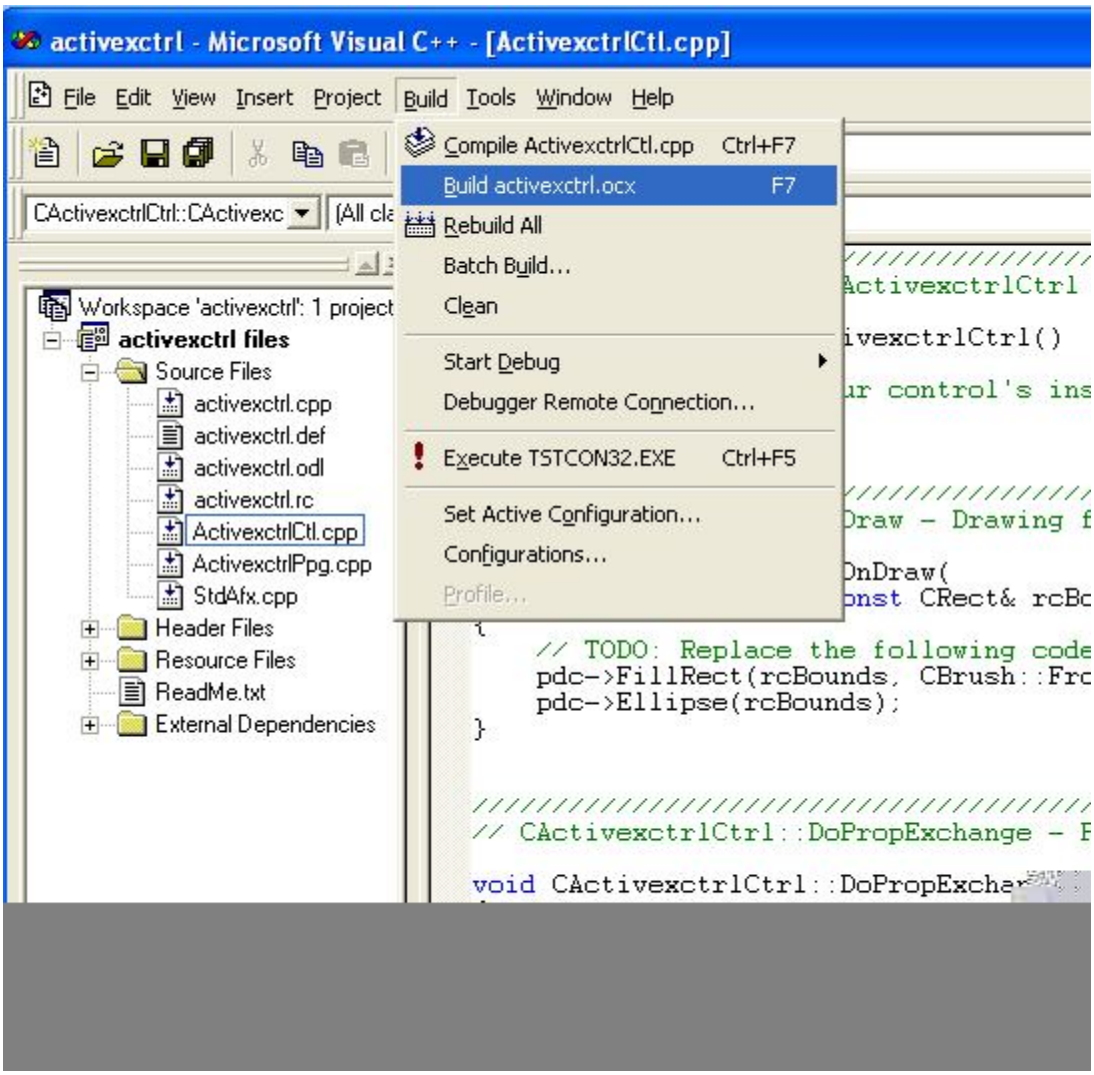
Step 4:


New Project Information
MFC ActiveX ControlWizard will create a new skeleton project with the following specifications:
MFC ActiveX Control project activexctrl.mak
DLL name: activexctrl.ocx
DLL initialization code in activexctrl.h and activexctrl.cpp
Type library source in activexctrl.odl
Activexctrl Control:
Control class CActivexctrlCtrl in ActivexctrlCtl.h and ActivexctrlCtl.cpp
Property page class CActivexctrlPropPage in ActivexctrlPpg.h and ActivexctrlPpg.cpp
Toolbar bitmap in ActivexctrlCtl.bmp
Activates when visible
Available in "Insert Object" dialog
Has an "About" box
Project Directory:
E:\MAM\VISUAL PROGRAMMING\activexctrl
OK
Cancel

Step 5:


activexctrl - Microsoft Visual C++ - [ActivexctrlCtl.cpp]
File Edit View Insert Project Build Tools Window Help
UpdateRegistry
Workspace 'activexctrl': 1 project
activexctrl files
Source Files
activexctrl.cpp
activexctrl.def
activexctrl.odl
activexctrl.rc
ActivexctrlCtl.cpp
ActivexctrlPpg.cpp
StdAfx.cpp
Header Files
Resource Files
ReadMe.txt
Class...
Reso...
FileVi...
// CActivexctrlCtrl::~CActivexctrlCtrl - Destructor
CActivexctrlCtrl::~CActivexctrlCtrl()
{
// TODO: Cleanup your control's instance data here.
}
// CActivexctrlCtrl::OnDraw - Drawing function
void CActivexctrlCtrl::OnDraw(
CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
// TODO: Replace the following code with your own drawing code.
pdc->FillRect(rcBounds, CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUS
pdc->Ellipse(rcBounds);
}
// CActivexctrlCtrl::DoPropExchange - Persistence support
void CActivexctrlCtrl::DoPropExchange(CPropExchange* pPX)
{
ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
COleControl::DoPropExchange(pPX);
// TODO: Call PX_ functions for each persistent custom property.
}

Step 6:



Program:

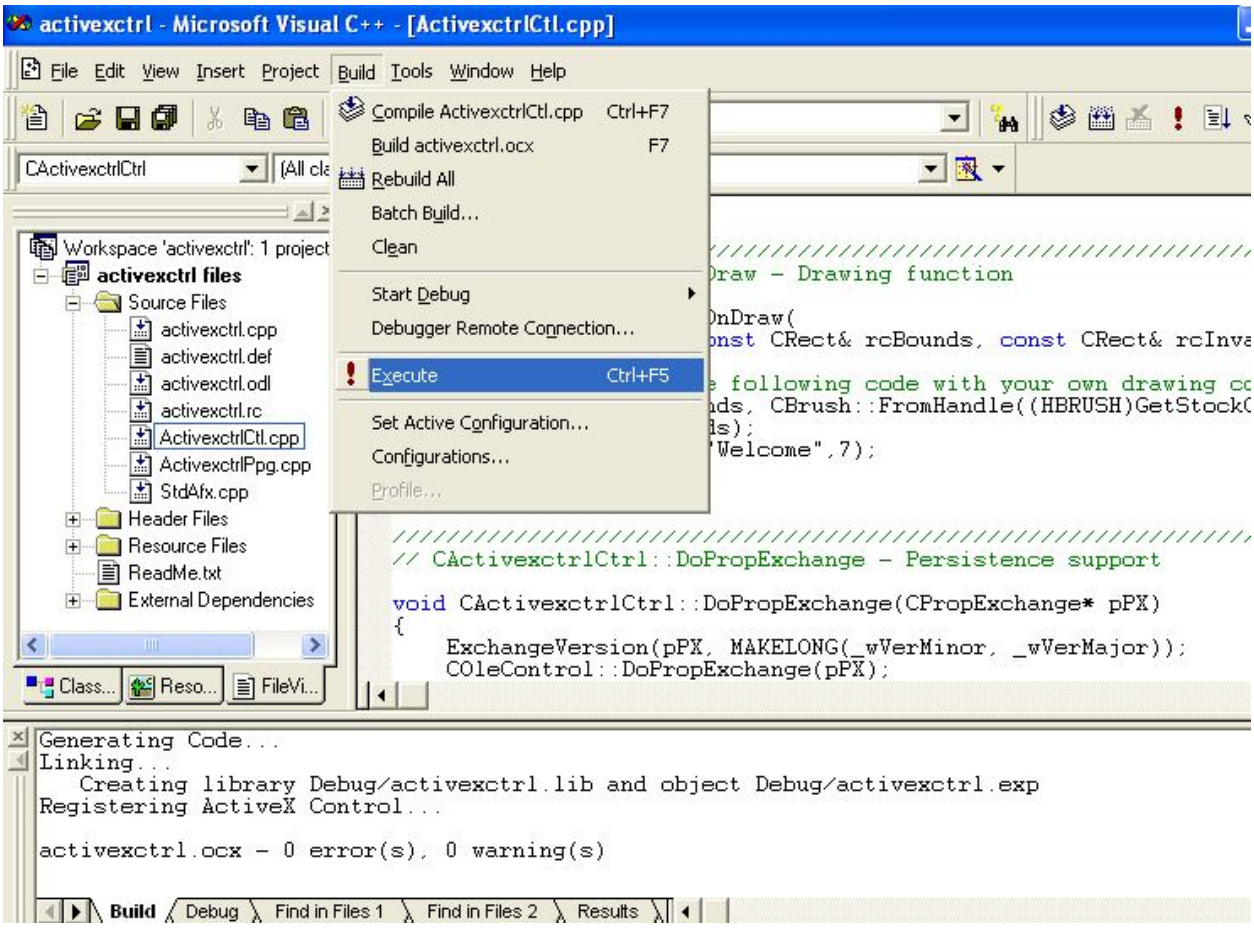
// ActivexctrlCtl.cpp

// CActivexctrlCtrl::OnDraw - Drawing function

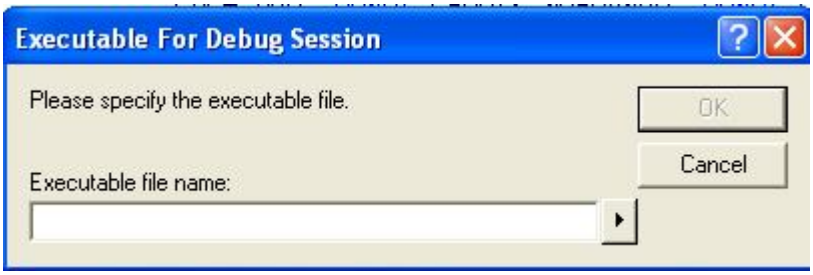
```
void CActivexctrlCtrl::OnDraw(
                    CDC*  pdc, const CRect&  rcBounds, const CRect&  rcInvalid)
{
        pdc->FillRect(rcBounds,
        CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
        pdc->Ellipse(rcBounds);
        pdc->TextOut(20,30,"Welcome",7);
}
```

Execution:

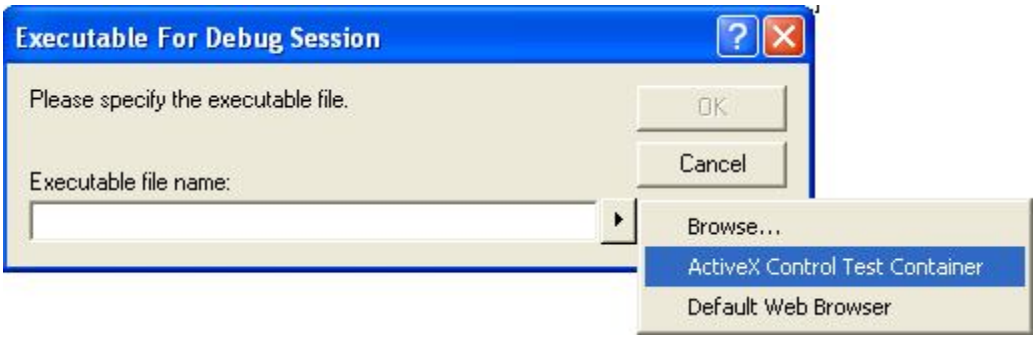
Step 7:



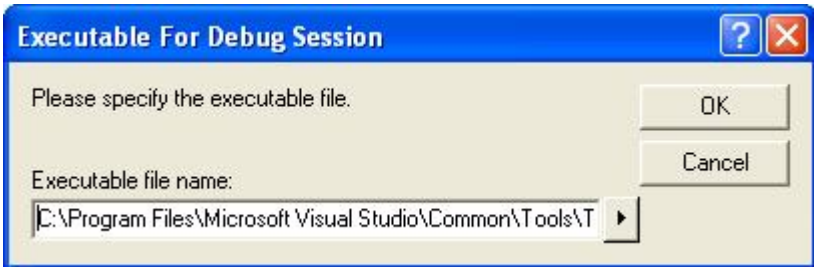
Step 8:



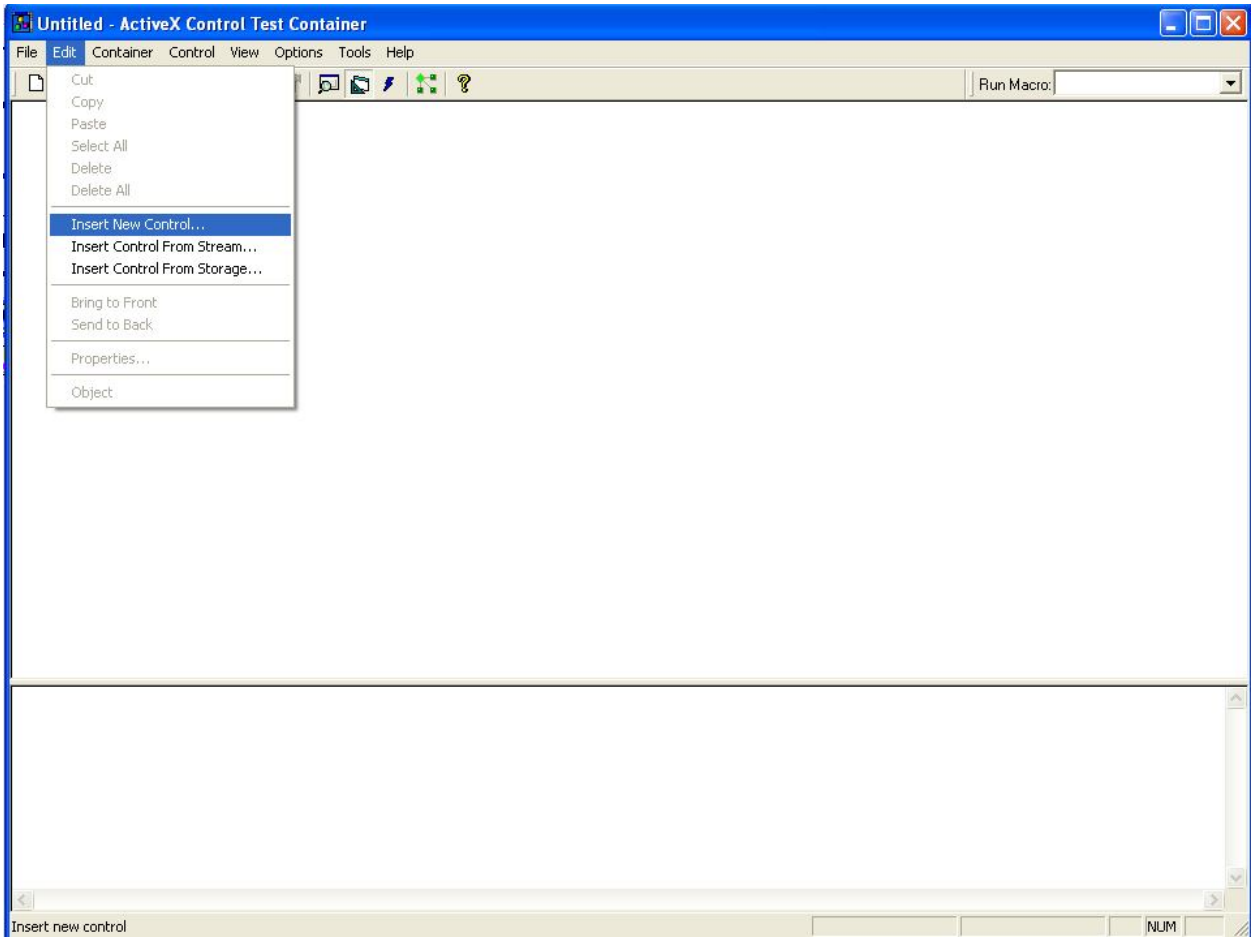
Step 9:



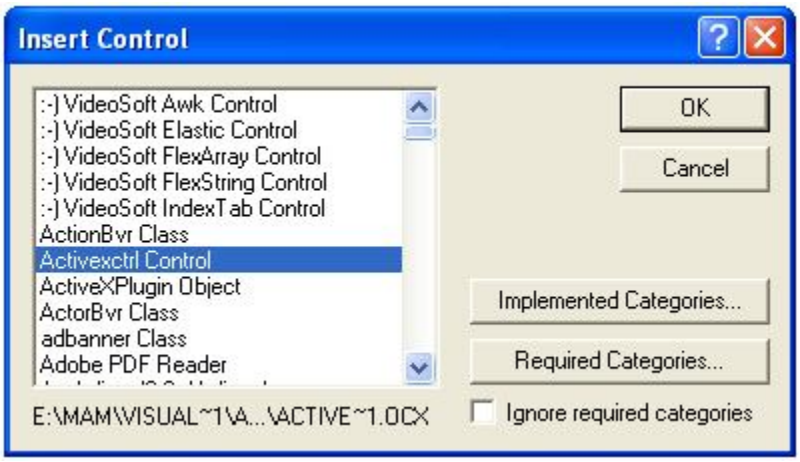
Step 10:



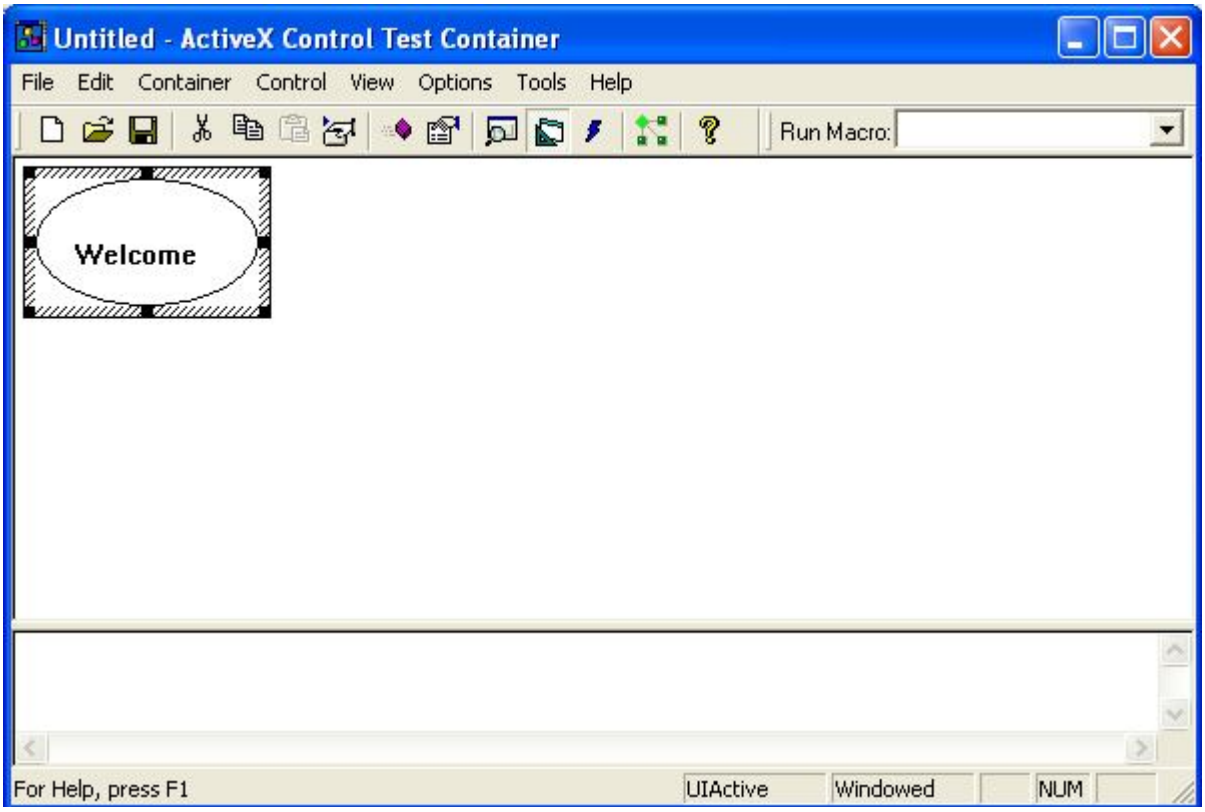
Step 11:



Step 12:

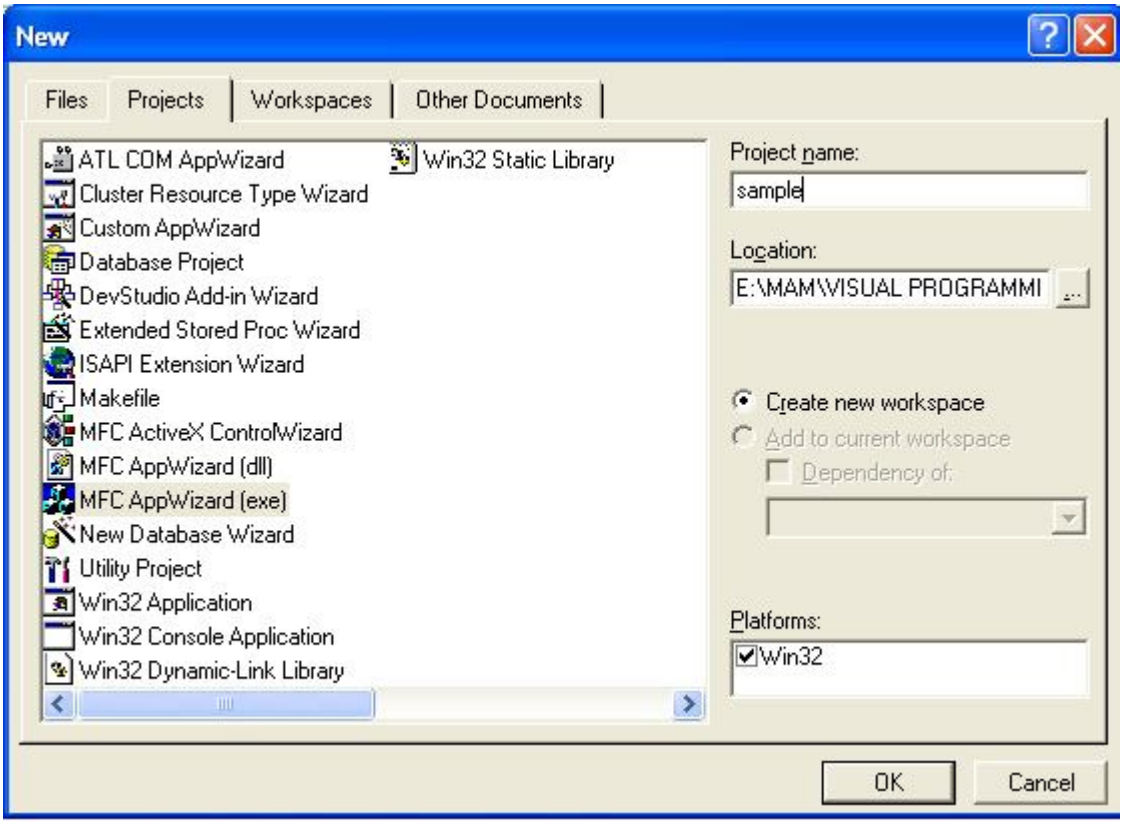


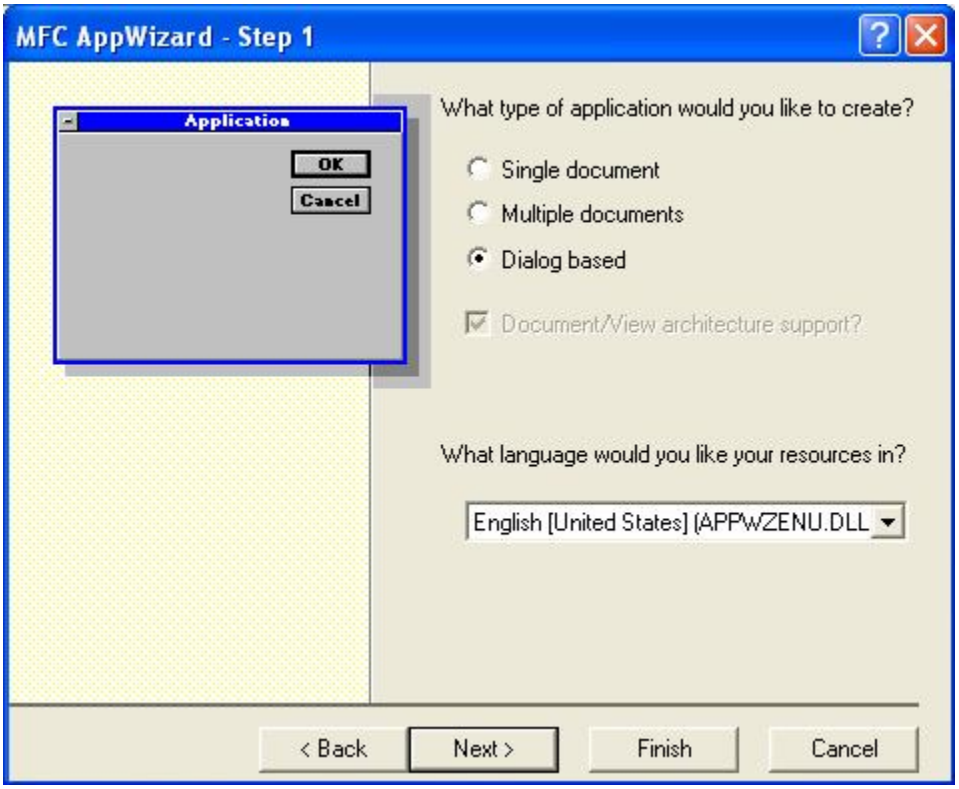
Step 13:

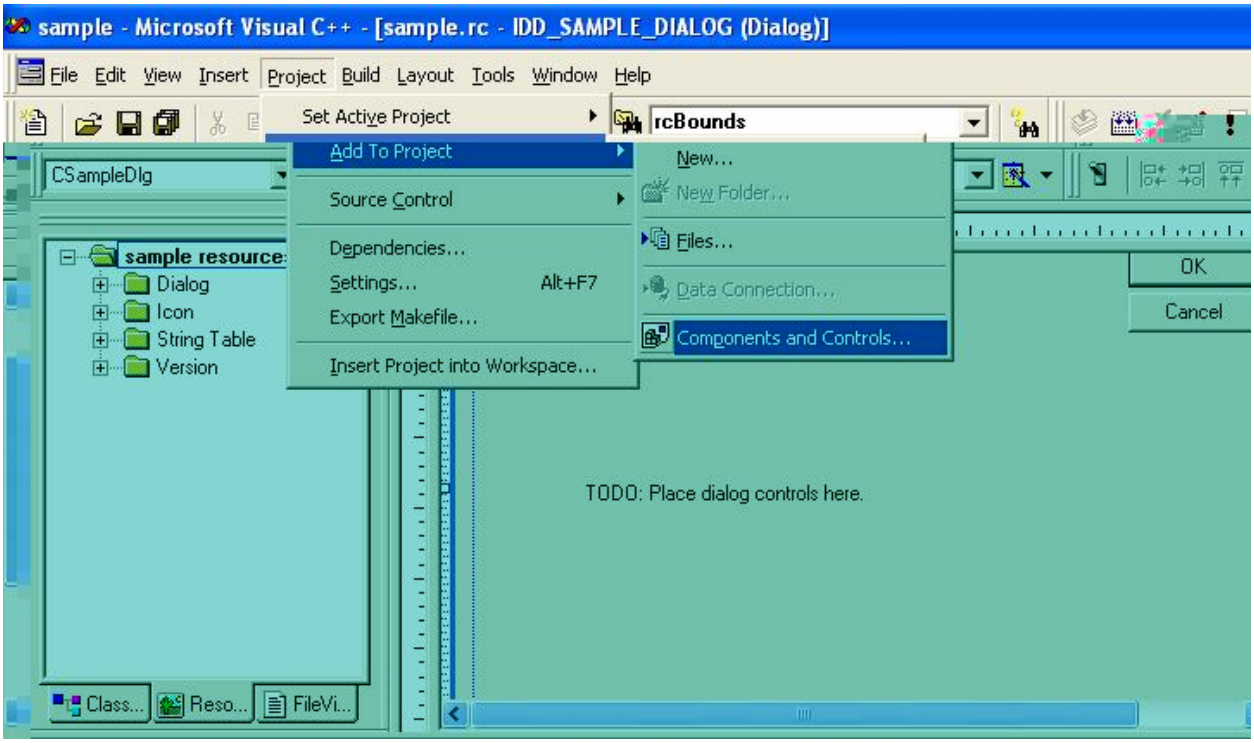


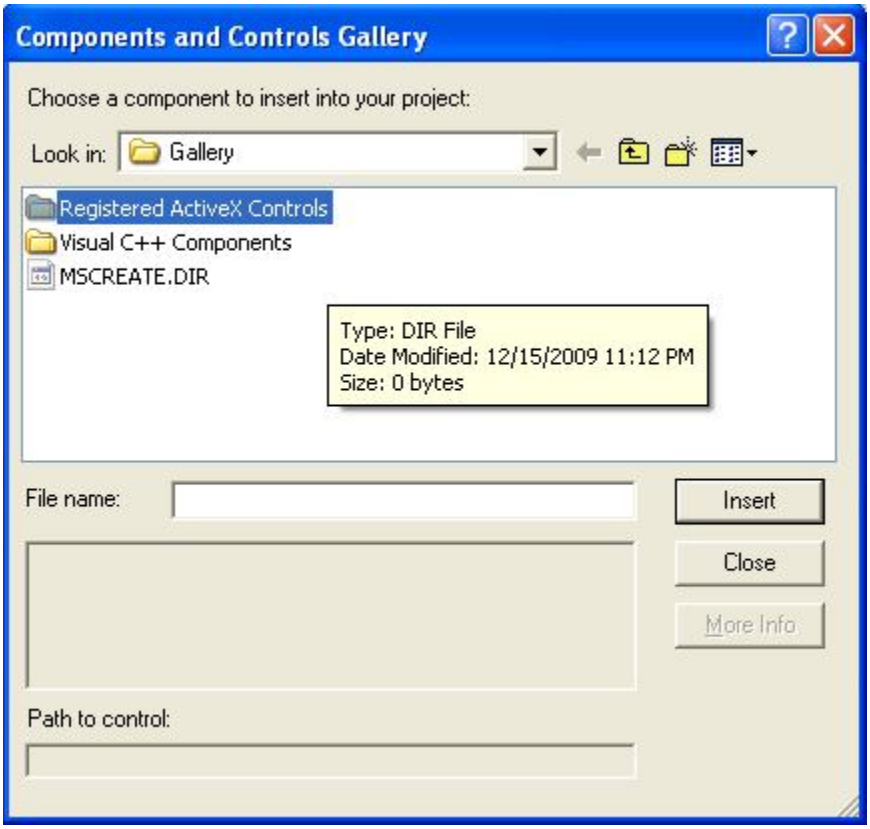
Step 13:

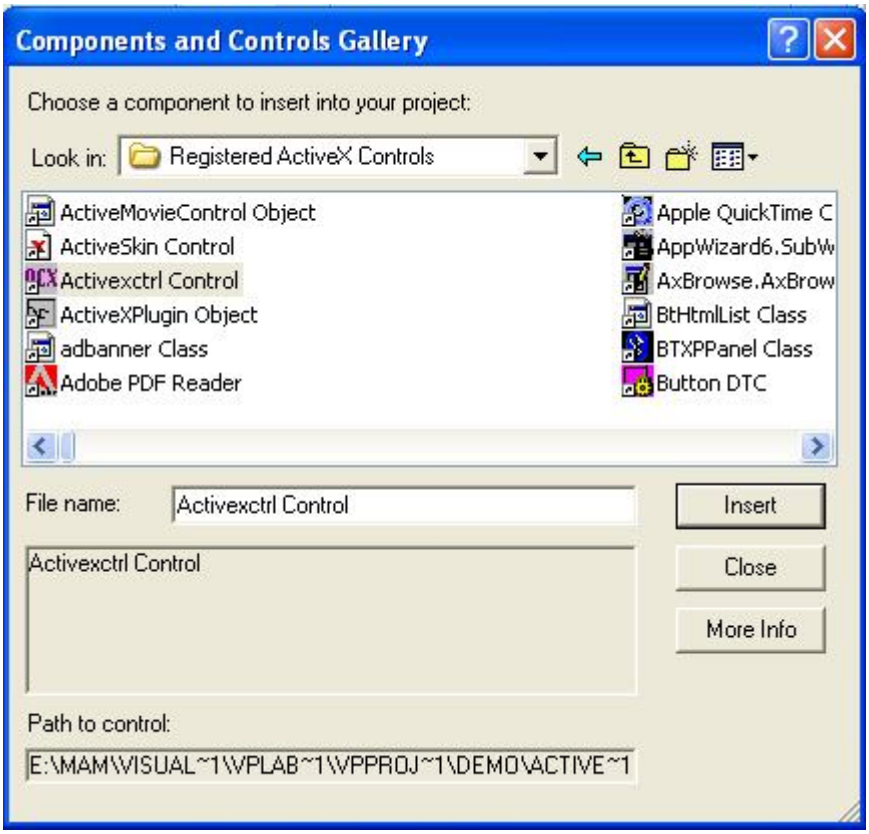
Output:

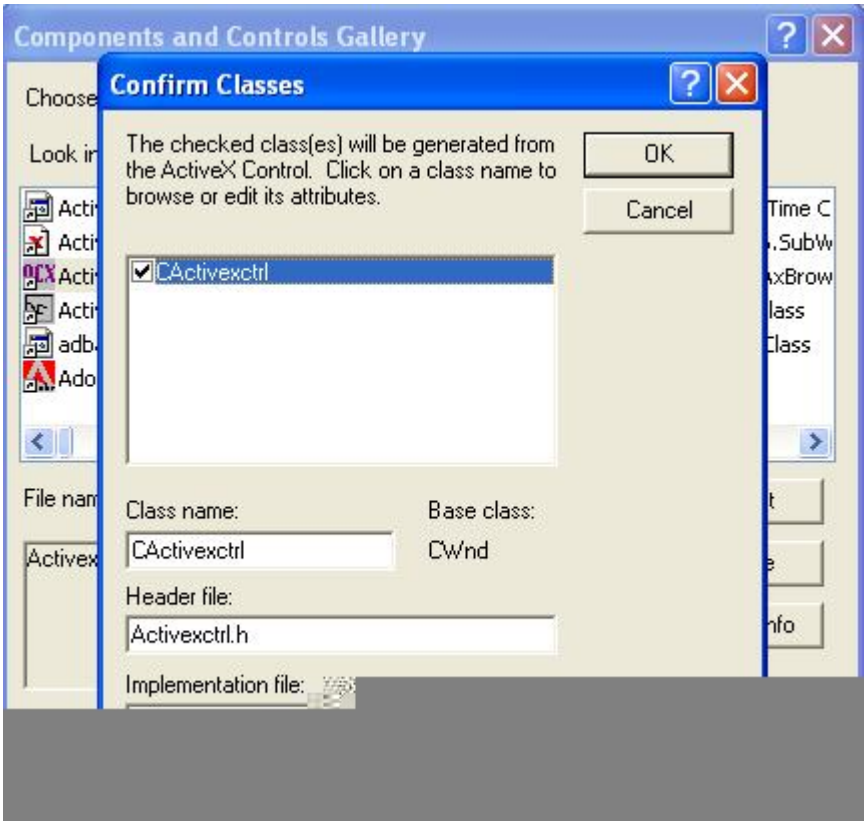

New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
sample
Location:
E:\MAM\VISUAL PROGRAMMI
Create new workspace
Add to current workspace
Dependency of:
Platforms:
Win32
OK
Cancel

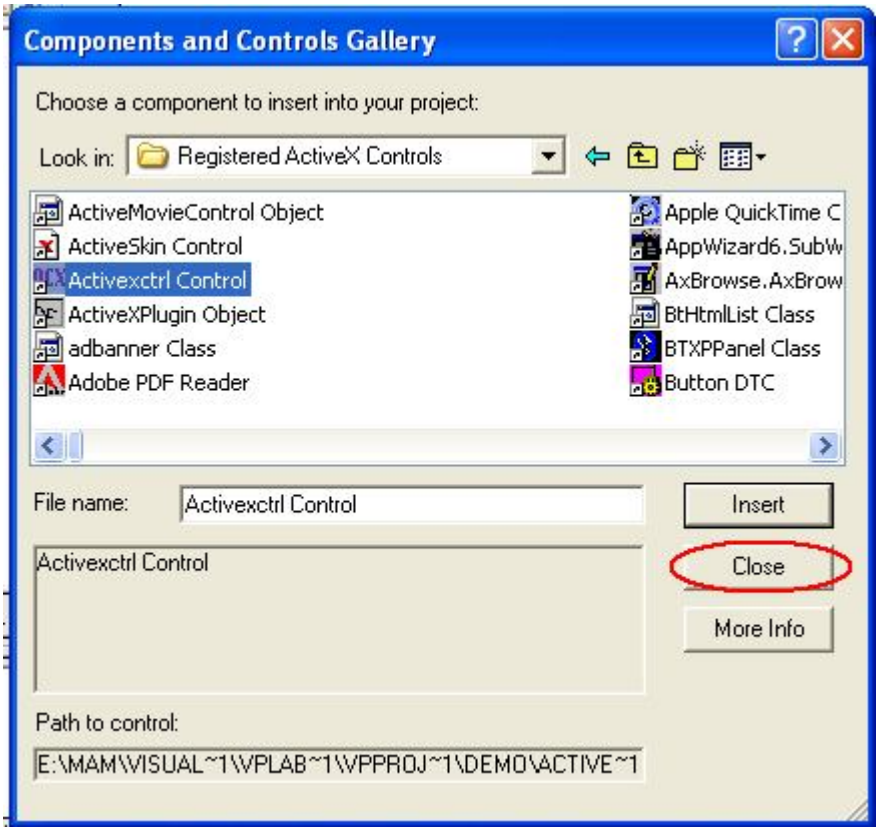

MFC AppWizard - Step 1
Application
OK
Cancel
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

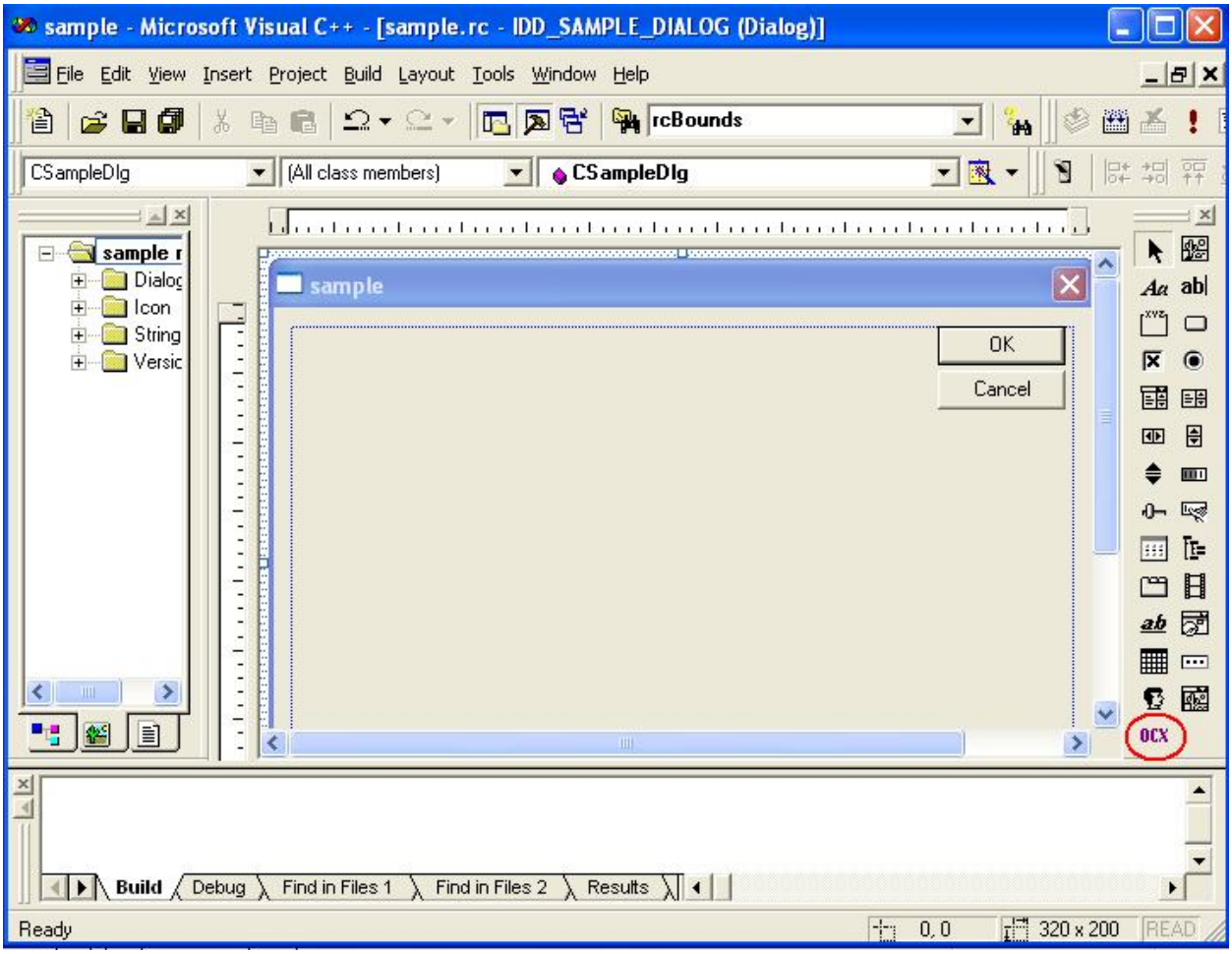

sample - Microsoft Visual C++ - [sample.rc - IDD_SAMPLE_DIALOG (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
Set Active Project
rcBounds
Add To Project
Source Control
Dependencies...
Settings... Alt+F7
Export Makefile...
Insert Project into Workspace...
New...
New Folder...
Files...
Data Connection...
Components and Controls...
CSampleDlg
sample resources
Dialog
Icon
String Table
Version
OK
Cancel
TODO: Place dialog controls here.
Class...
Reso...
FileVi...

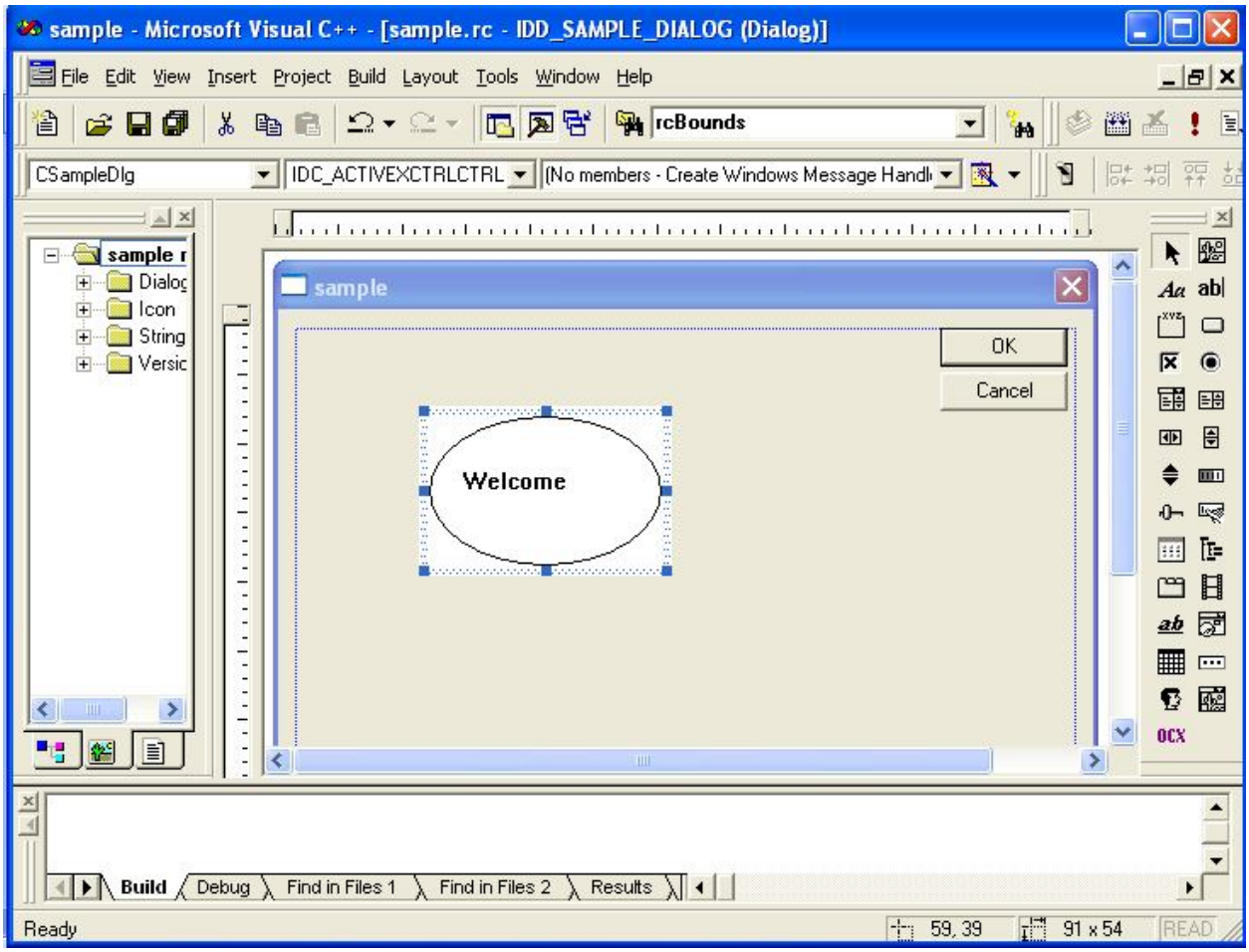

Components and Controls Gallery
Choose a component to insert into your project:
Look in:
Gallery
Registered ActiveX Controls
Visual C++ Components
MSCREATE.DIR
Type: DIR File
Date Modified: 12/15/2009 11:12 PM
Size: 0 bytes
File name:
Insert
Close
More Info
Path to control:


Components and Controls Gallery
Choose a component to insert into your project:
Look in:
Registered ActiveX Controls
ActiveMovieControl Object
ActiveSkin Control
Activexctrl Control
ActiveXPlugin Object
adbanner Class
Adobe PDF Reader
Apple QuickTime C
AppWizard6.SubW
AxBrowse.AxBrow
BtHtmlList Class
BTXPPanel Class
Button DTC
File name:
Activexctrl Control
Insert
Close
More Info
Activexctrl Control
Path to control:
E:\MAM\VISUAL~1\VPLAB~1\VPPROJ~1\DEMO\ACTIVE~1


Components and Controls Gallery
Confirm Classes
The checked class(es) will be generated from the ActiveX Control. Click on a class name to browse or edit its attributes.
OK
Cancel
CActivexctrl
Class name:
CActivexctrl
Base class:
CWnd
Header file:
Activexctrl.h
Implementation file:


Components and Controls Gallery
Choose a component to insert into your project:
Look in:
Registered ActiveX Controls
ActiveMovieControl Object
ActiveSkin Control
Activexctrl Control
ActiveXPlugin Object
adbanner Class
Adobe PDF Reader
Apple QuickTime C
AppWizard6.SubW
AxBrowse.AxBrow
BtHtmlList Class
BTXPPanel Class
Button DTC
File name:
Activexctrl Control
Insert
Close
More Info
Activexctrl Control
Path to control:
E:\MAM\VISUAL~1\VPLAB~1\VPPROJ~1\DEMO\ACTIVE~1


sample - Microsoft Visual C++ - [sample.rc - IDD_SAMPLE_DIALOG (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
rcBounds
CSampleDlg
(All class members)
CSampleDlg
sample r
Dialog
Icon
String
Versic
sample
OK
Cancel
OCX
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
0, 0
320 x 200
READ



Result:

Thus the VC++ program to build an application that uses ActiveX Control has been developed, built and executed.

11. CREATING DLL

Aim:

To write a VC++ program to develop DLL and export the function using MFC (.dll) and use the same function in the client program using MFC (.exe).

Concept:

DLL is a file on disk with a DLL extension consisting of global data, compiled functions, and resources that becomes part of our process. It is compiled to load at a preferred base address, and if there is no conflict with other DLLs, the file gets mapped to the same virtual address in our process. The DLL consists of exported function, and the client program imports those exported functions. Windows matches up the imports and exports when it loads the DLL.

- Classes are build-time modular
- DLLs are runtime modular

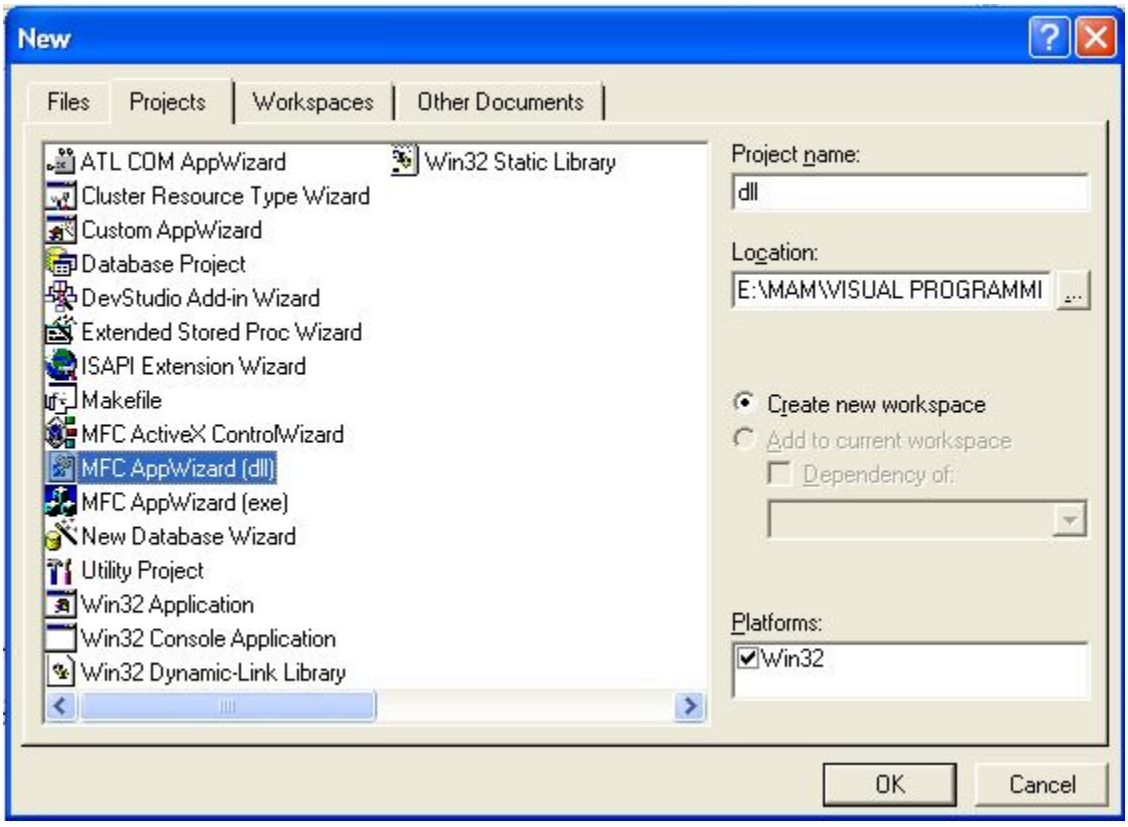
Client programs can load and link our DLL very quickly when they run.

This program is to create hello( ) in the DLL and exported this function to any client program to use it. Also this DLL is included in the client program to use the hello(). This hello() will return a message box and also the sum of two numbers in the client area from the DLL.

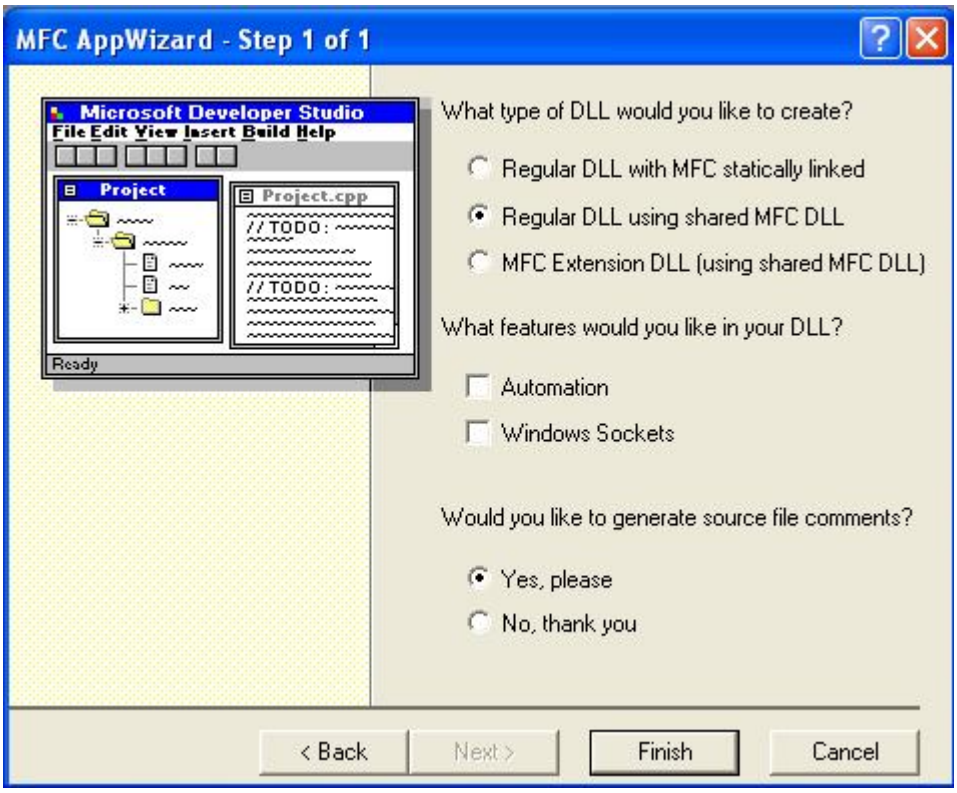
Procedure:

DLL Creation:

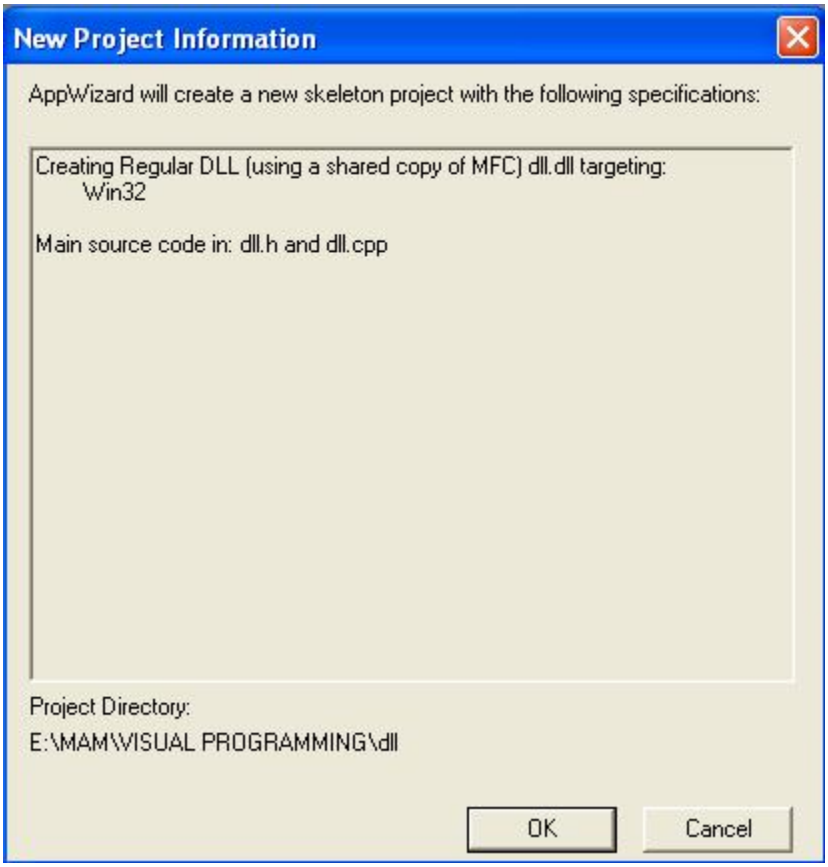
Step 1:


New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
dll
Location:
E:\MAM\VISUAL PROGRAMMI
Create new workspace
Add to current workspace
Dependency of:
Platforms:
Win32
OK
Cancel

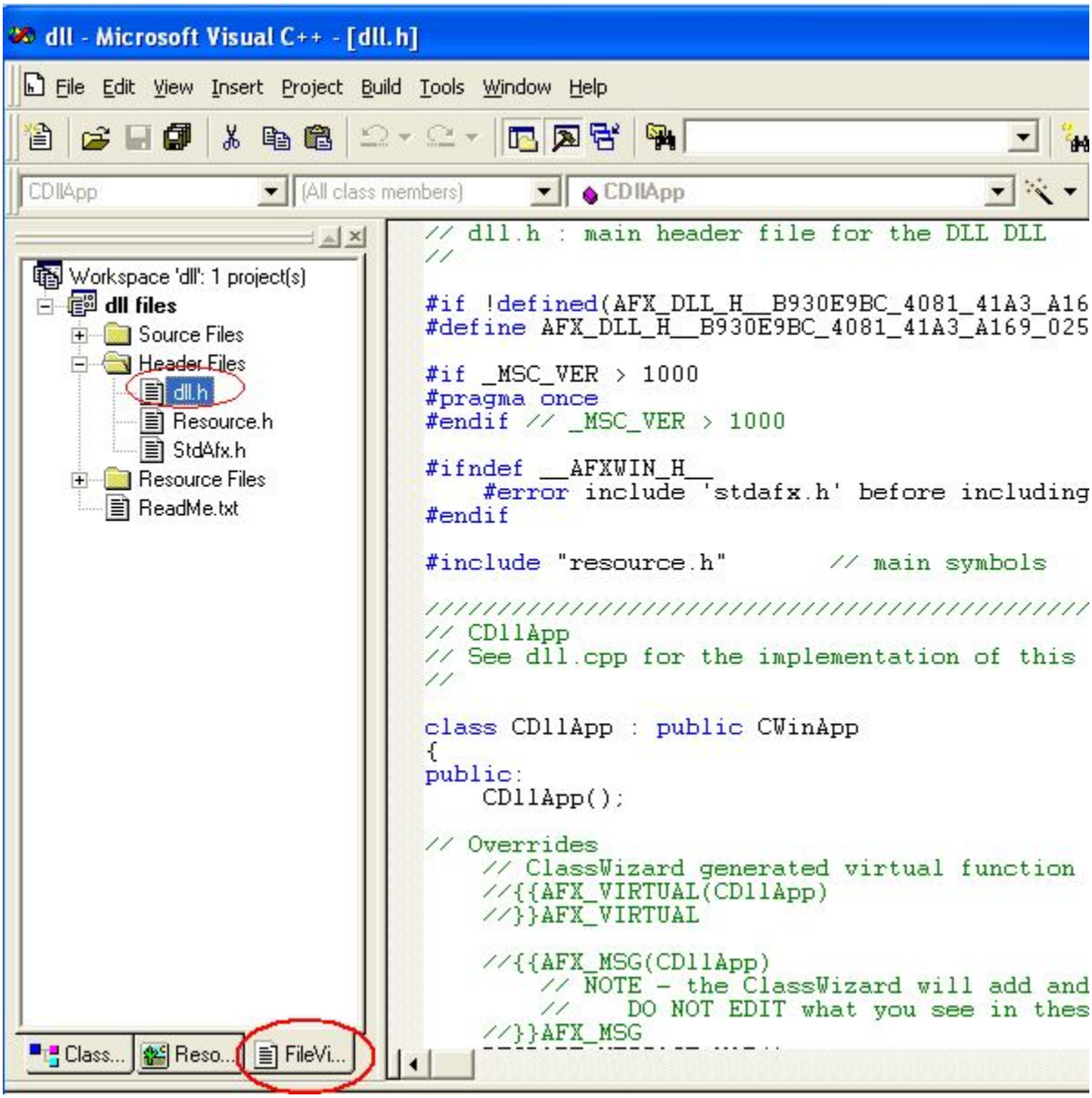
Step 2:


MFC AppWizard - Step 1 of 1
Microsoft Developer Studio
File Edit View Insert Build Help
Project
Project.cpp
//TODO:
//TODO:
Ready
What type of DLL would you like to create?
Regular DLL with MFC statically linked
Regular DLL using shared MFC DLL
MFC Extension DLL (using shared MFC DLL)
What features would you like in your DLL?
Automation
Windows Sockets
Would you like to generate source file comments?
Yes, please
No, thank you
< Back
Next >
Finish
Cancel

Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Creating Regular DLL (using a shared copy of MFC) dll.dll targeting:
Win32
Main source code in: dll.h and dll.cpp
Project Directory:
E:\MAM\VISUAL PROGRAMMING\dll
OK
Cancel

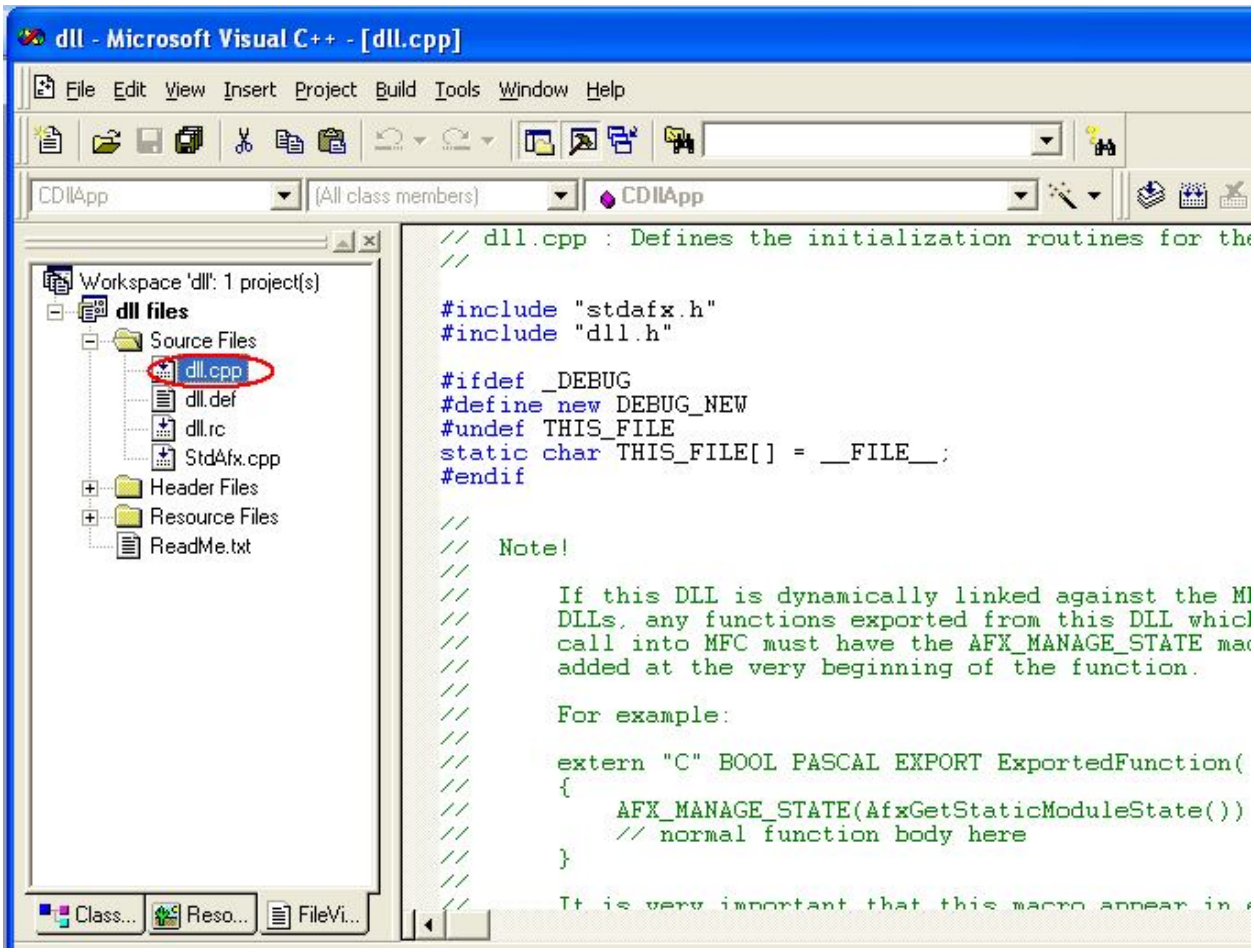
Step 4:



Step 5:

_declspec (dllexport) int WINAPI hello(int, int);

Step 6:

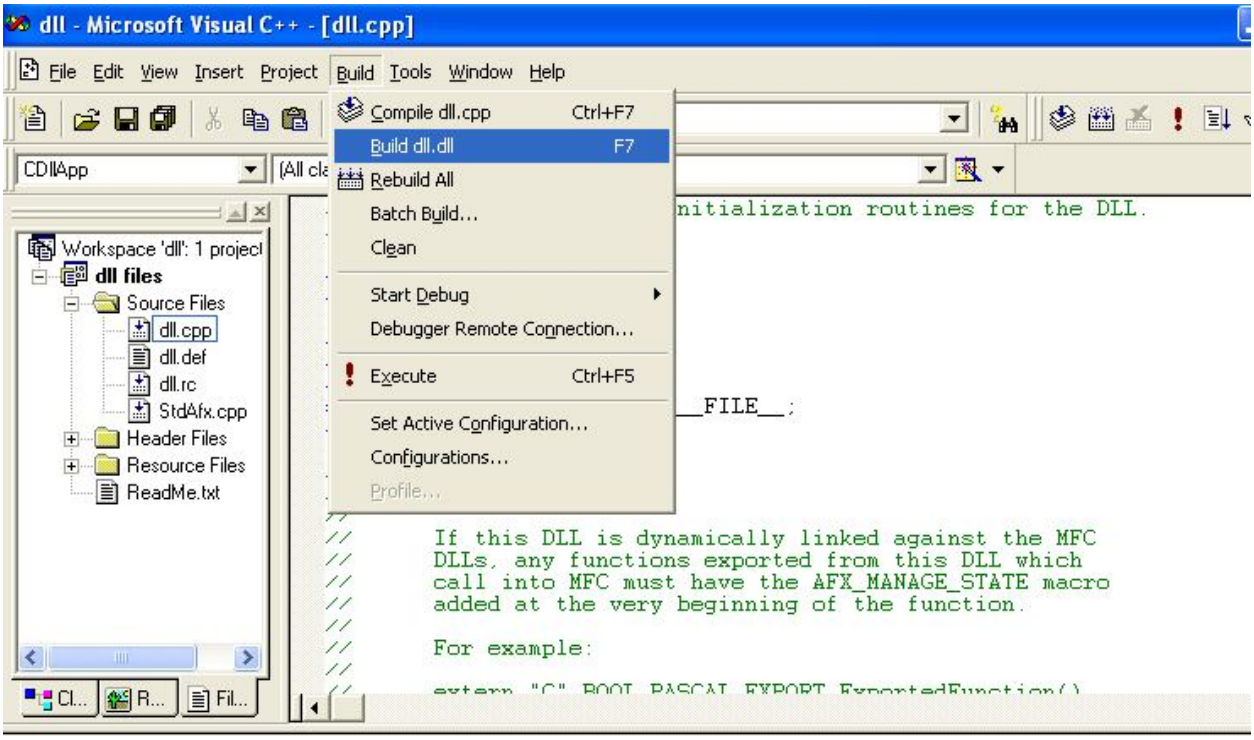


Step 7:

```
_declspec (dllexport) int WINAPI hello(int a, int b)
{
        int sum;
        sum= a+b;
        MessageBox (NULL, TEXT("The sum of 3 and 4 is"),TEXT("welcome"),0);
        return sum;
}
```

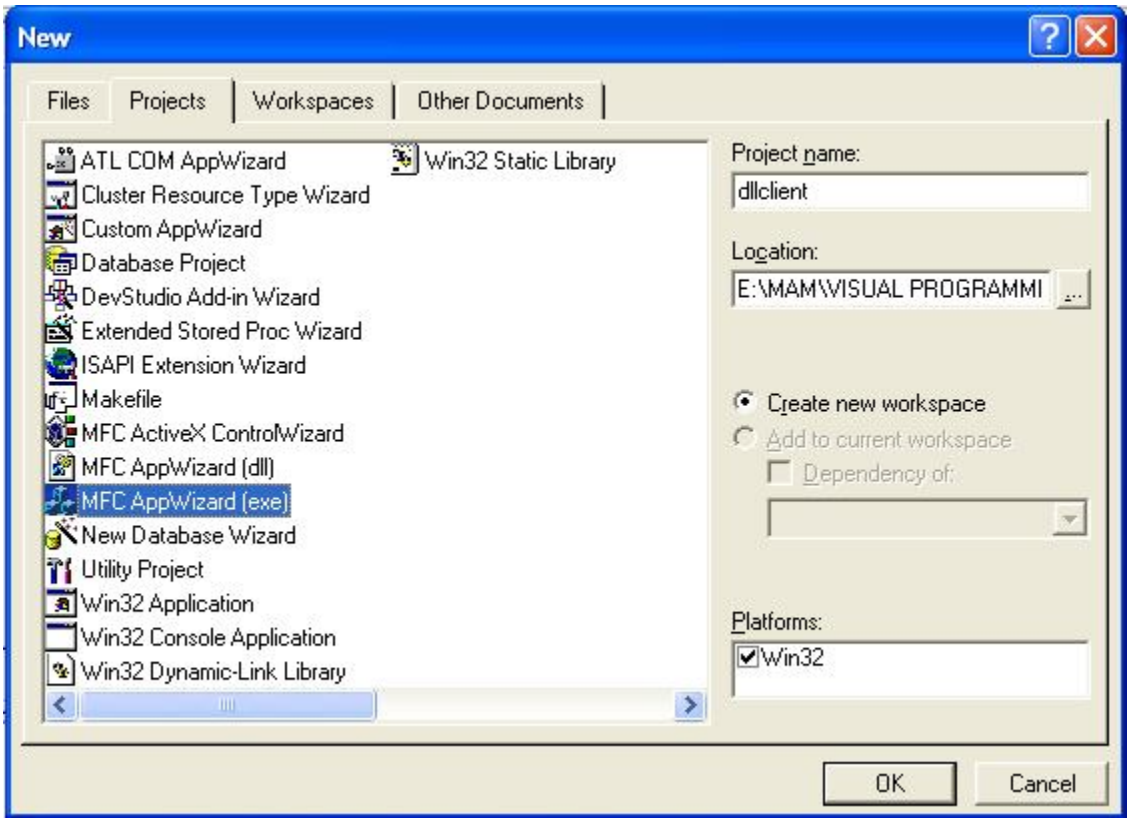
Step 8:

Build the code without executing it.

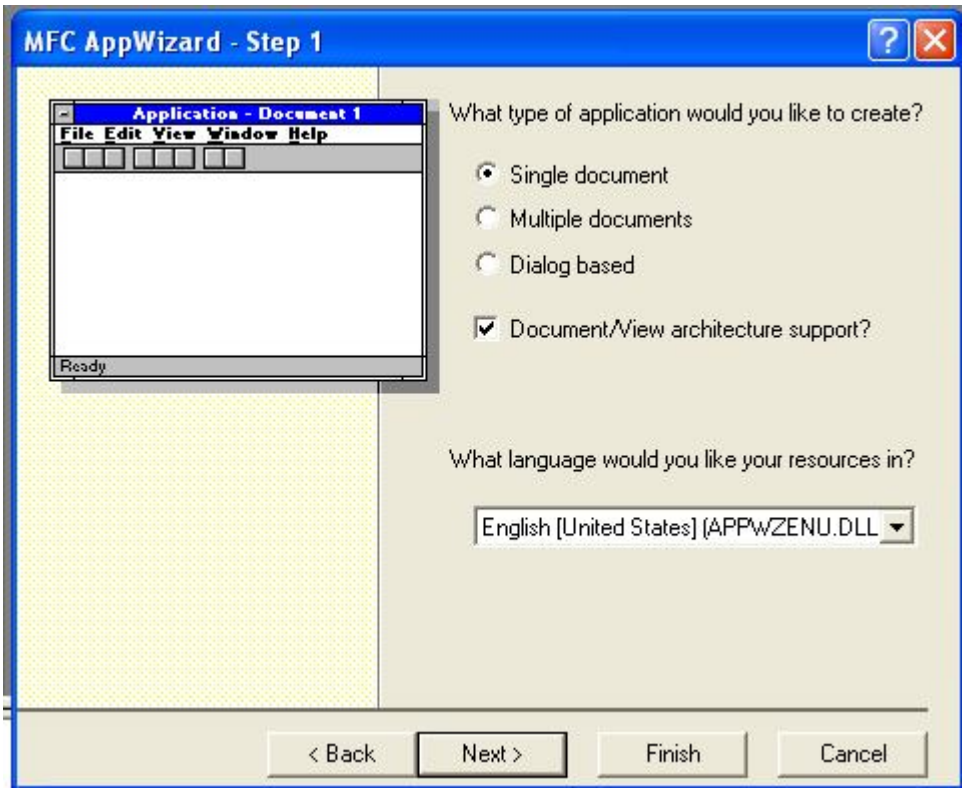


Dll Client Application:

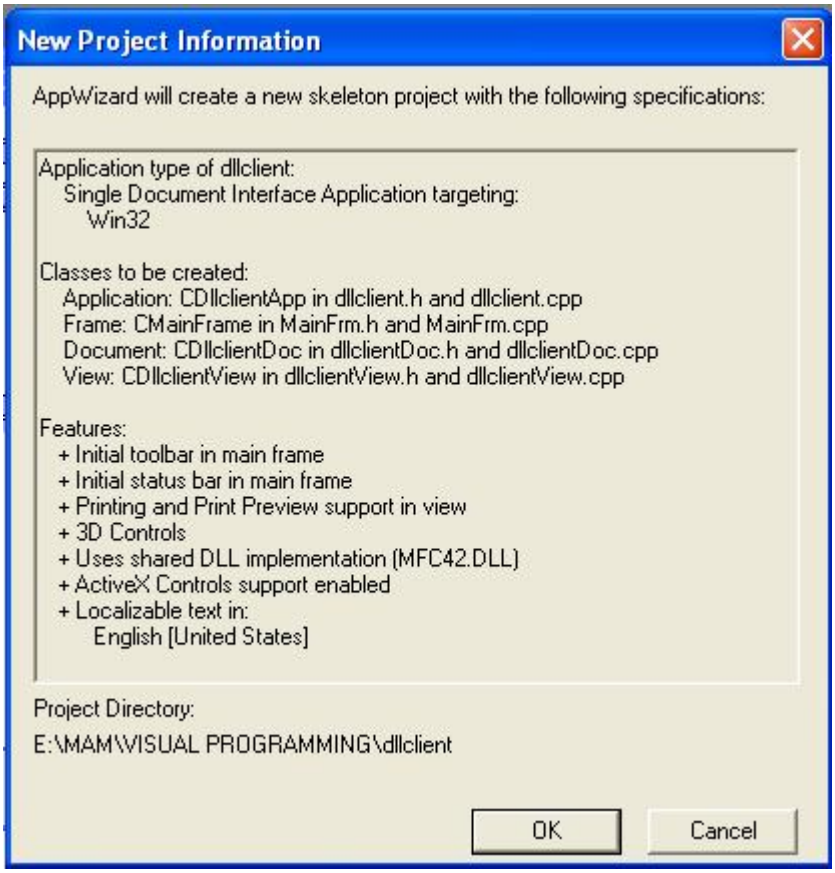
Step 10:


New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
dllclient
Location:
E:\MAM\VISUAL PROGRAMMI
Create new workspace
Add to current workspace
Dependency of:
Platforms:
Win32
OK
Cancel

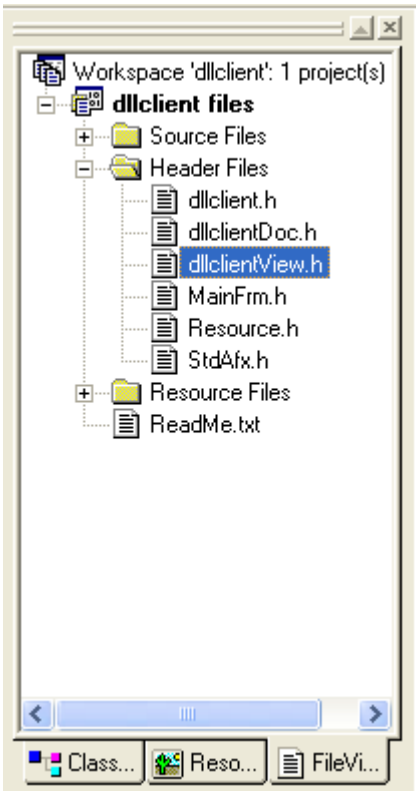
Step 11:


MFC AppWizard - Step 1
Application - Document 1
File Edit View Window Help
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

Step 12:

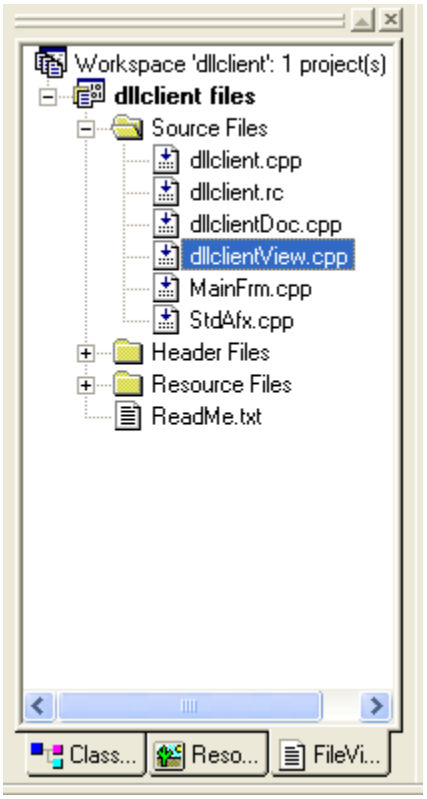

New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of dllclient:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CDllclientApp in dllclient.h and dllclient.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CDllclientDoc in dllclientDoc.h and dllclientDoc.cpp
View: CDllclientView in dllclientView.h and dllclientView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\MAM\VISUAL PROGRAMMING\dllclient
OK
Cancel

Step 13:

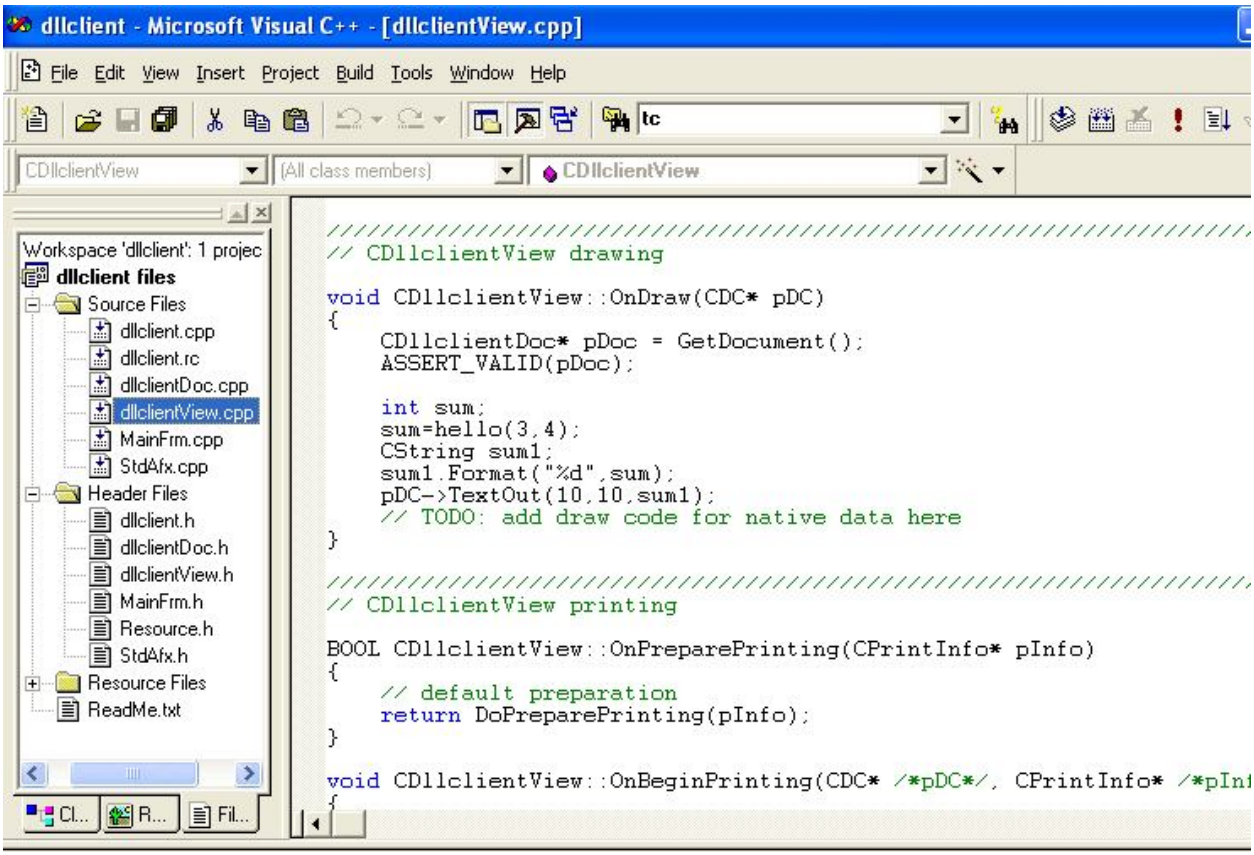


_declspec (dllimport) int WINAPI hello(int, int);

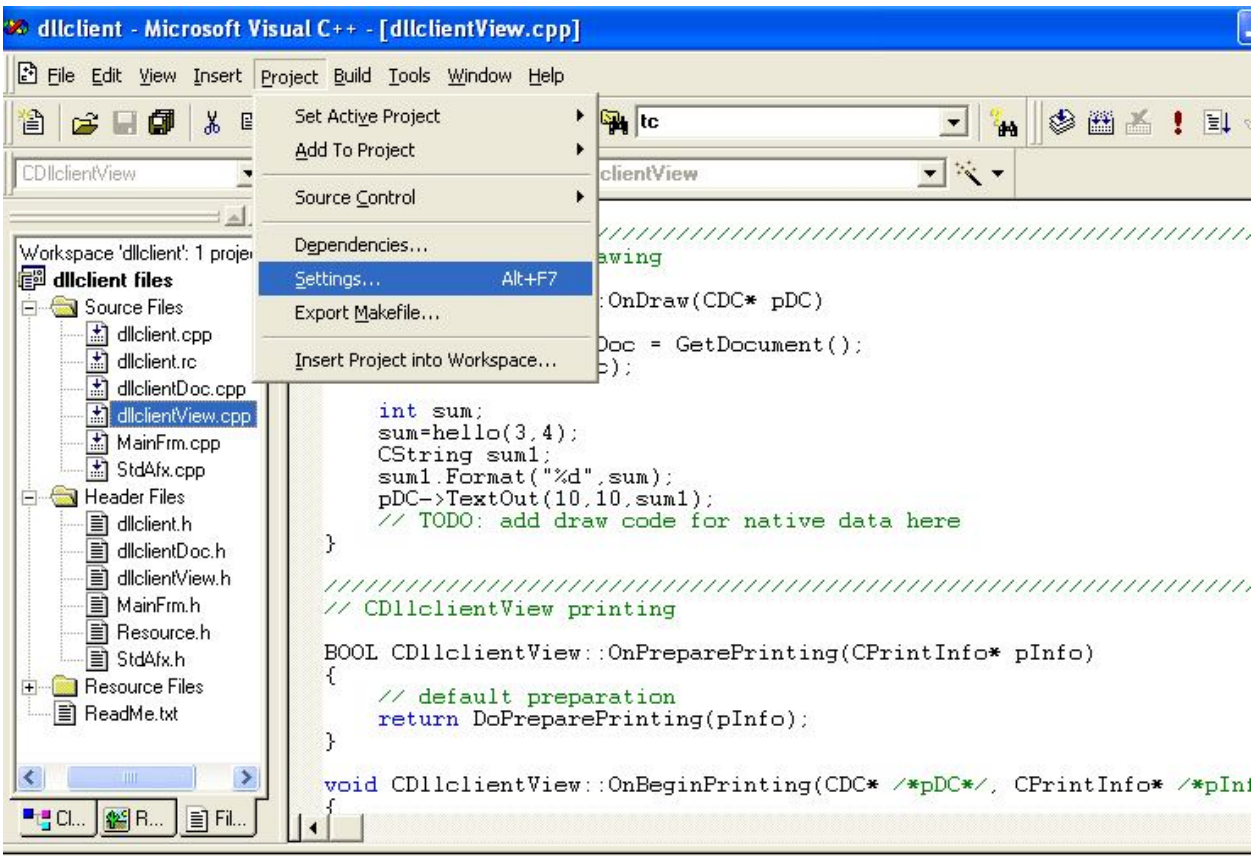
Step 14:


Workspace 'dllclient': 1 project(s)
dllclient files
Source Files
dllclient.cpp
dllclient.rc
dllclientDoc.cpp
dllclientView.cpp
MainFrm.cpp
StdAfx.cpp
Header Files
Resource Files
ReadMe.txt
Class...
Reso...
FileVi...

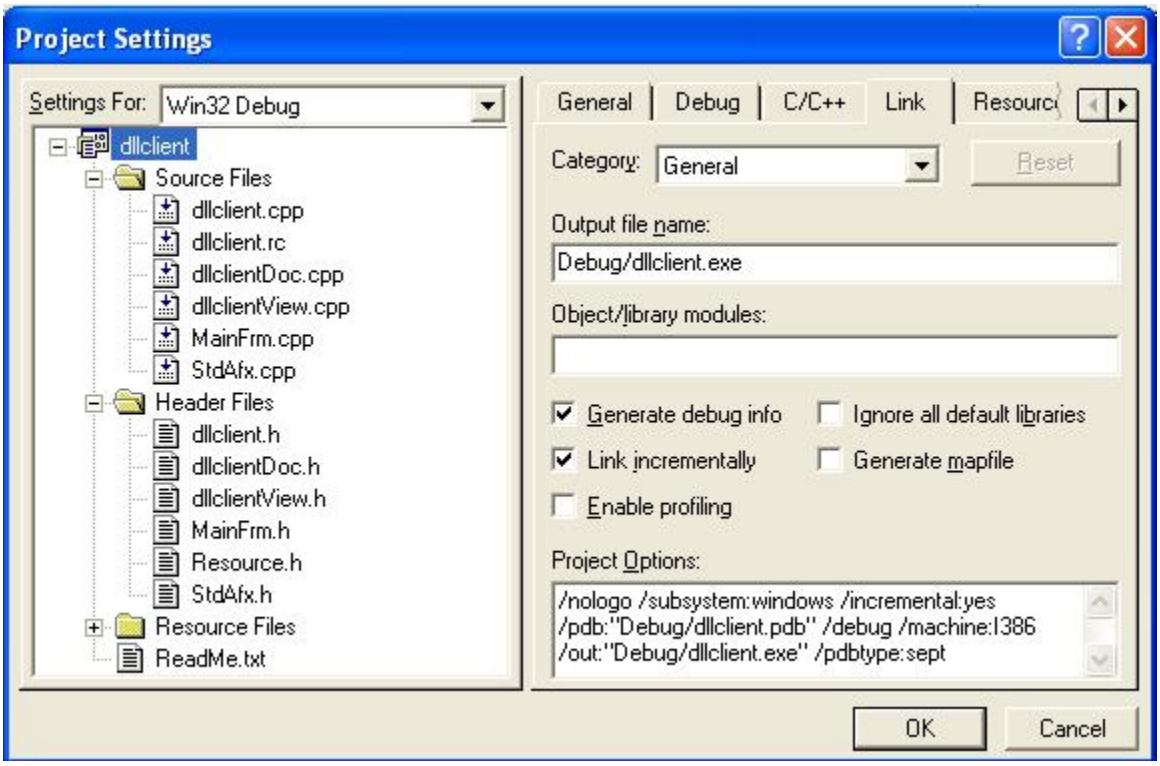
Step 15:


dllclient - Microsoft Visual C++ - [dllclientView.cpp]
File Edit View Insert Project Build Tools Window Help
tc
CDllclientView
(All class members)
CDllclientView
Workspace 'dllclient': 1 projec
dllclient files
Source Files
dllclient.cpp
dllclient.rc
dllclientDoc.cpp
dllclientView.cpp
MainFrm.cpp
StdAfx.cpp
Header Files
dllclient.h
dllclientDoc.h
dllclientView.h
MainFrm.h
Resource.h
StdAfx.h
Resource Files
ReadMe.txt
Cl...
R...
Fil...
// CDllclientView drawing
void CDllclientView::OnDraw(CDC* pDC)
{
CDllclientDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
int sum;
sum=hello(3,4);
CString sum1;
sum1.Format("%d",sum);
pDC->TextOut(10,10,sum1);
// TODO: add draw code for native data here
}
// CDllclientView printing
BOOL CDllclientView::OnPreparePrinting(CPrintInfo* pInfo)
{
// default preparation
return DoPreparePrinting(pInfo);
}
void CDllclientView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInf
{

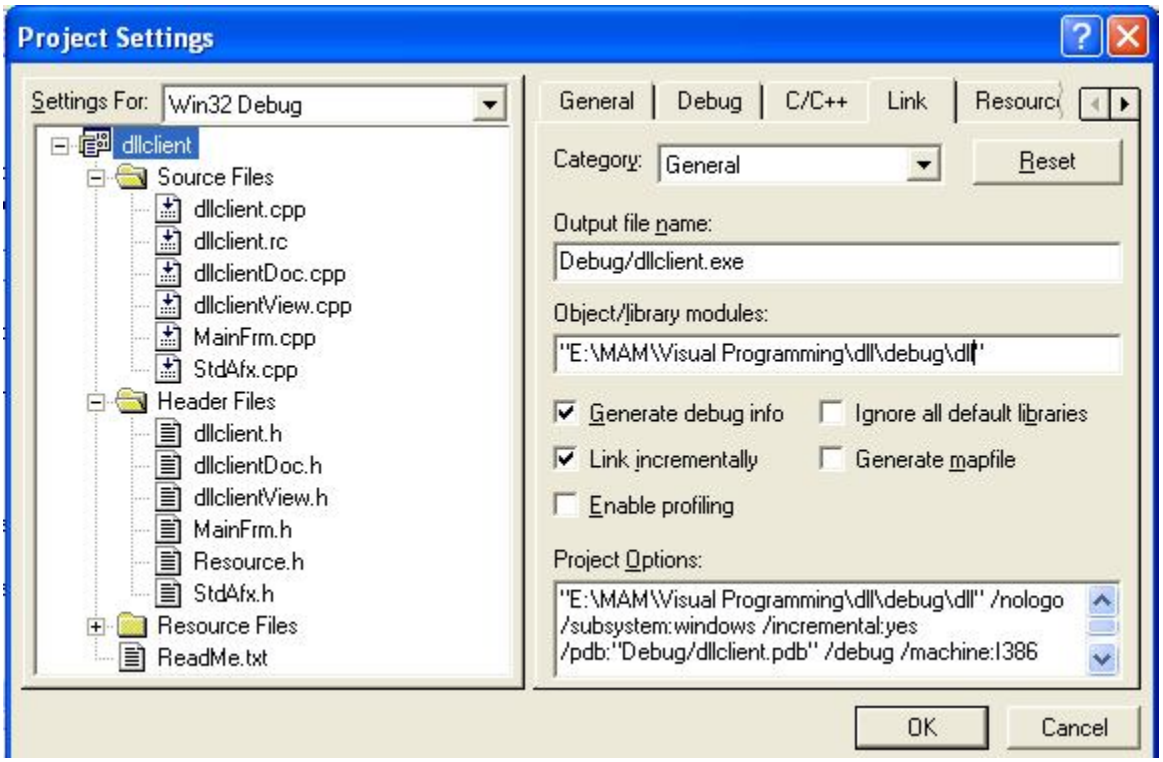
Step 16:


dllclient - Microsoft Visual C++ - [dllclientView.cpp]
File Edit View Insert Project Build Tools Window Help
Set Active Project
Add To Project
Source Control
Dependencies...
Settings... Alt+F7
Export Makefile...
Insert Project into Workspace...
Workspace 'dllclient': 1 proje
dllclient files
Source Files
dllclient.cpp
dllclient.rc
dllclientDoc.cpp
dllclientView.cpp
MainFrm.cpp
StdAfx.cpp
Header Files
dllclient.h
dllclientDoc.h
dllclientView.h
MainFrm.h
Resource.h
StdAfx.h
Resource Files
ReadMe.txt
int sum;
sum=hello(3,4);
CString sum1;
sum1.Format("%d",sum);
pDC->TextOut(10,10,sum1);
// TODO: add draw code for native data here
// CDllclientView printing
BOOL CDllclientView::OnPreparePrinting(CPrintInfo* pInfo)
// default preparation
return DoPreparePrinting(pInfo);
void CDllclientView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInf

Step 17:



Step 18:



Step 19:

Program:

// dll.h

```
_declspec (dllexport) int WINAPI hello(int, int);
```

//dll.cpp

```
_declspec (dllexport) int WINAPI hello(int a, int b)
{
        int sum;
        sum= a+b;
        MessageBox (NULL, TEXT("The sum of 3 and 4 is"),TEXT("welcome"),0);
        return sum;
}
```

//dllclientView.h

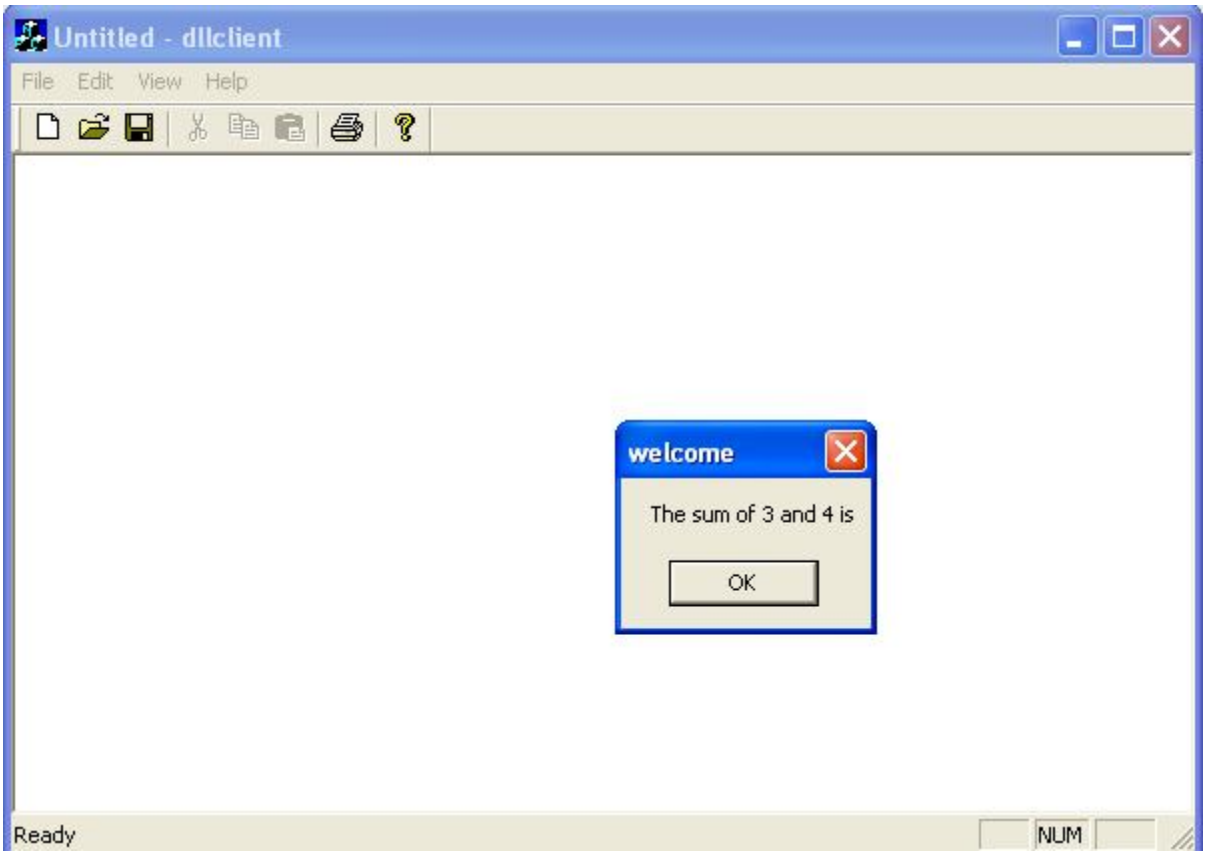
```
_declspec (dllimport) int WINAPI hello(int, int);
```

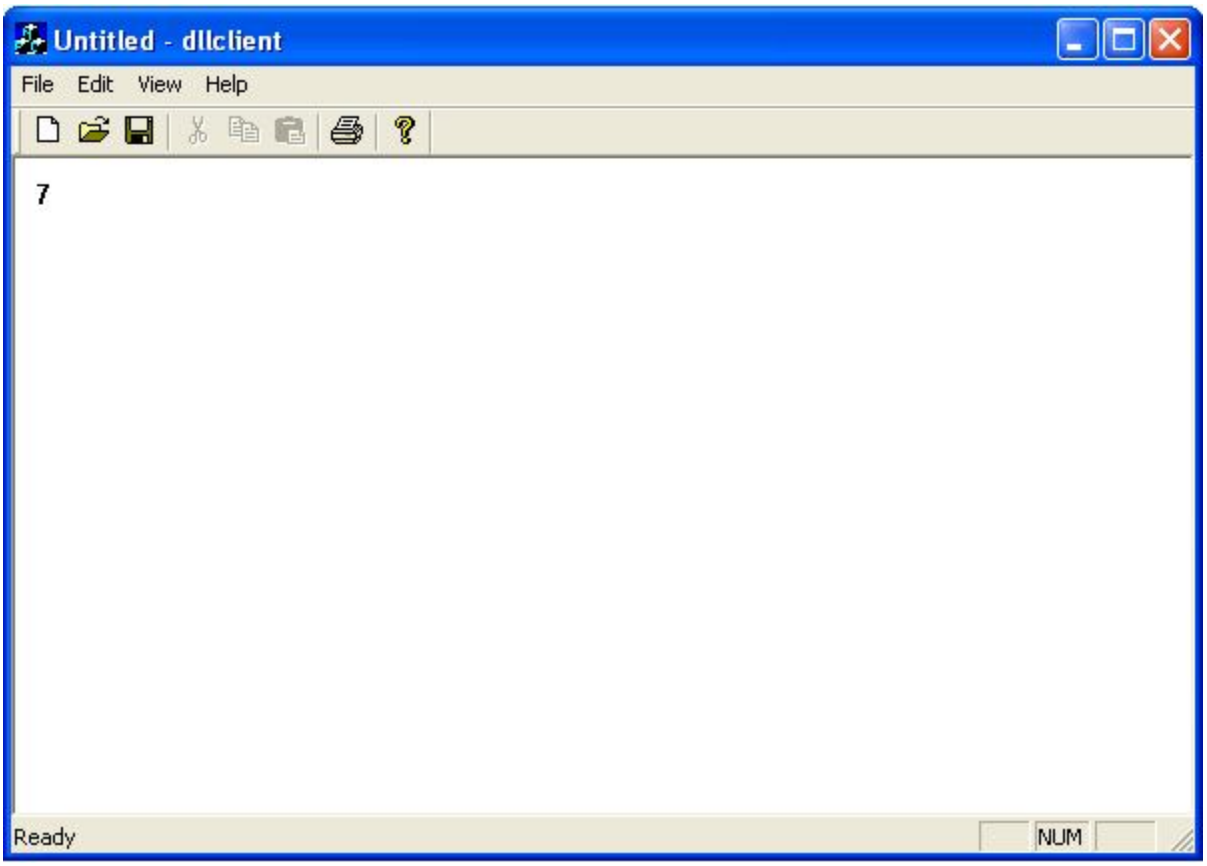
//dllclientView.cpp

```
void CDllclientView::OnDraw(CDC* pDC)
{
        CDllclientDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        int sum;
        sum=hello(3,4);
        CString sum1;
        sum1.Format("%d",sum);
        pDC->TextOut(10,10,sum1);
```

}

Output:


Untitled - dllclient
File Edit View Help
welcome
The sum of 3 and 4 is
OK
Ready
NUM



Result:

Thus the VC++ program to develop DLL and export the function to use in the client program has been developed, builds and verified.

12. DYNAMIC CONTROL

Aim:

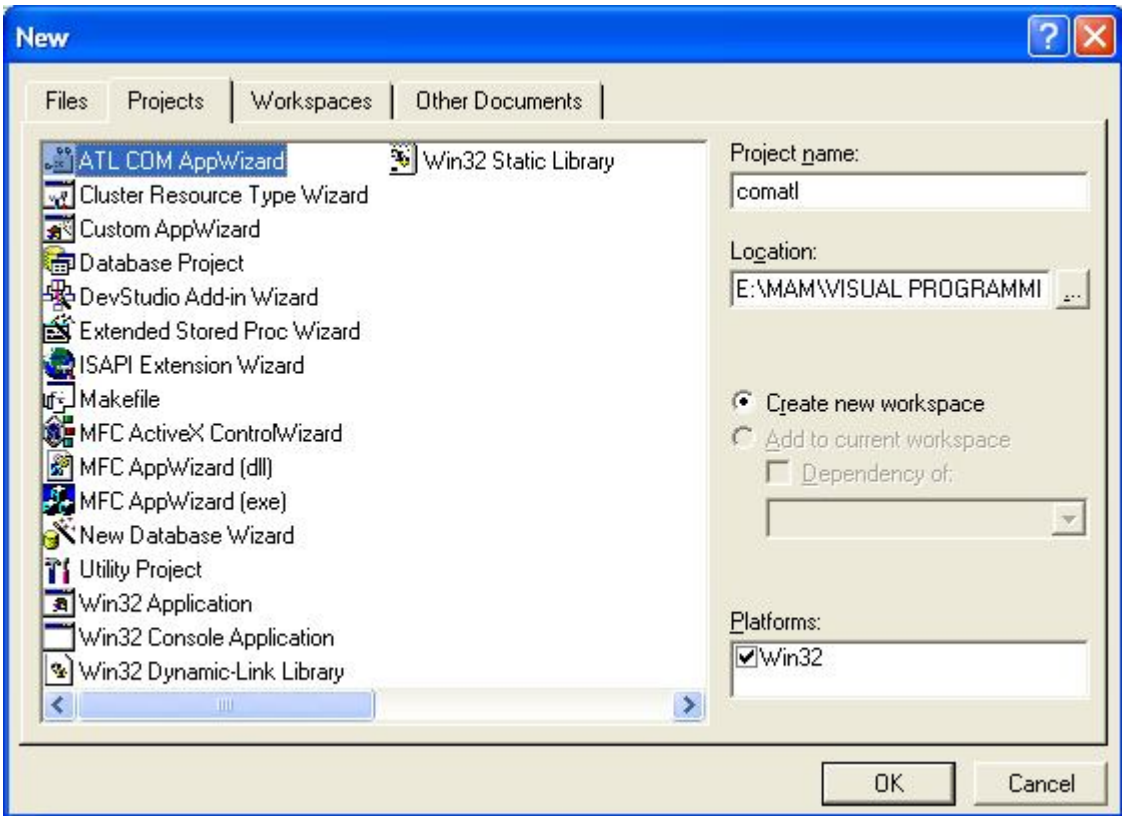
To write a VC++ program to build COM component using ATL COM AppWizard and test the created COM component using VB6.0.

Concept:

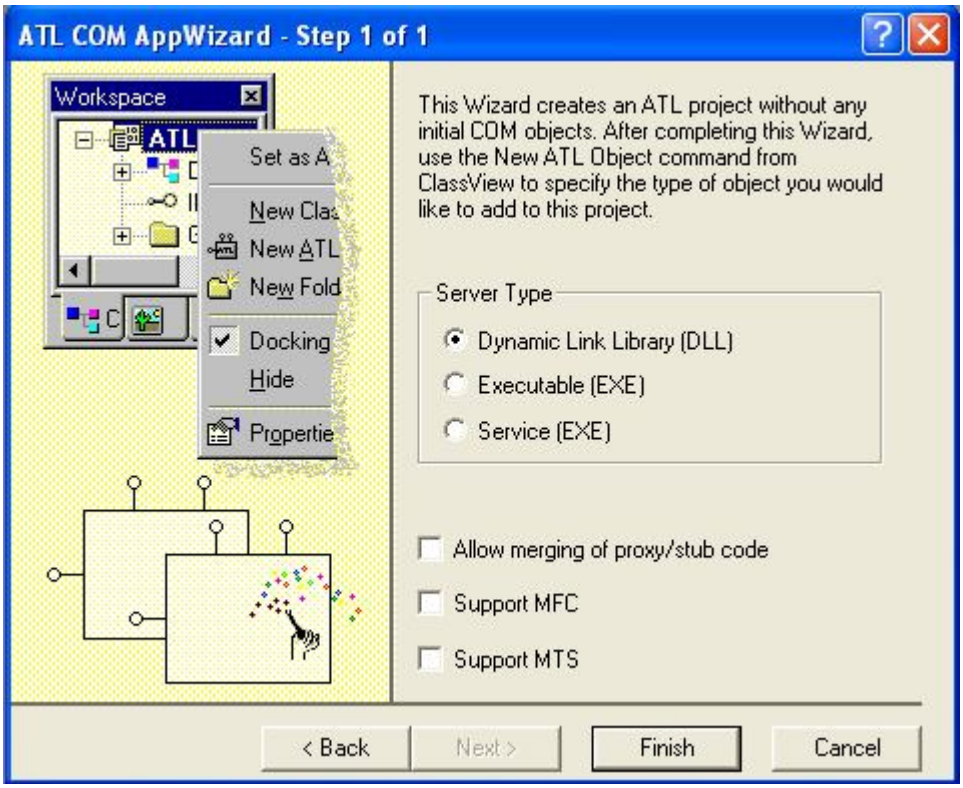
COM is a powerful integrating technology. It allows developers to write software that runs regardless of issues such as thread awareness and language choice. This program is used to create a DLL as a COM component and using that component in the visual basic 6.0.

Procedure:

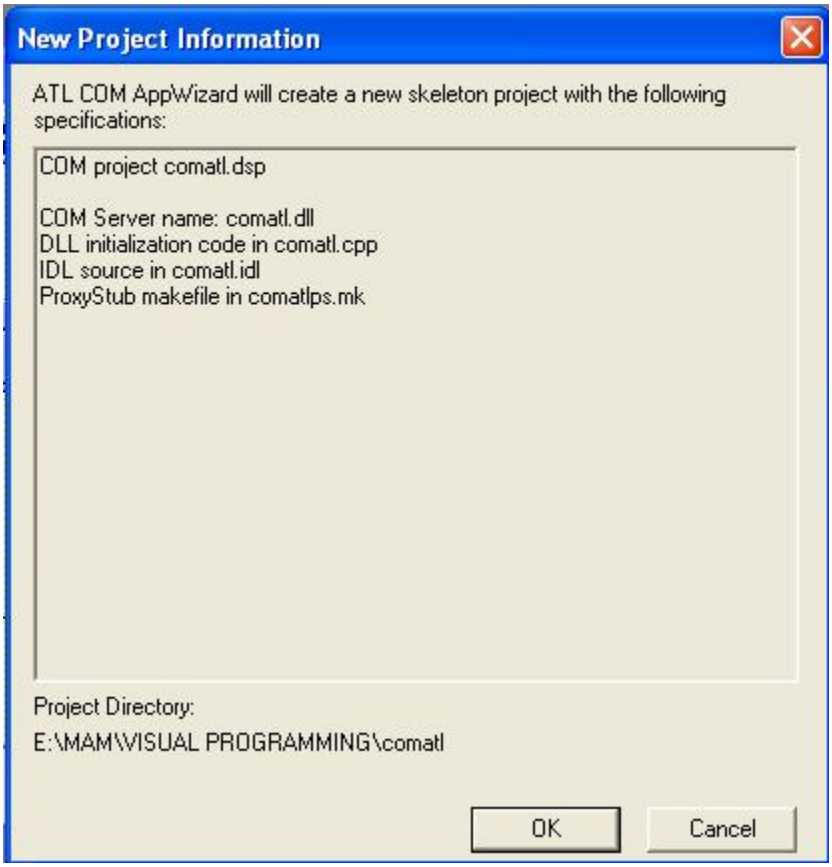
Step 1:



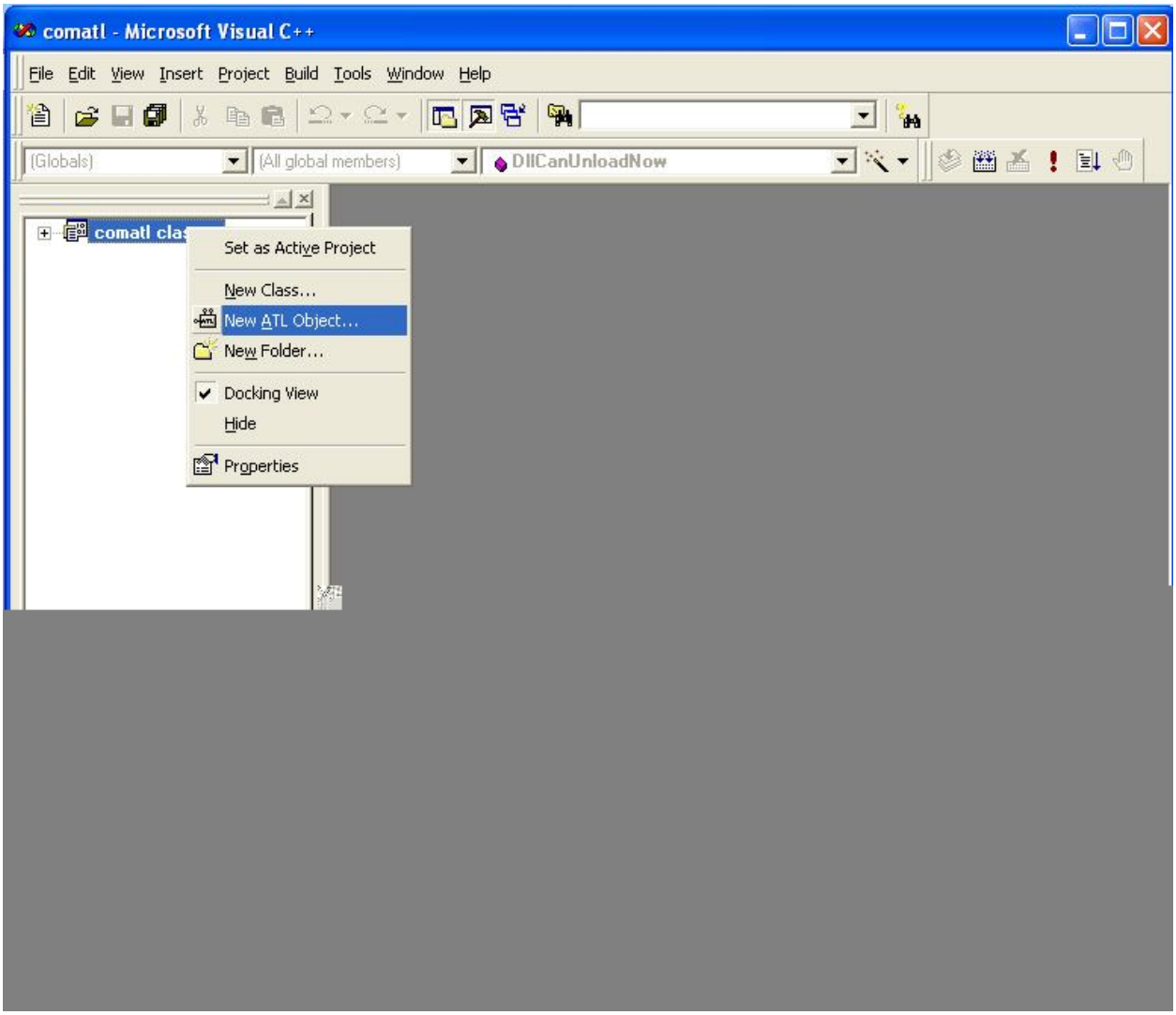
Step 2:


ATL COM AppWizard - Step 1 of 1
Workspace
Set as A
New Cla
New ATL
New Fold
Docking
Hide
Propertie
This Wizard creates an ATL project without any initial COM objects. After completing this Wizard, use the New ATL Object command from ClassView to specify the type of object you would like to add to this project.
Server Type
Dynamic Link Library (DLL)
Executable (EXE)
Service (EXE)
Allow merging of proxy/stub code
Support MFC
Support MTS
< Back
Next >
Finish
Cancel

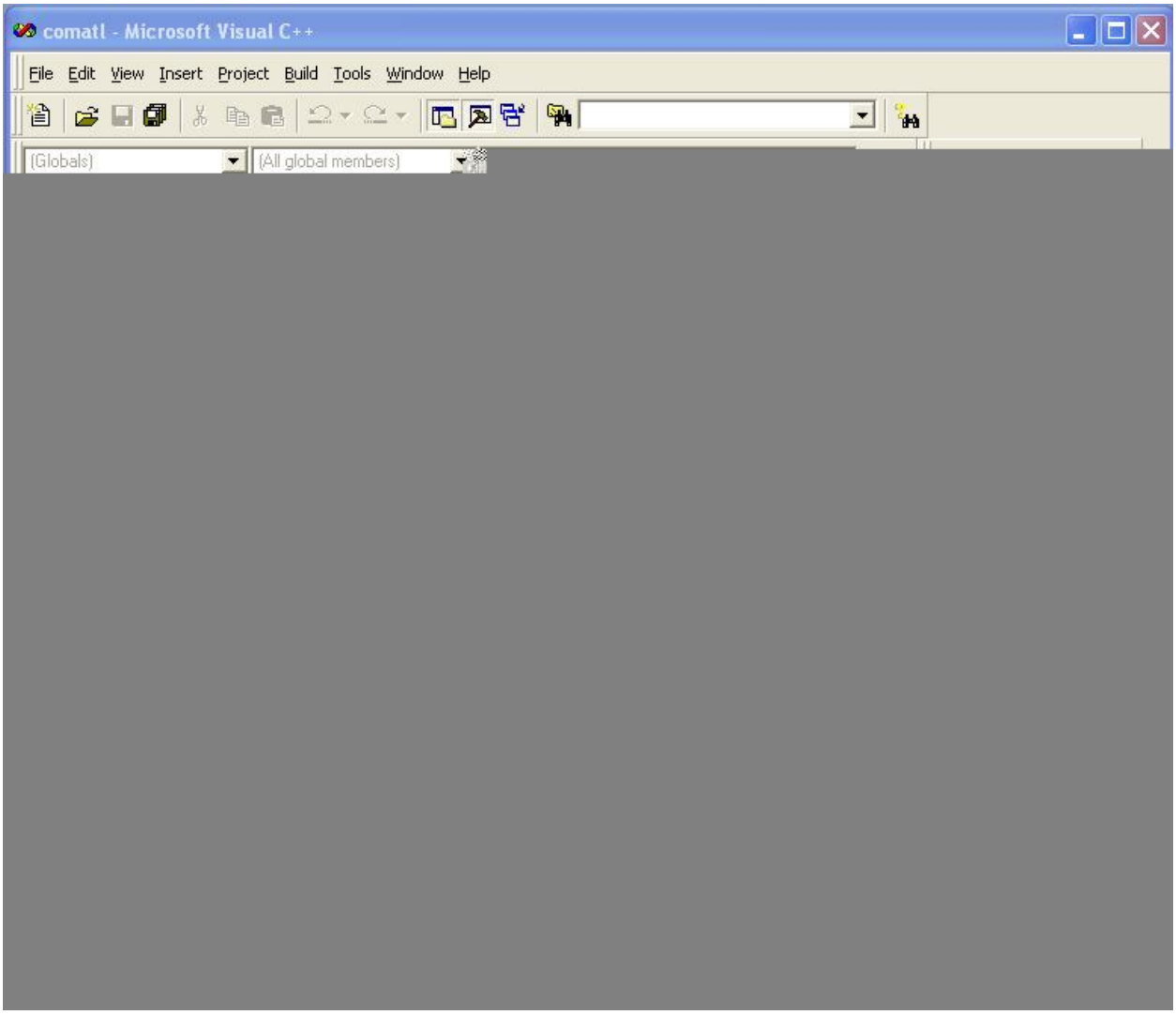
Step 3:


New Project Information
ATL COM AppWizard will create a new skeleton project with the following specifications:
COM project comatl.dsp
COM Server name: comatl.dll
DLL initialization code in comatl.cpp
IDL source in comatl.idl
ProxyStub makefile in comatlps.mk
Project Directory:
E:\MAM\VISUAL PROGRAMMING\comatl
OK
Cancel

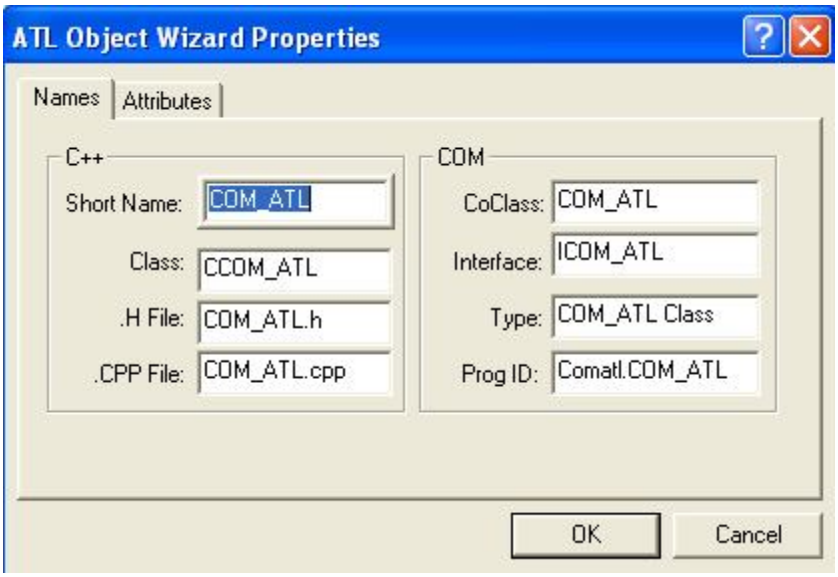
Step 4:


comatl - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
DllCanUnloadNow
comatl cla
Set as Active Project
New Class...
New ATL Object...
New Folder...
Docking View
Hide
Properties

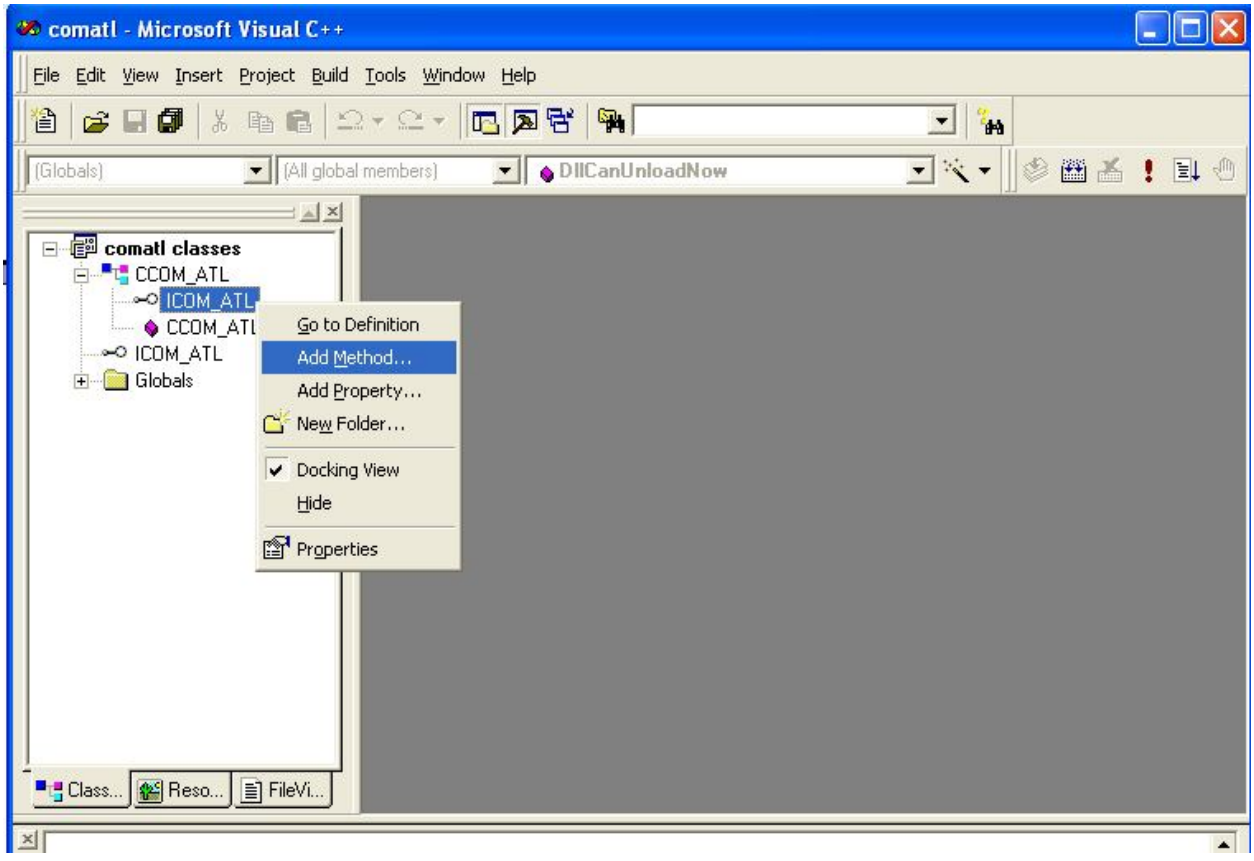
Step 5:


comatl - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)

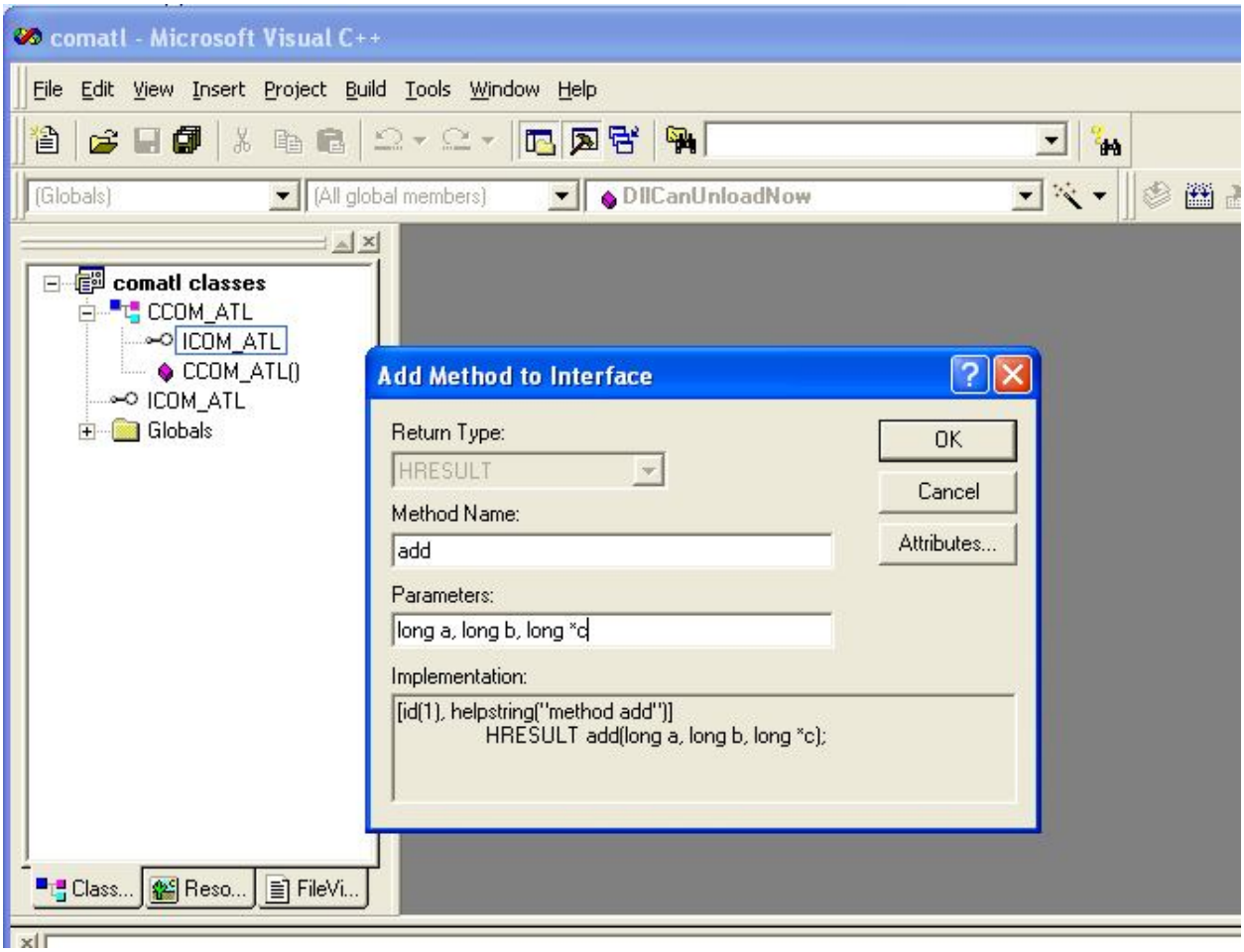
Step 6:


ATL Object Wizard Properties
Names
Attributes
C++
Short Name:
COM_ATL
Class:
CCOM_ATL
.H File:
COM_ATL.h
.CPP File:
COM_ATL.cpp
COM
CoClass:
COM_ATL
Interface:
ICOM_ATL
Type:
COM_ATL Class
Prog ID:
Comatl.COM_ATL
OK
Cancel

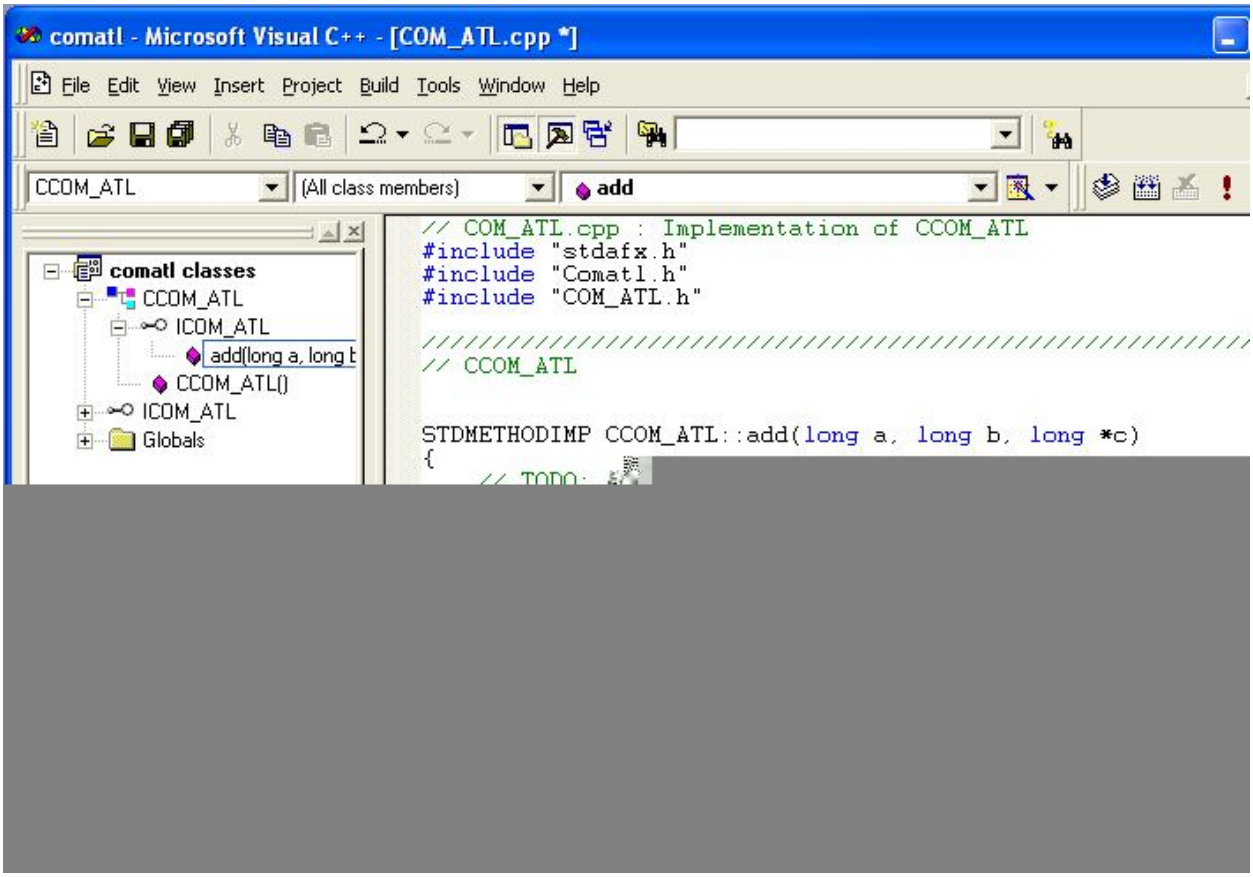
Step 7:


comatl - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
DllCanUnloadNow
comatl classes
CCOM_ATL
ICOM_ATL
CCOM_ATL
ICOM_ATL
Globals
Go to Definition
Add Method...
Add Property...
New Folder...
Docking View
Hide
Properties
Class...
Reso...
FileVi...

Step 8:

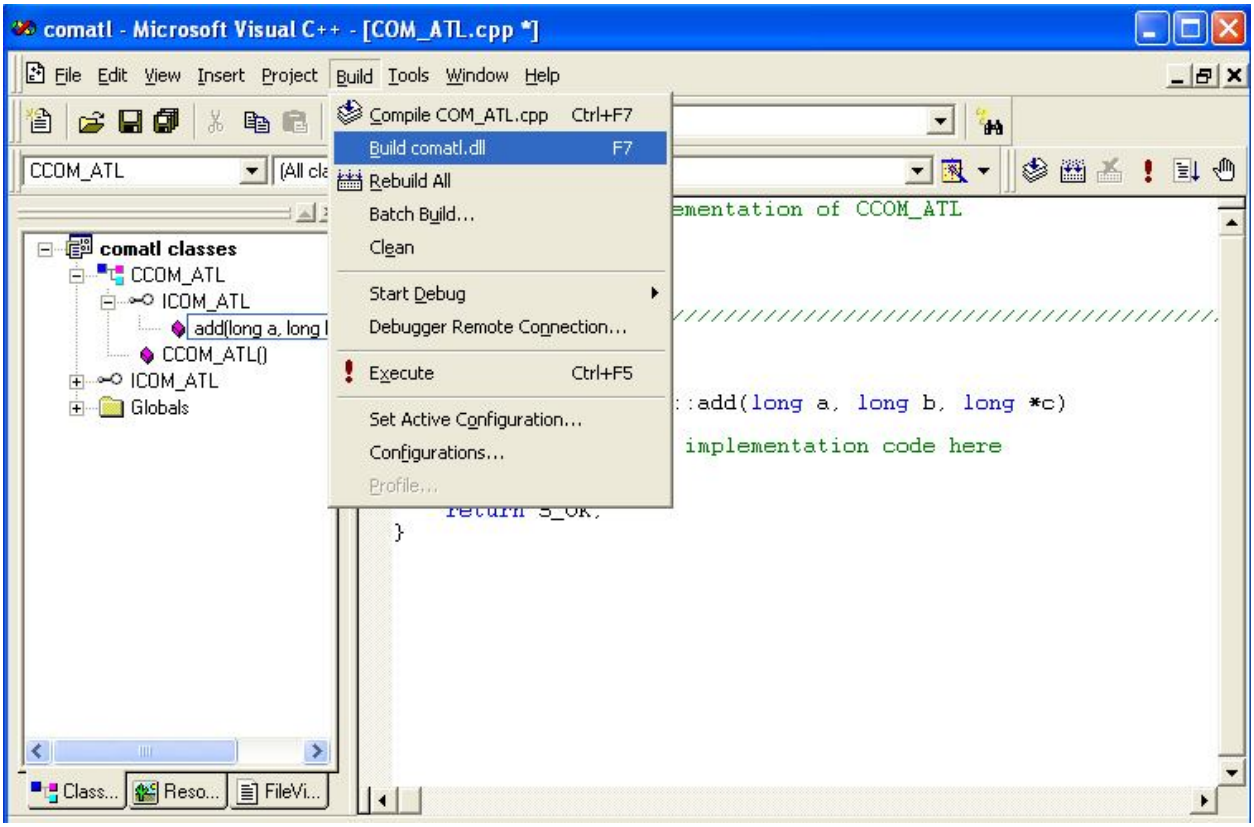

comatl - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
DllCanUnloadNow
comatl classes
CCOM_ATL
ICOM_ATL
CCOM_ATL()
ICOM_ATL
Globals
Class...
Reso...
FileVi...
Add Method to Interface
Return Type:
HRESULT
Method Name:
add
Parameters:
long a, long b, long *c
Implementation:
[id(1), helpstring("method add")]
HRESULT add(long a, long b, long *c);
OK
Cancel
Attributes...

Step 9:



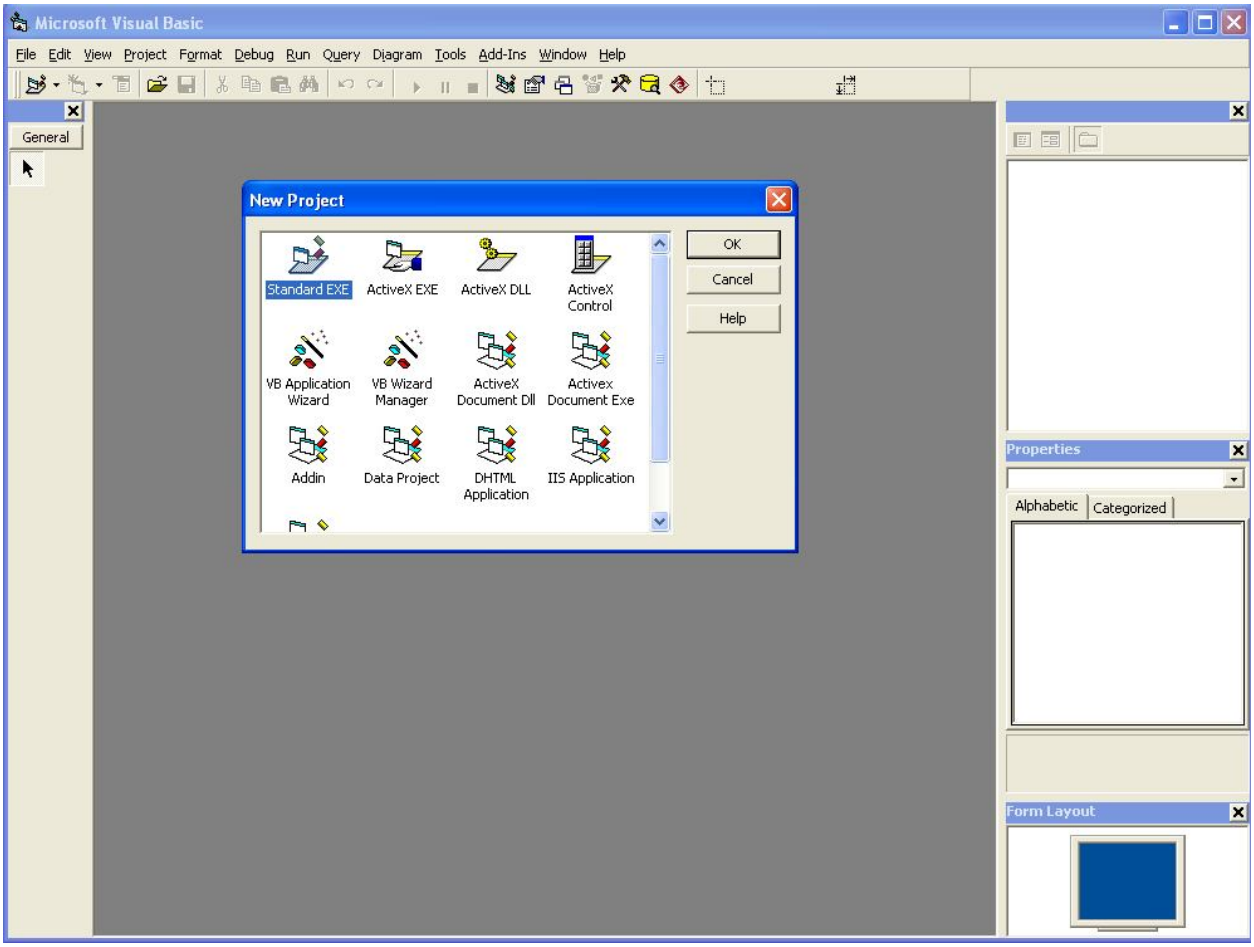
Step 10:

Save, compile and build the MY_ATL application

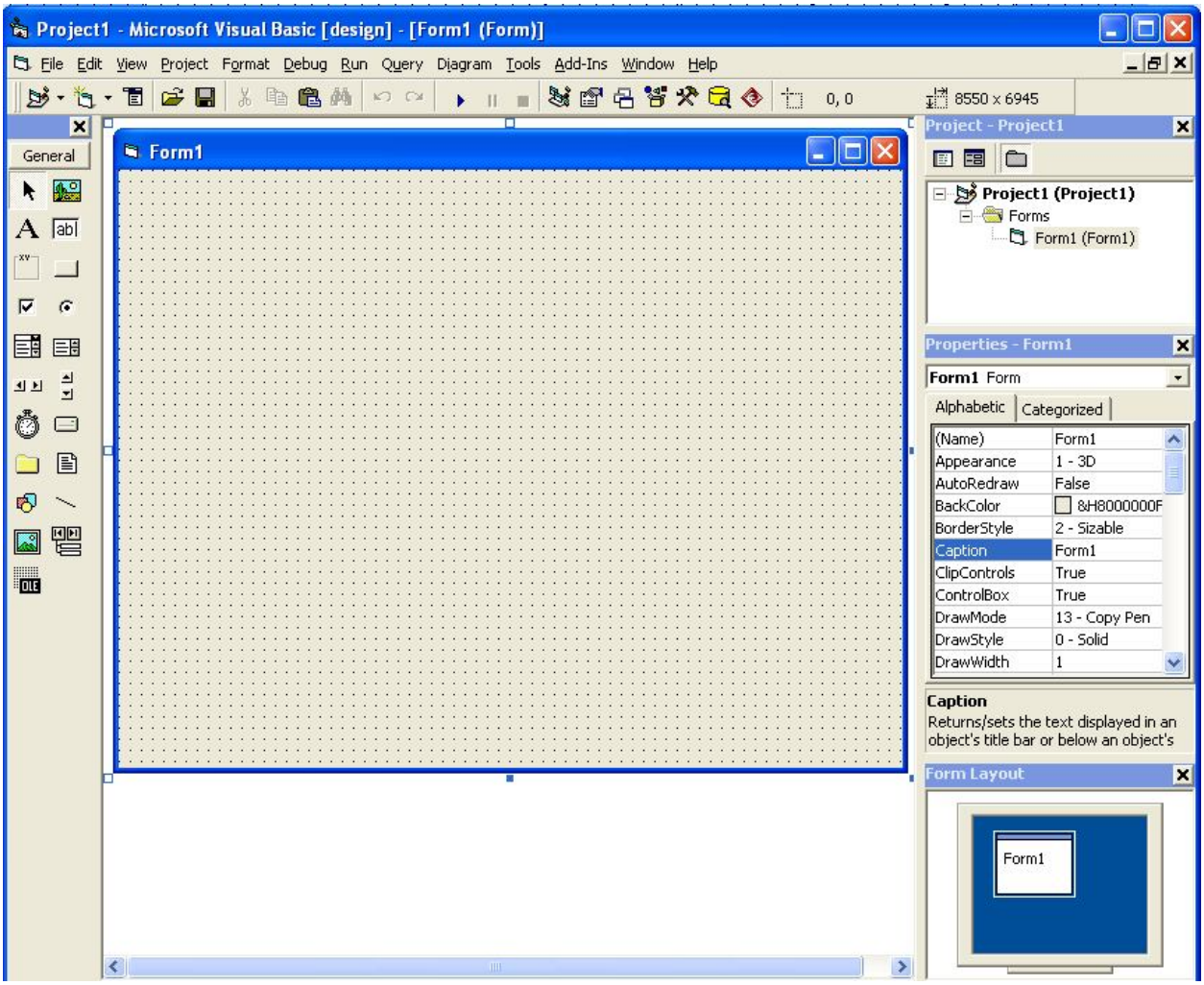


Steps to test the COM component using VB:

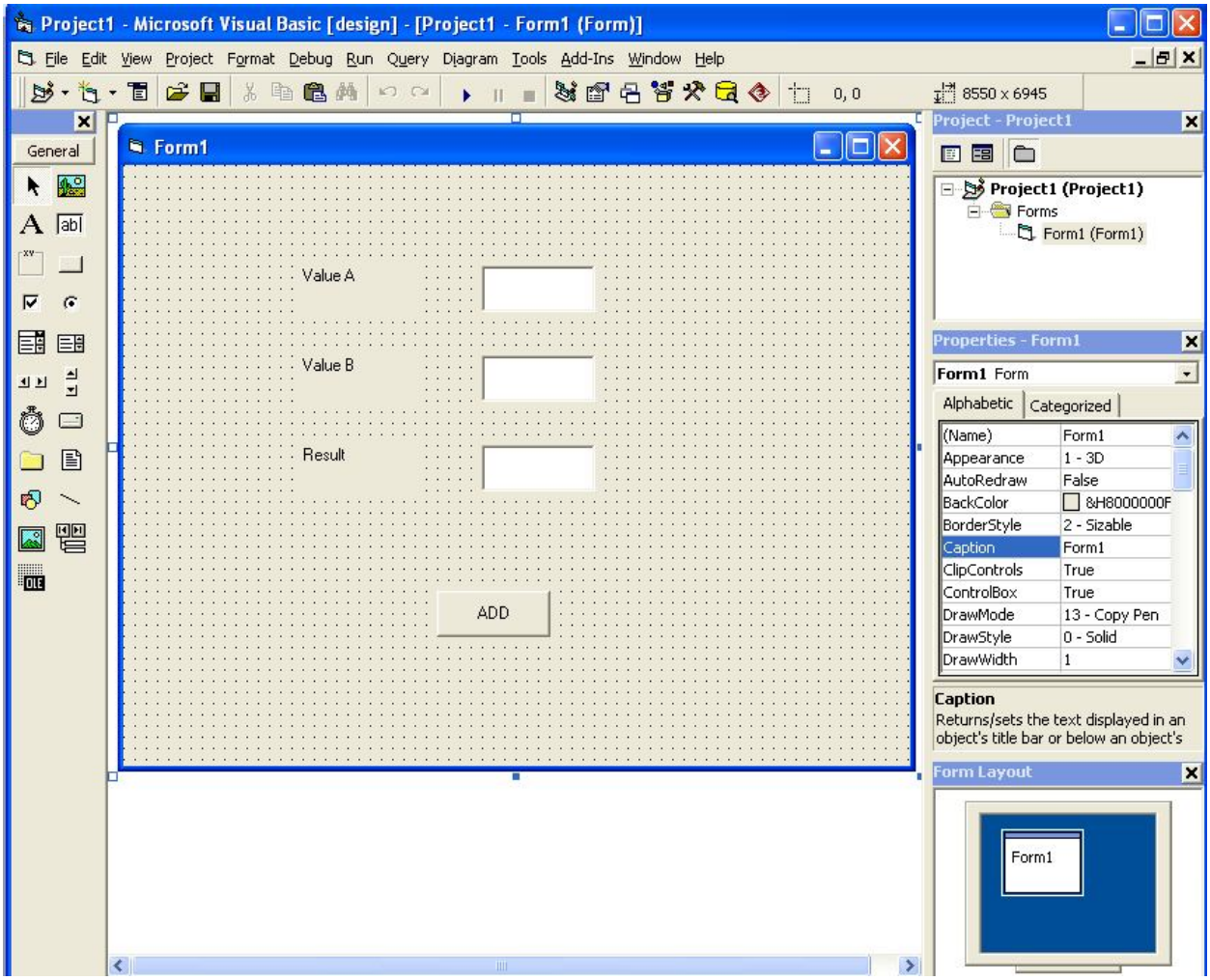
Step 11:


Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
New Project
Standard EXE
ActiveX EXE
ActiveX DLL
ActiveX Control
VB Application Wizard
VB Wizard Manager
ActiveX Document Dll
Activex Document Exe
Addin
Data Project
DHTML Application
IIS Application
OK
Cancel
Help
Properties
Alphabetic
Categorized
Form Layout

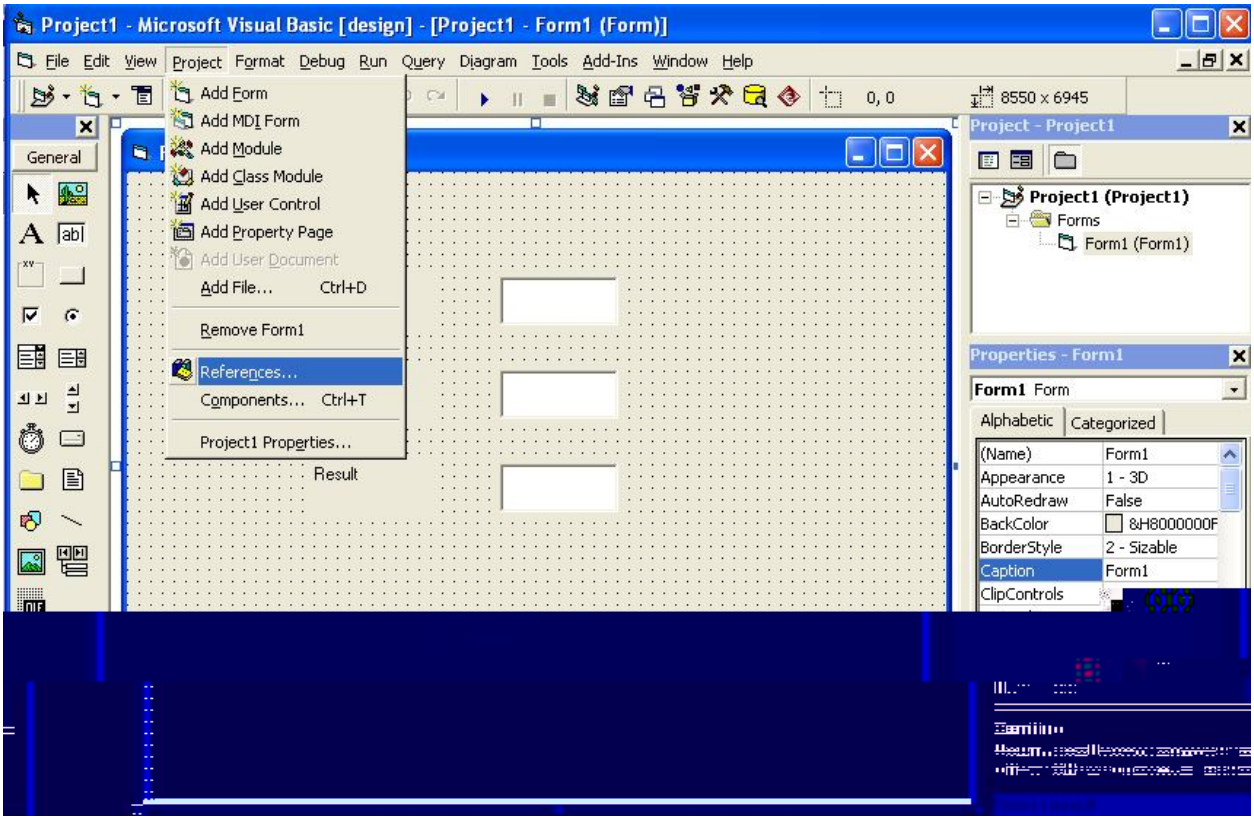
Step 12:


Project1 - Microsoft Visual Basic [design] - [Form1 (Form)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
8550 x 6945
General
Form1
Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Properties - Form1
Form1 Form
Alphabetic
Categorized
(Name) Form1
Appearance 1 - 3D
AutoRedraw False
BackColor &H8000000F
BorderStyle 2 - Sizable
Caption Form1
ClipControls True
ControlBox True
DrawMode 13 - Copy Pen
DrawStyle 0 - Solid
DrawWidth 1
Caption
Returns/sets the text displayed in an object's title bar or below an object's
Form Layout
Form1

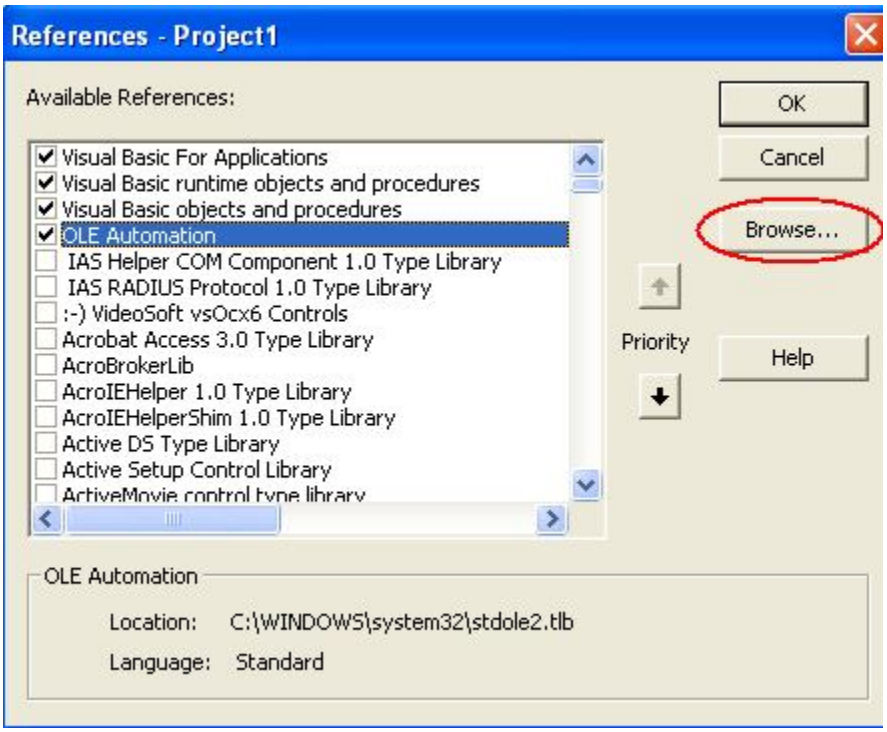
Step 13:


Project1 - Microsoft Visual Basic [design] - [Project1 - Form1 (Form)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
0, 0
8550 x 6945
General
Form1
Value A
Value B
Result
ADD
Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Properties - Form1
Form1 Form
Alphabetic
Categorized
(Name) Form1
Appearance 1 - 3D
AutoRedraw False
BackColor &H8000000F
BorderStyle 2 - Sizable
Caption Form1
ClipControls True
ControlBox True
DrawMode 13 - Copy Pen
DrawStyle 0 - Solid
DrawWidth 1
Caption
Returns/sets the text displayed in an object's title bar or below an object's
Form Layout
Form1

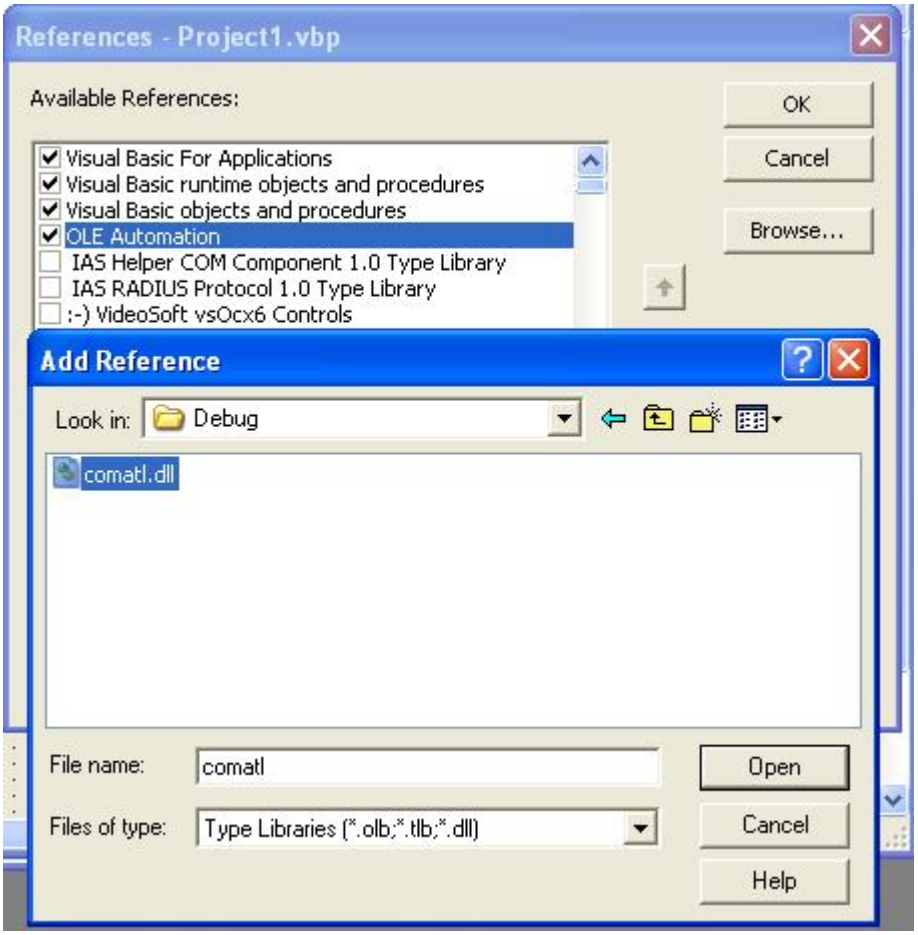
Step 14:


Project1 - Microsoft Visual Basic [design] - [Project1 - Form1 (Form)]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Add Form
Add MDI Form
Add Module
Add Class Module
Add User Control
Add Property Page
Add User Document
Add File... Ctrl+D
Remove Form1
References...
Components... Ctrl+T
Project1 Properties...
General
Result
0, 0
8550 x 6945
Project - Project1
Project1 (Project1)
Forms
Form1 (Form1)
Properties - Form1
Form1 Form
Alphabetic
Categorized
(Name) Form1
Appearance 1 - 3D
AutoRedraw False
BackColor &H8000000F
BorderStyle 2 - Sizable
Caption Form1
ClipControls

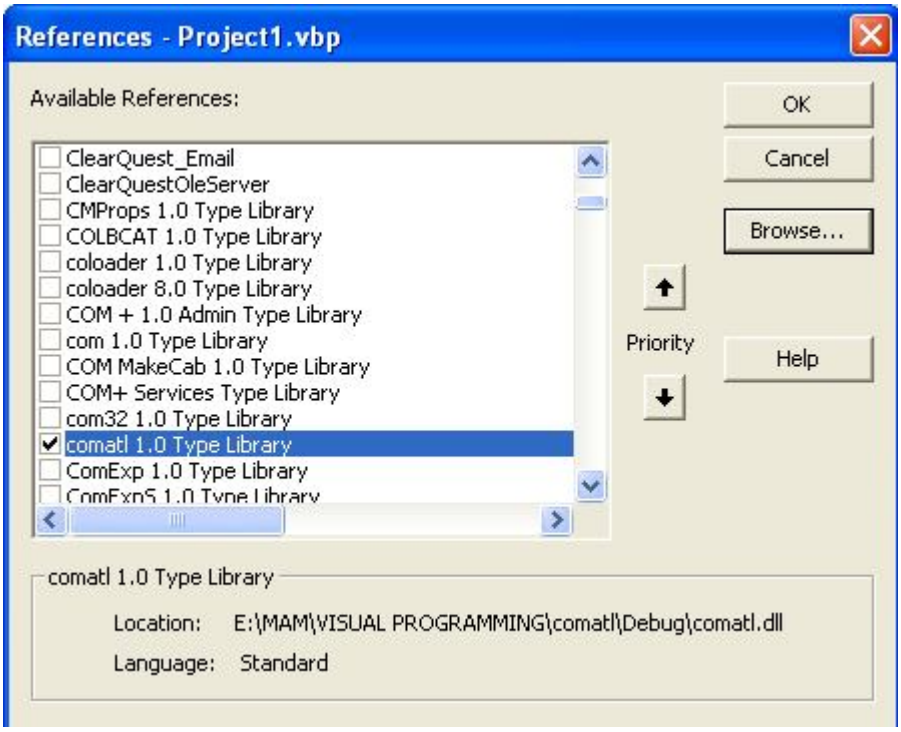
Step 15:


References - Project1
Available References:
Visual Basic For Applications
Visual Basic runtime objects and procedures
Visual Basic objects and procedures
OLE Automation
IAS Helper COM Component 1.0 Type Library
IAS RADIUS Protocol 1.0 Type Library
:-) VideoSoft vsOcx6 Controls
Acrobat Access 3.0 Type Library
AcroBrokerLib
AcroIEHelper 1.0 Type Library
AcroIEHelperShim 1.0 Type Library
Active DS Type Library
Active Setup Control Library
OK
Cancel
Browse...
Priority
Help
OLE Automation
Location: C:\WINDOWS\system32\stdole2.tlb
Language: Standard

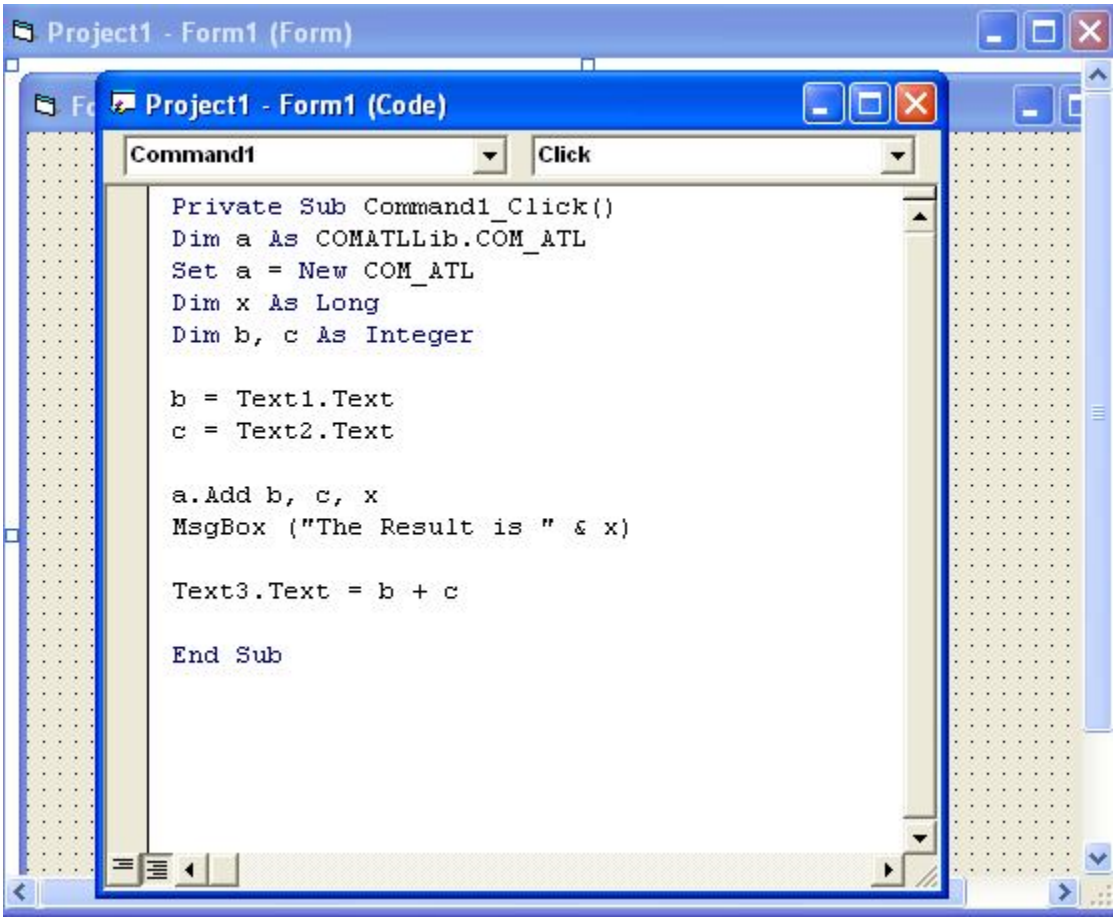
Step 16:



Step 17:


References - Project1.vbp
Available References:
ClearQuest_Email
ClearQuestOleServer
CMProps 1.0 Type Library
COLBCAT 1.0 Type Library
coloader 1.0 Type Library
coloader 8.0 Type Library
COM + 1.0 Admin Type Library
com 1.0 Type Library
COM MakeCab 1.0 Type Library
COM+ Services Type Library
com32 1.0 Type Library
comatl 1.0 Type Library
ComExp 1.0 Type Library
OK
Cancel
Browse...
Priority
Help
comatl 1.0 Type Library
Location: E:\MAM\VISUAL PROGRAMMING\comatl\Debug\comatl.dll
Language: Standard

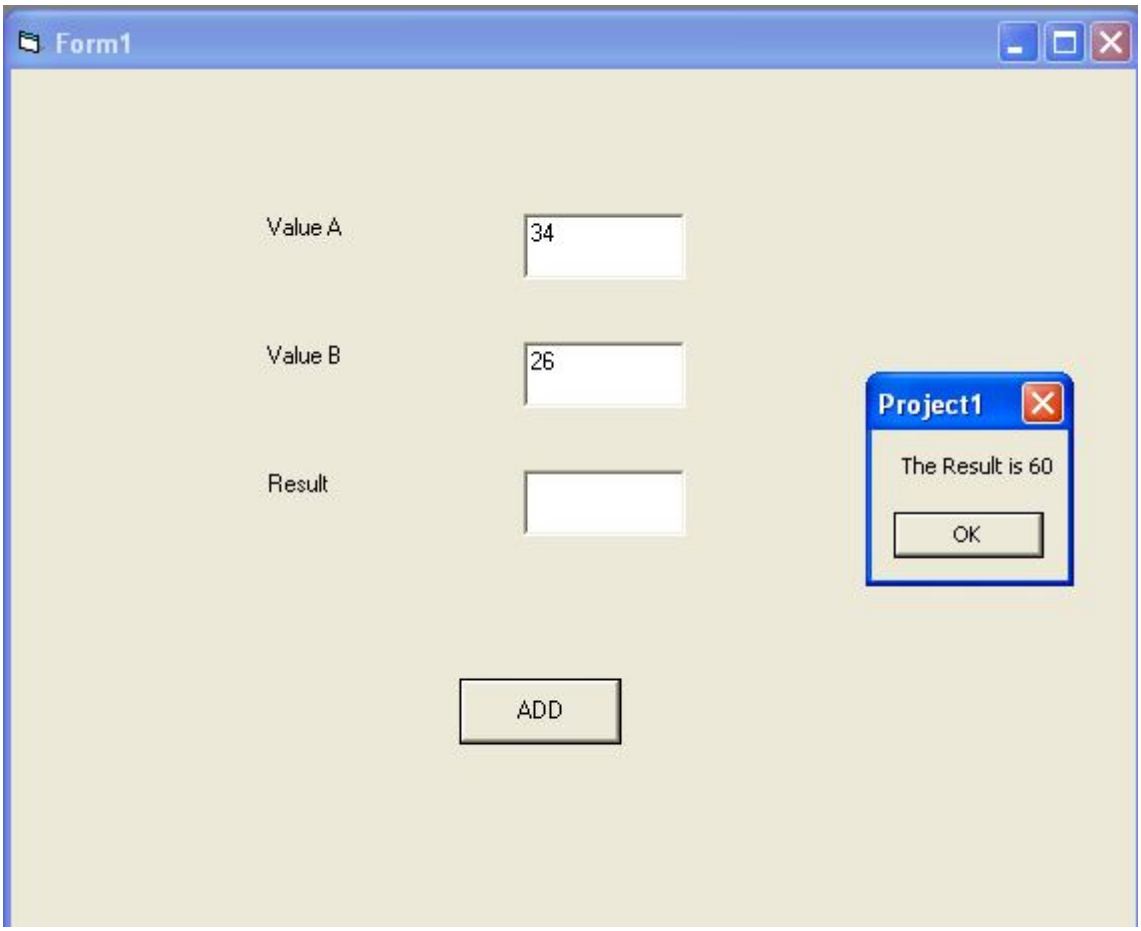
Step 18:

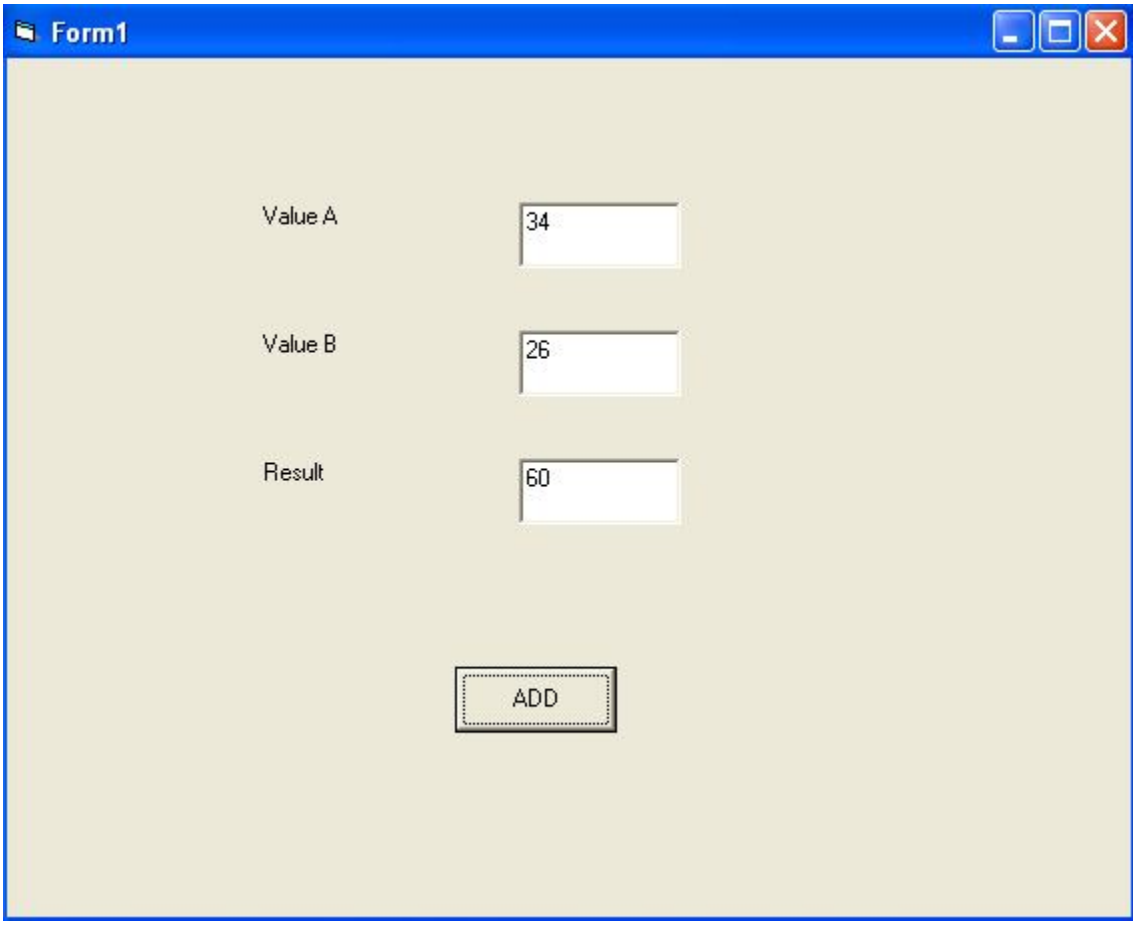


Step 19:

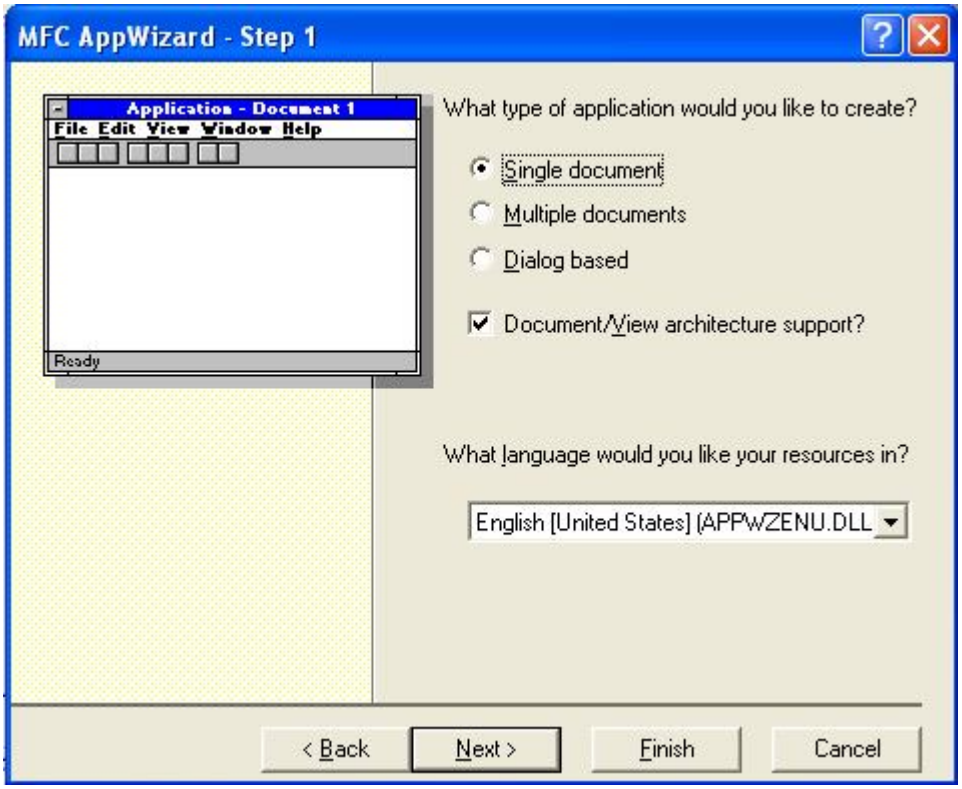
Step 20:

Output:

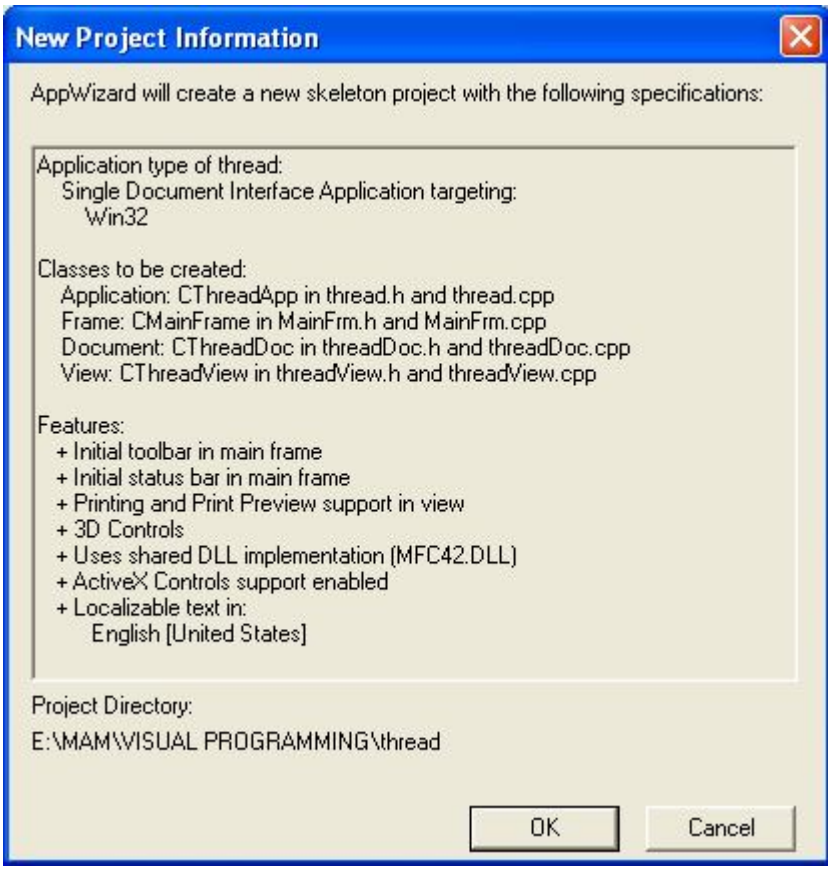

Form1
Value A
34
Value B
26
Result
ADD
Project1
The Result is 60
OK



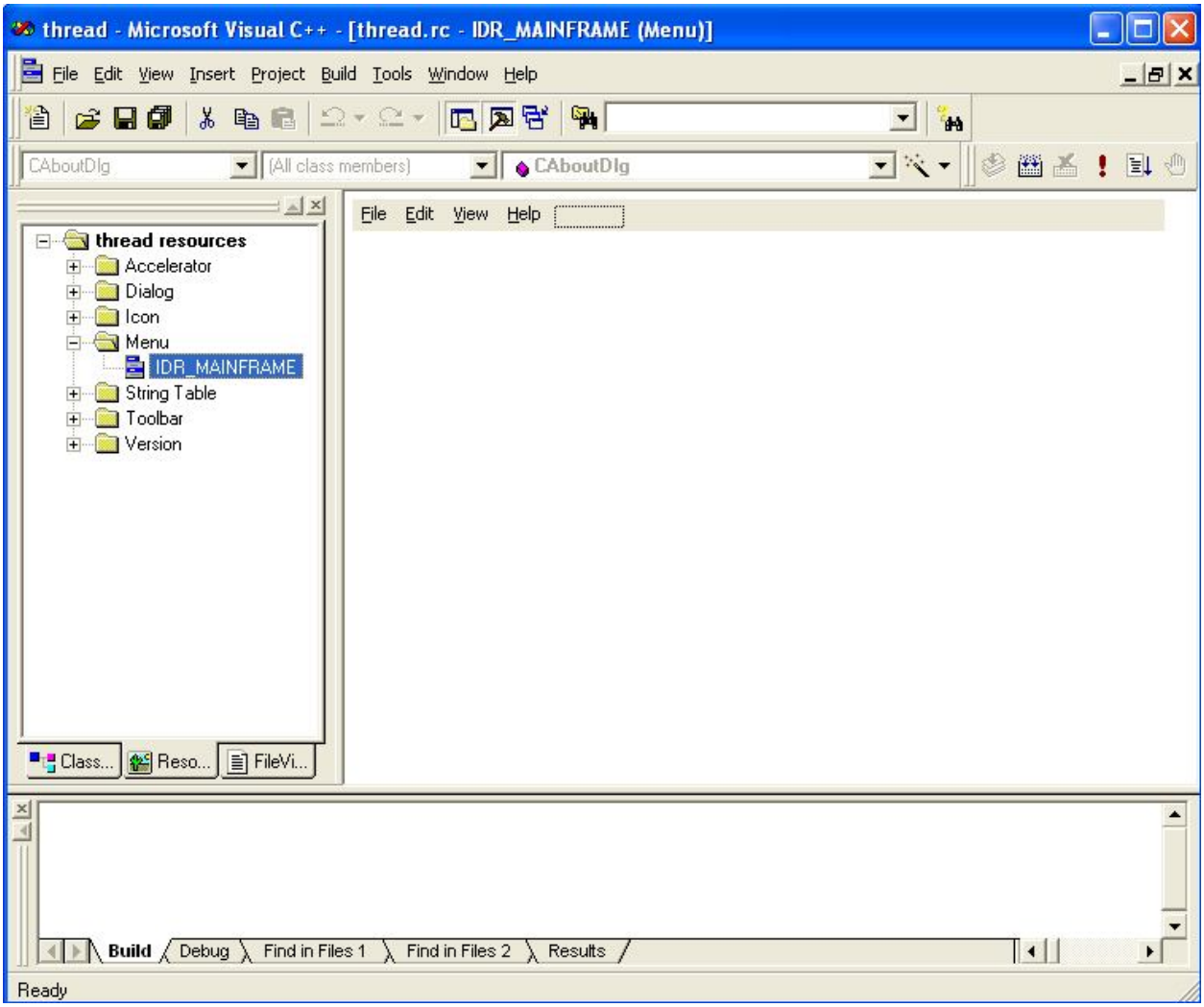
Result:


MFC AppWizard - Step 1
Application - Document 1
File Edit View Window Help
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

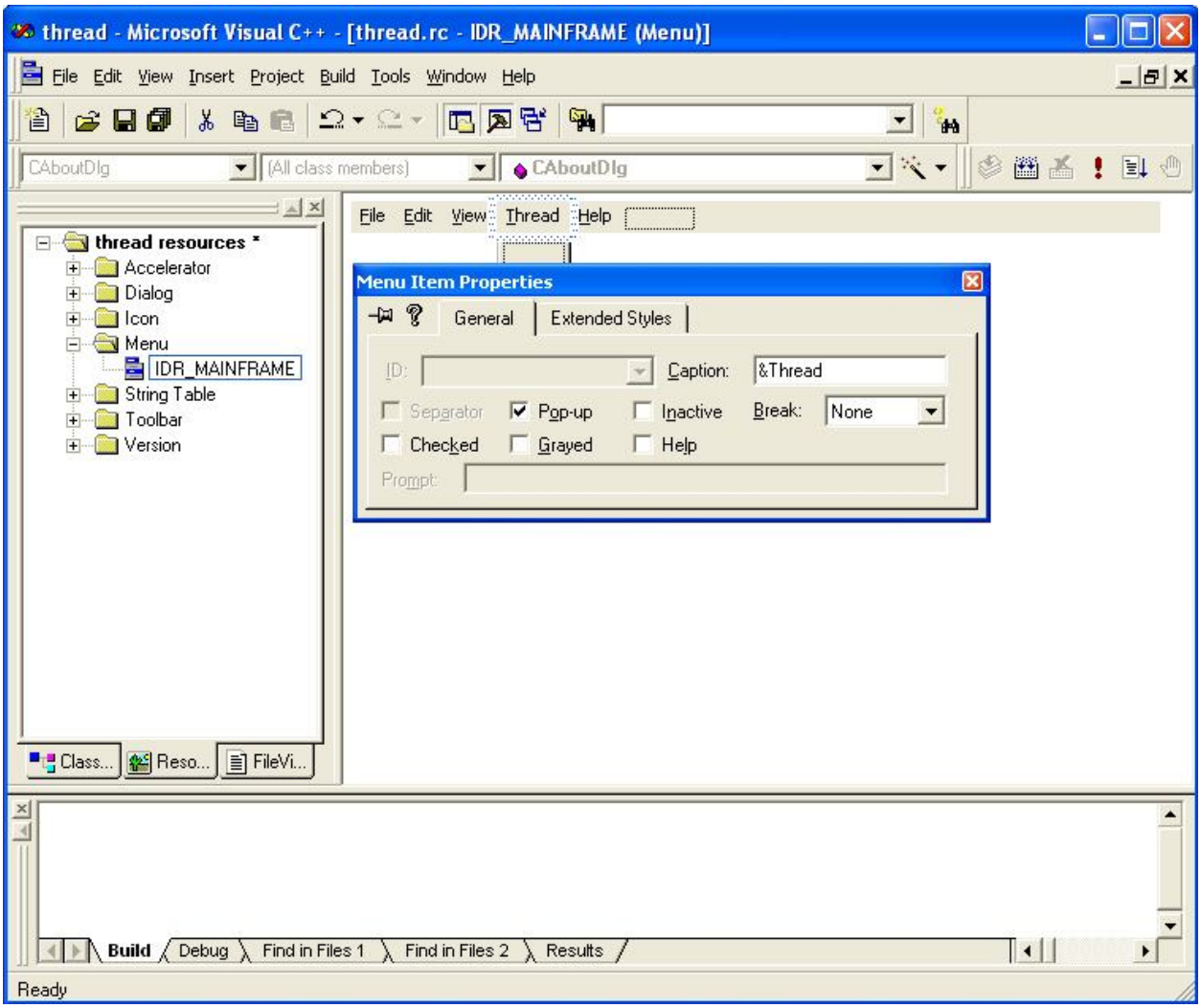
Step 3:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of thread:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CThreadApp in thread.h and thread.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CThreadDoc in threadDoc.h and threadDoc.cpp
View: CThreadView in threadView.h and threadView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
E:\MAM\VISUAL PROGRAMMING\thread
OK
Cancel

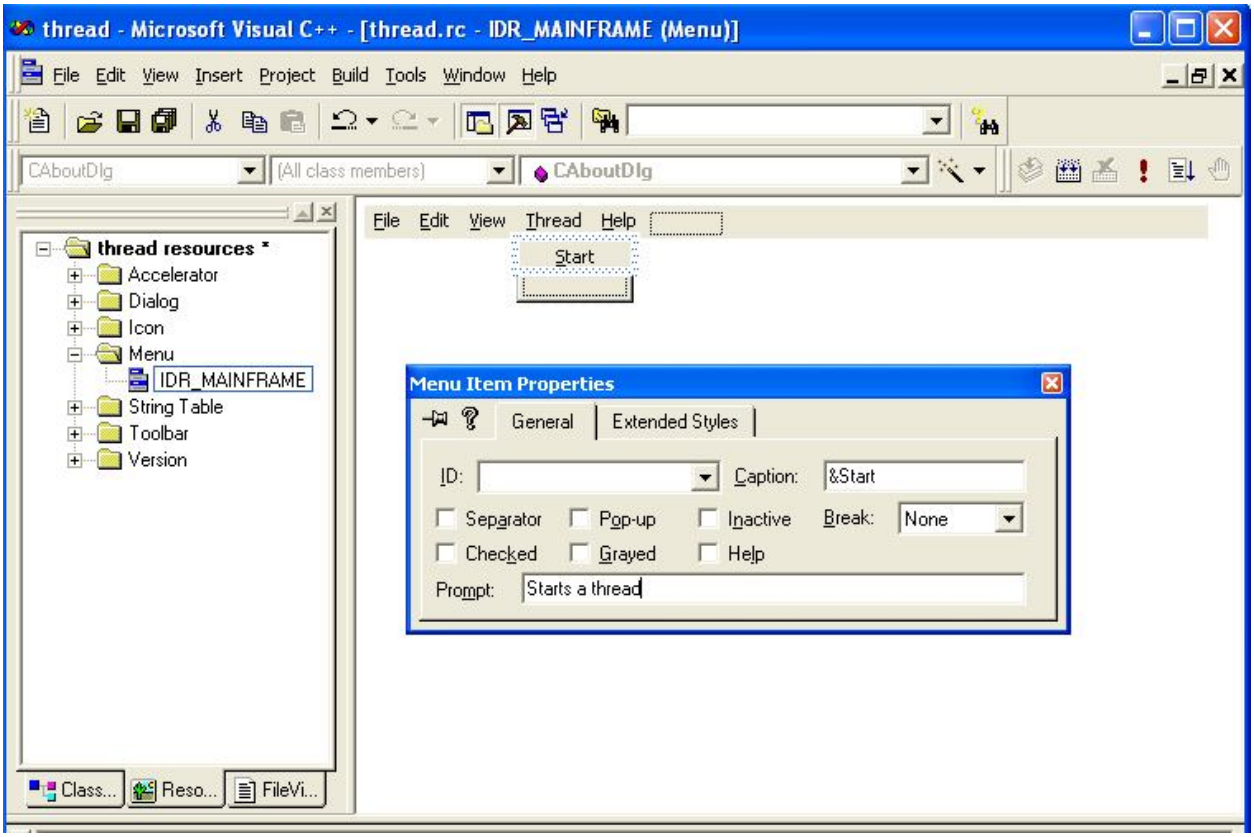
Step 4:


thread - Microsoft Visual C++ - [thread.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
thread resources
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
File Edit View Help
Class...
Reso...
FileVi...
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

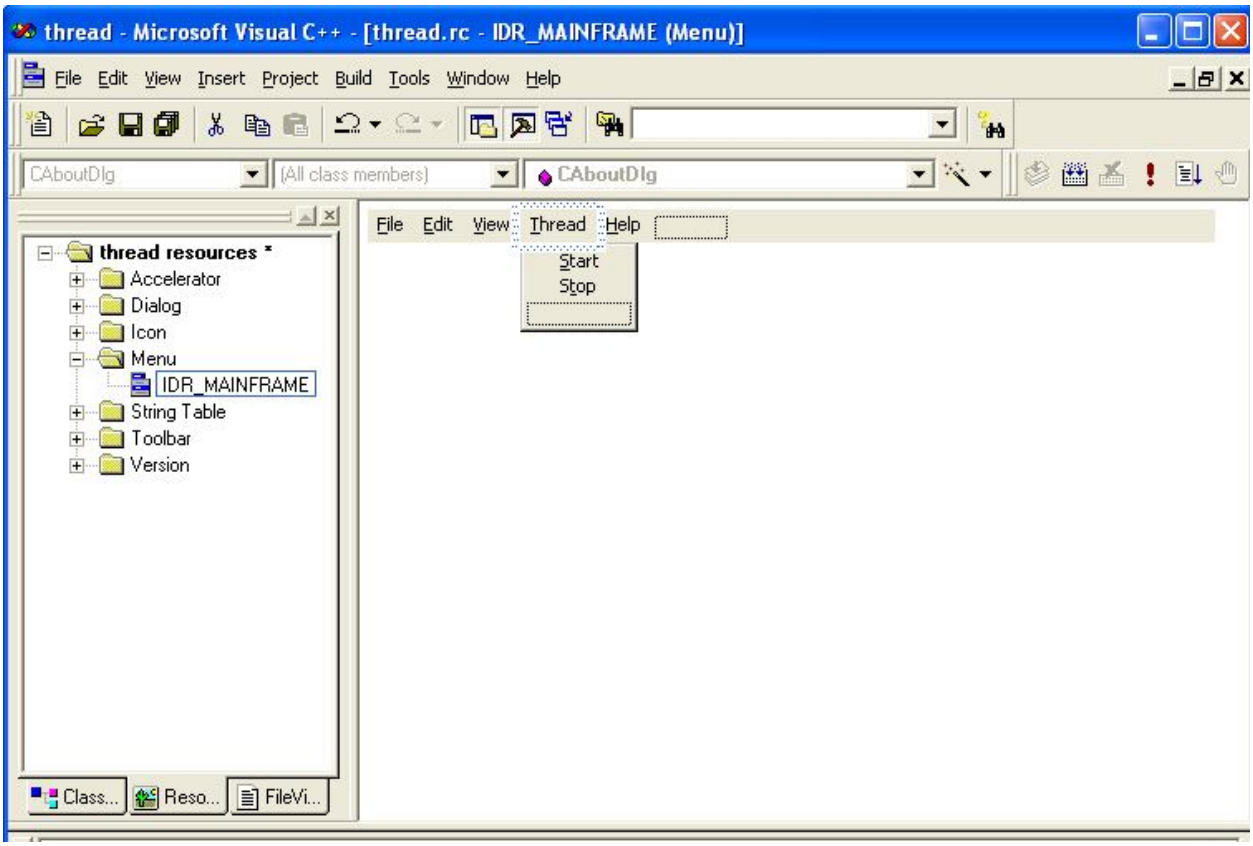
Step 5:


thread - Microsoft Visual C++ - [thread.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
thread resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Thread Help
Menu Item Properties
General
Extended Styles
ID:
Caption:
&Thread
Separator
Pop-up
Inactive
Break:
None
Checked
Grayed
Help
Prompt:
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

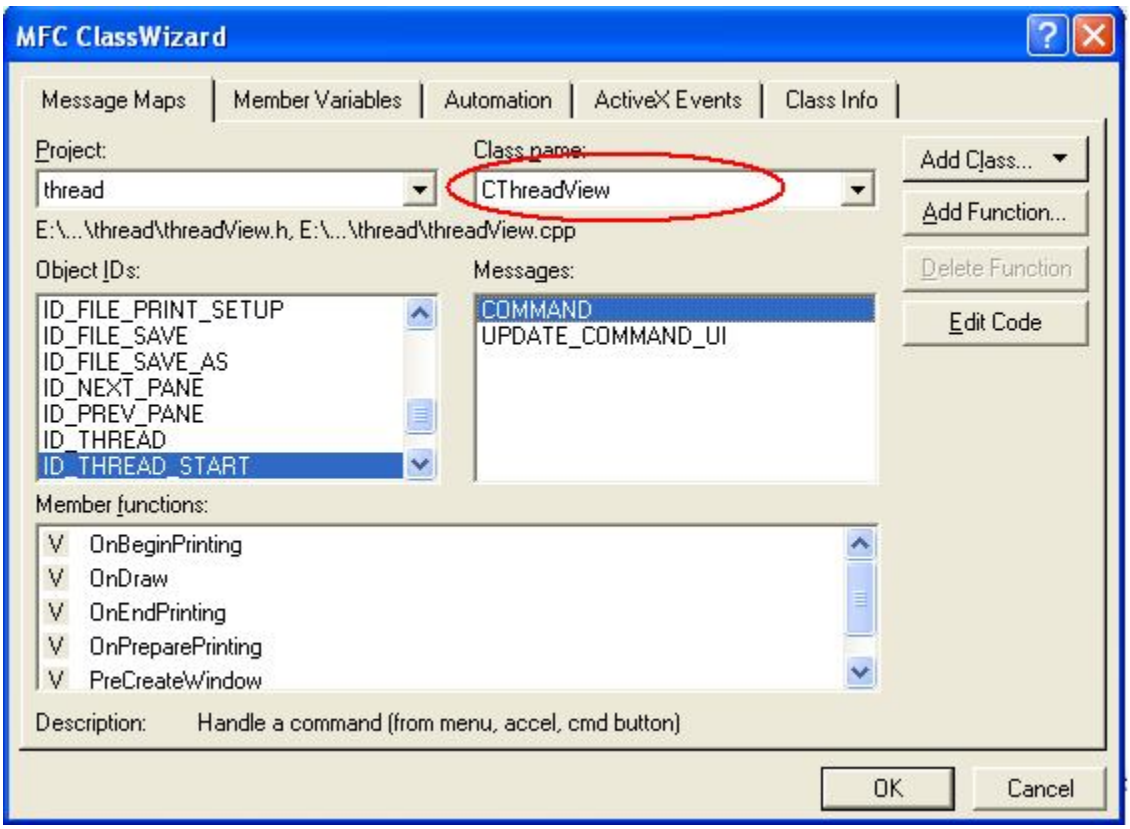
Step 6:


thread - Microsoft Visual C++ - [thread.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
thread resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Thread Help
Start
Menu Item Properties
General
Extended Styles
ID:
Caption:
&Start
Separator
Pop-up
Inactive
Break:
None
Checked
Grayed
Help
Prompt:
Starts a thread

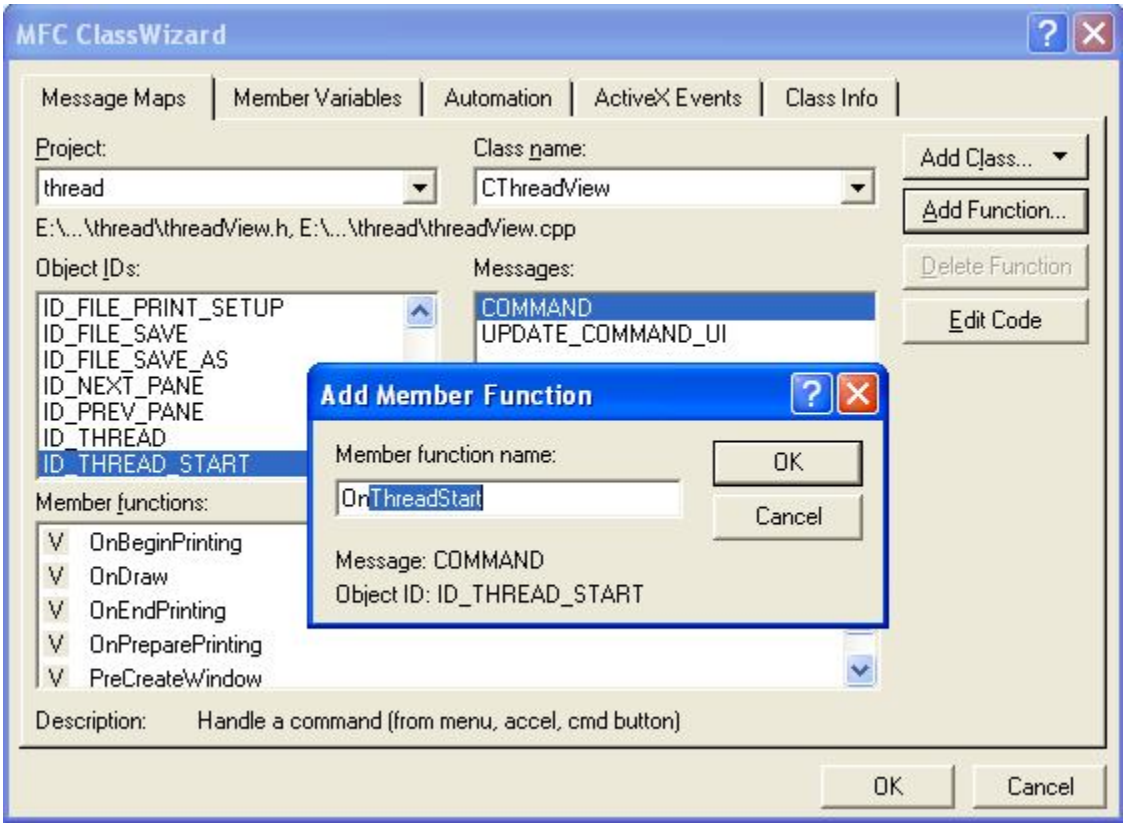
Step 7:


thread - Microsoft Visual C++ - [thread.rc - IDR_MAINFRAME (Menu)]
File Edit View Insert Project Build Tools Window Help
CAboutDlg
(All class members)
CAboutDlg
thread resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
File Edit View Thread Help
Start
Stop

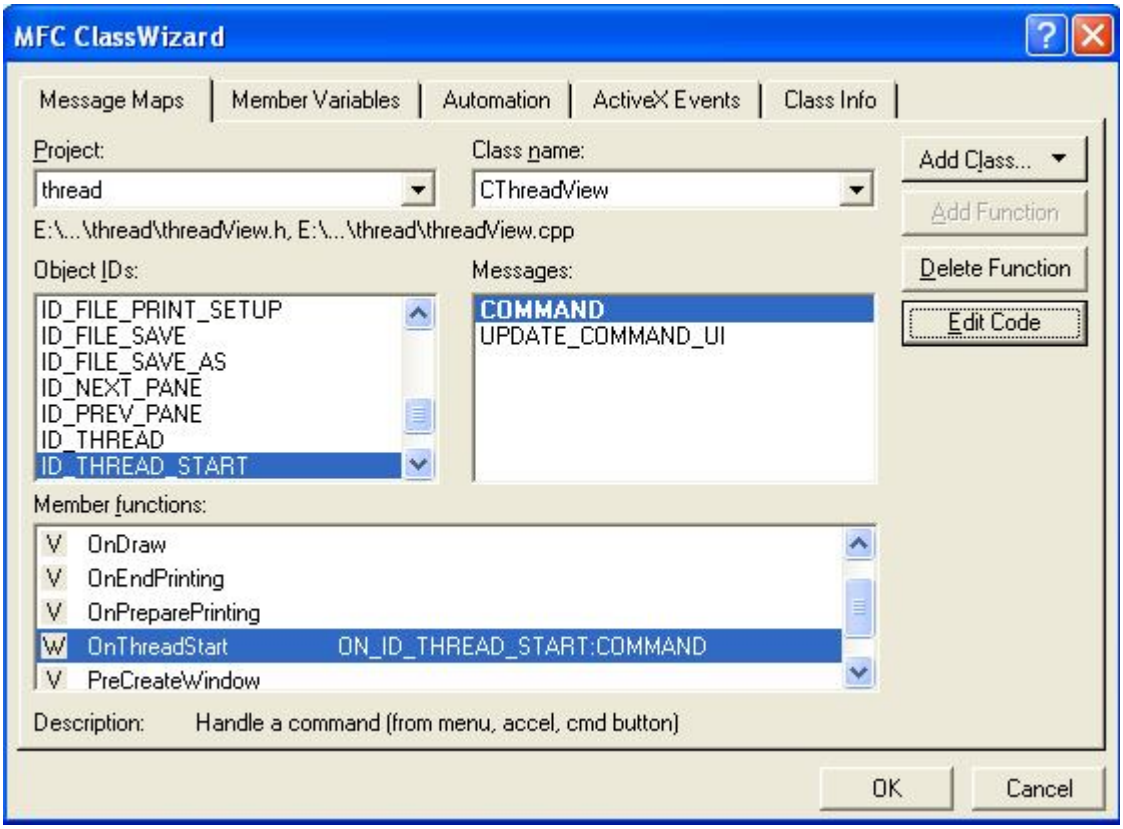
Step 8:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
thread
Class name:
CThreadView
E:\...\thread\threadView.h, E:\...\thread\threadView.cpp
Object IDs:
ID_FILE_PRINT_SETUP
ID_FILE_SAVE
ID_FILE_SAVE_AS
ID_NEXT_PANE
ID_PREV_PANE
ID_THREAD
ID_THREAD_START
Messages:
COMMAND
UPDATE_COMMAND_UI
Add Class...
Add Function...
Delete Function
Edit Code
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
V OnPreparePrinting
V PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

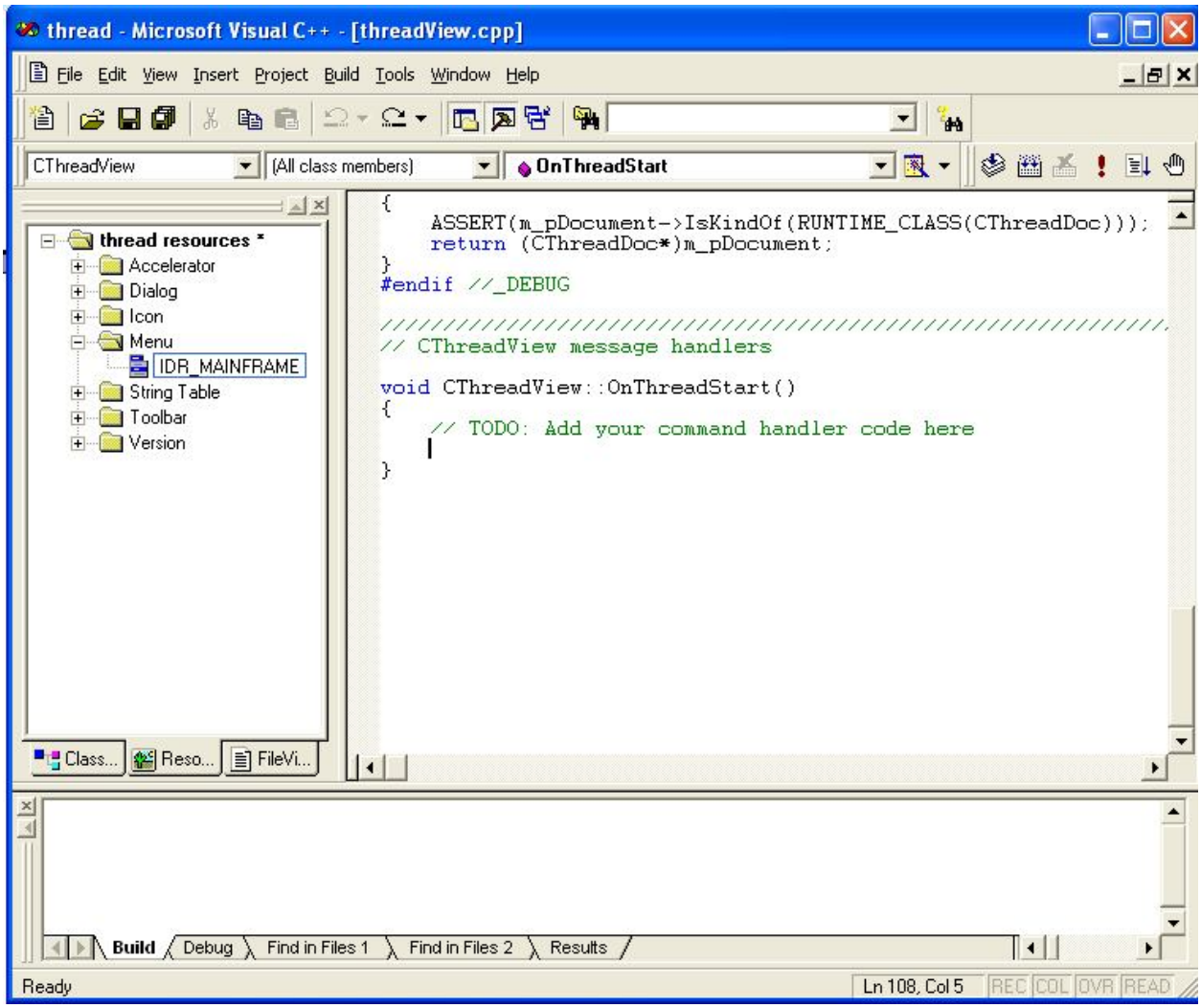
Step 9:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
thread
Class name:
CThreadView
Add Class...
Add Function...
Delete Function
Edit Code
E:\...\thread\threadView.h, E:\...\thread\threadView.cpp
Object IDs:
ID_FILE_PRINT_SETUP
ID_FILE_SAVE
ID_FILE_SAVE_AS
ID_NEXT_PANE
ID_PREV_PANE
ID_THREAD
ID_THREAD_START
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
V OnBeginPrinting
V OnDraw
V OnEndPrinting
V OnPreparePrinting
V PreCreateWindow
Add Member Function
Member function name:
OnThreadStart
OK
Cancel
Message: COMMAND
Object ID: ID_THREAD_START
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

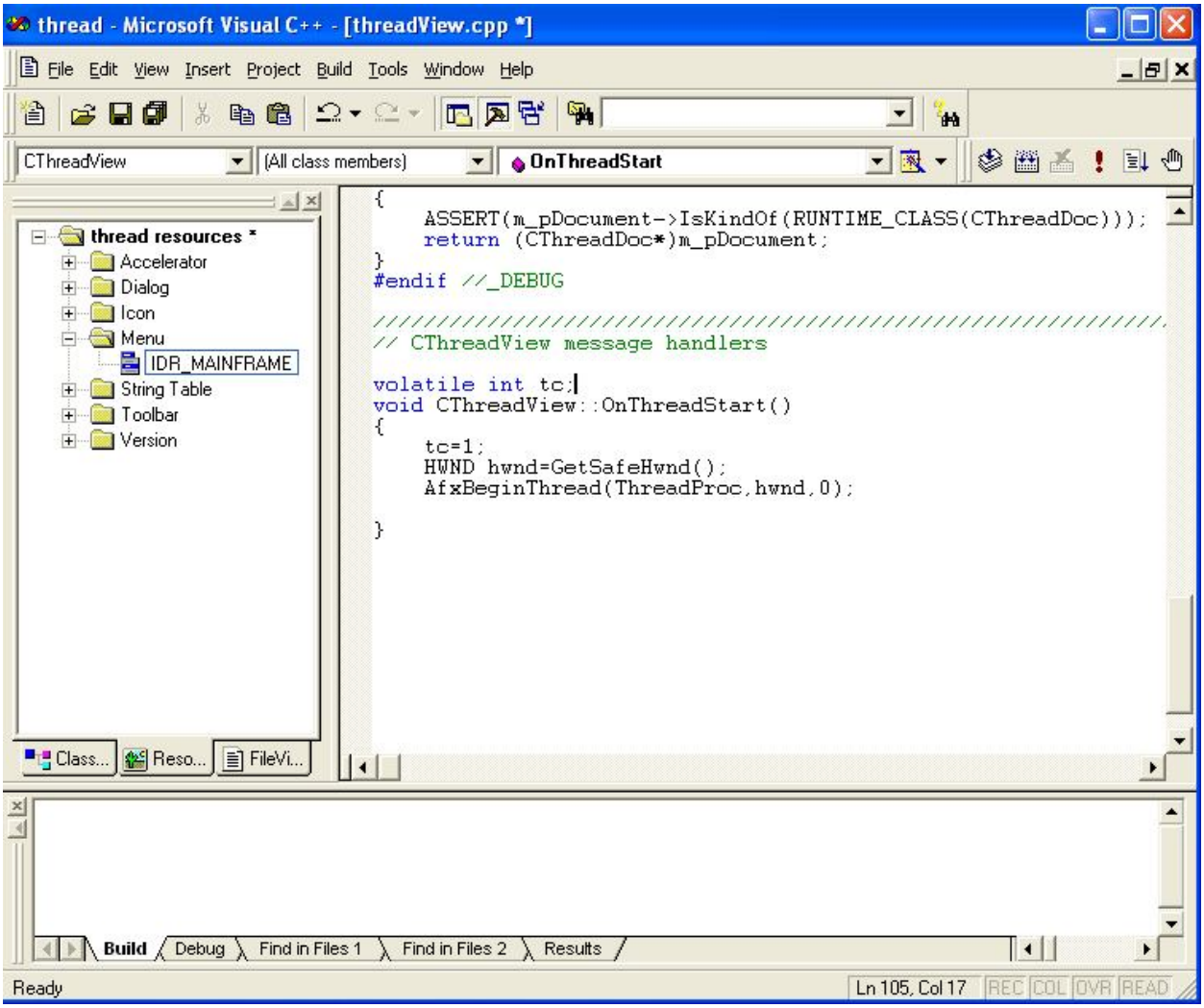
Step 10:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
thread
Class name:
CThreadView
Add Class...
Add Function
Delete Function
Edit Code
E:\...\thread\threadView.h, E:\...\thread\threadView.cpp
Object IDs:
ID_FILE_PRINT_SETUP
ID_FILE_SAVE
ID_FILE_SAVE_AS
ID_NEXT_PANE
ID_PREV_PANE
ID_THREAD
ID_THREAD_START
Messages:
COMMAND
UPDATE_COMMAND_UI
Member functions:
OnDraw
OnEndPrinting
OnPreparePrinting
OnThreadStart ON_ID_THREAD_START:COMMAND
PreCreateWindow
Description: Handle a command (from menu, accel, cmd button)
OK
Cancel

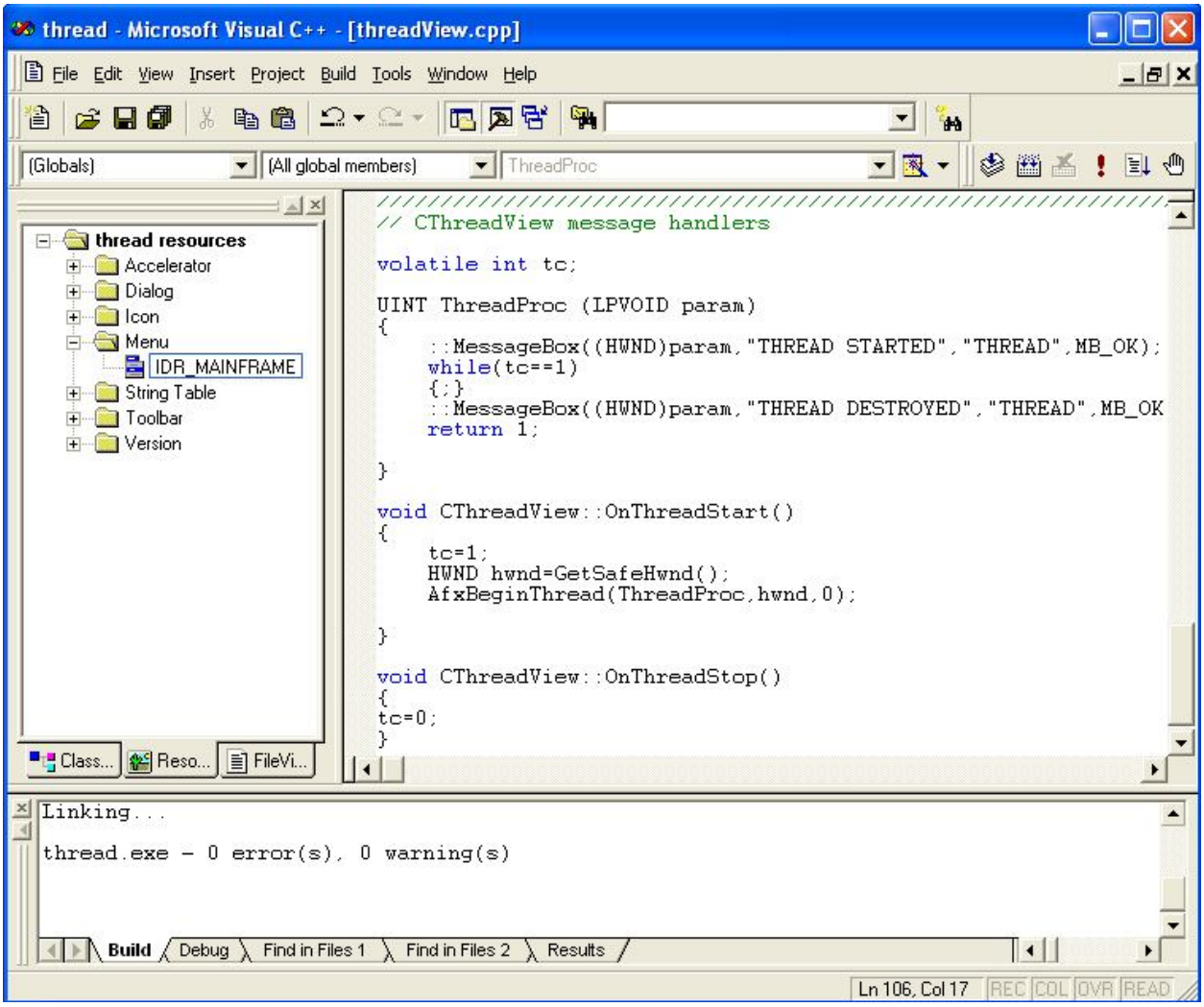
Step 11:


thread - Microsoft Visual C++ - [threadView.cpp]
File Edit View Insert Project Build Tools Window Help
CThreadView
(All class members)
OnThreadStart
thread resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
{
ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CThreadDoc)));
return (CThreadDoc*)m_pDocument;
}
#endif //_DEBUG
// CThreadView message handlers
void CThreadView::OnThreadStart()
{
// TODO: Add your command handler code here
}
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
Ln 108, Col 5
REC COL OVR READ

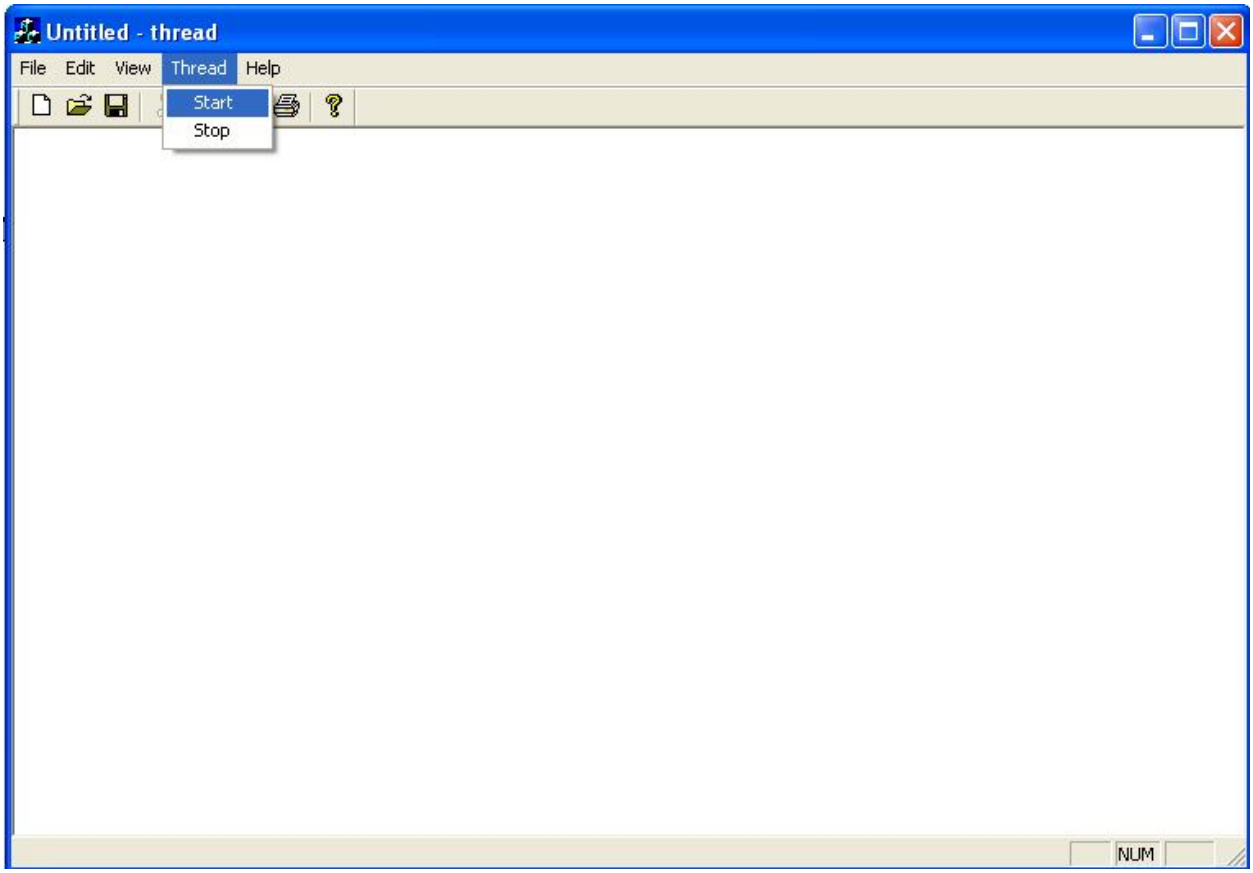
Step 12:

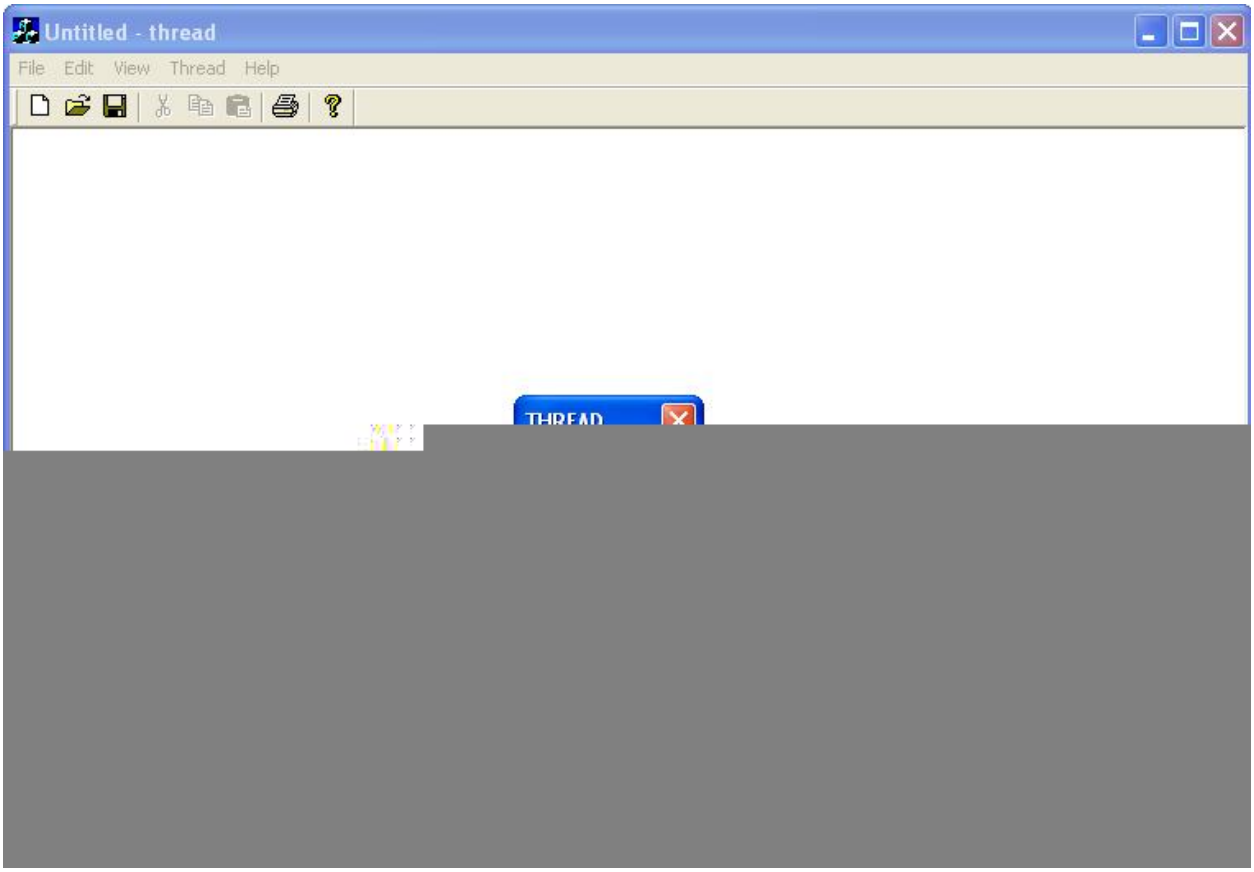

thread - Microsoft Visual C++ - [threadView.cpp *]
File Edit View Insert Project Build Tools Window Help
CThreadView
(All class members)
OnThreadStart
thread resources *
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
{
ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CThreadDoc)));
return (CThreadDoc*)m_pDocument;
}
#endif //_DEBUG
// CThreadView message handlers
volatile int tc;
void CThreadView::OnThreadStart()
{
tc=1;
HWND hwnd=GetSafeHwnd();
AfxBeginThread(ThreadProc,hwnd,0);
}
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
Ln 105, Col 17
REC COL OVR READ

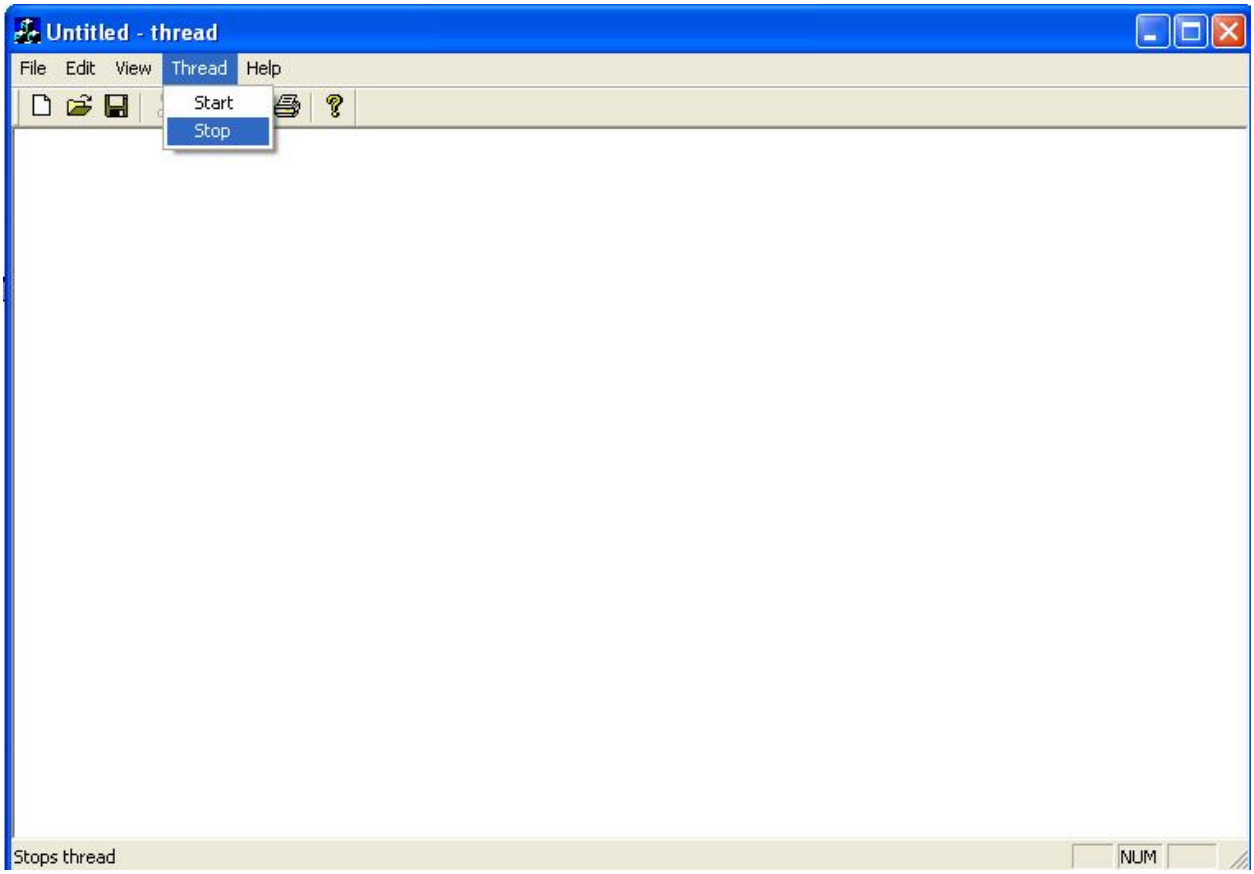
Step 13:

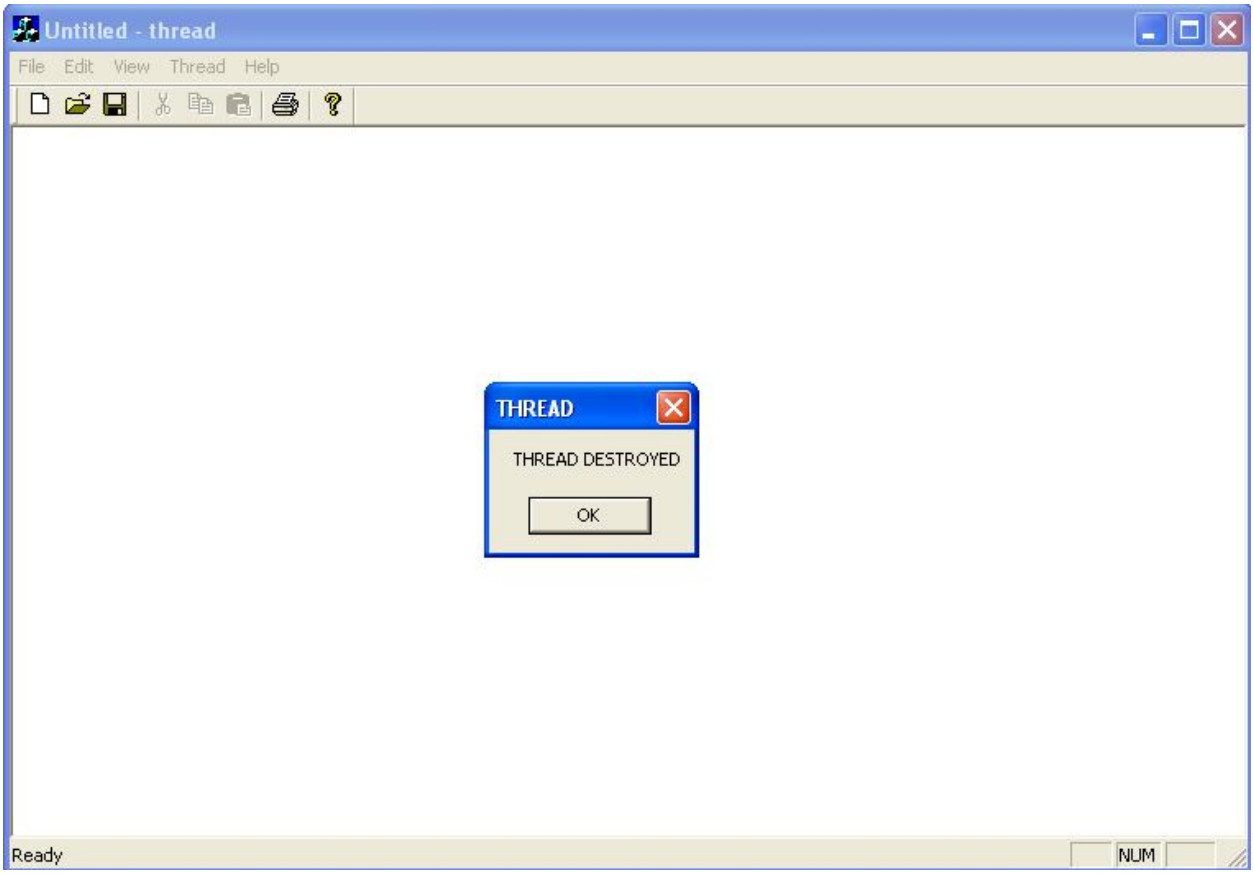

thread - Microsoft Visual C++ - [threadView.cpp]
File Edit View Insert Project Build Tools Window Help
(Globals)
(All global members)
ThreadProc
thread resources
Accelerator
Dialog
Icon
Menu
IDR_MAINFRAME
String Table
Toolbar
Version
Class...
Reso...
FileVi...
// CThreadView message handlers
volatile int tc;
UINT ThreadProc (LPVOID param)
{
::MessageBox((HWND)param,"THREAD STARTED","THREAD",MB_OK);
while(tc==1)
{;}
::MessageBox((HWND)param,"THREAD DESTROYED","THREAD",MB_OK
return 1;
}
void CThreadView::OnThreadStart()
{
tc=1;
HWND hwnd=GetSafeHwnd();
AfxBeginThread(ThreadProc,hwnd,0);
}
void CThreadView::OnThreadStop()
{
tc=0;
}
Linking...
thread.exe - 0 error(s), 0 warning(s)
Build
Debug
Find in Files 1
Find in Files 2
Results
Ln 106, Col 17
REC COL OVR READ

Output:


Untitled - thread
File
Edit
View
Thread
Help
Start
Stop
NUM


Untitled - thread
File Edit View Thread Help
THREAD


Untitled - thread
File Edit View Thread Help
Start
Stop
Stops thread
NUM



Result:

Thus the program to handle thread has been developed, built and executed using MFC application.

14. DATA ACCESS THROUGH ODBC

Aim:

To write a VC++ program to access the Student database through MFC ODBC.

Concept:

Procedure:

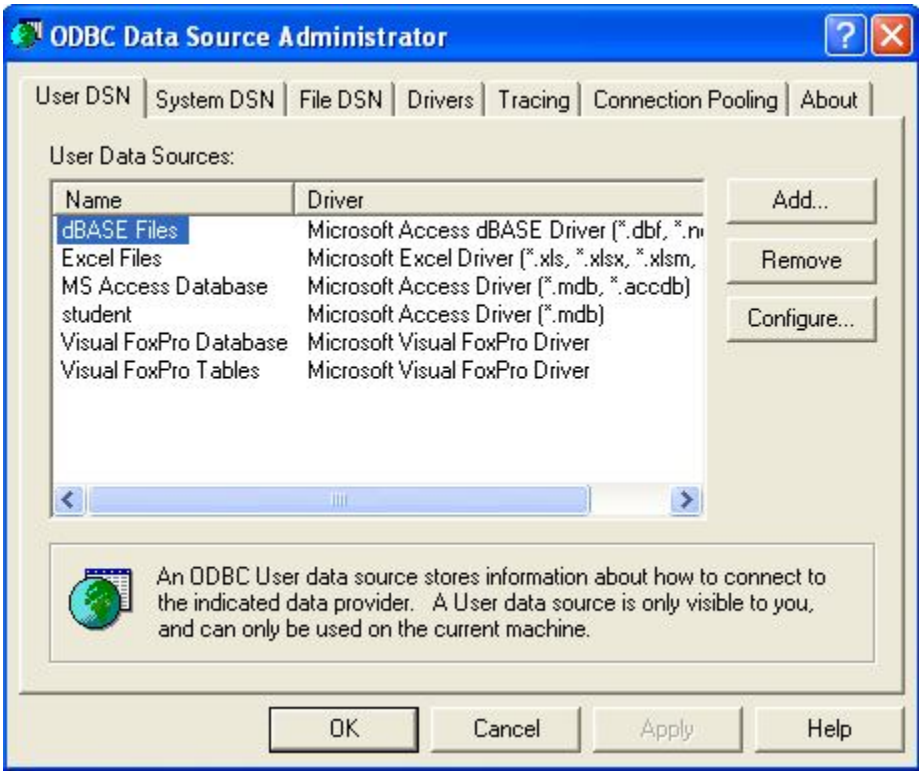
Step 1:

Create the database **student1.mdb** using MS Access with the following table structure

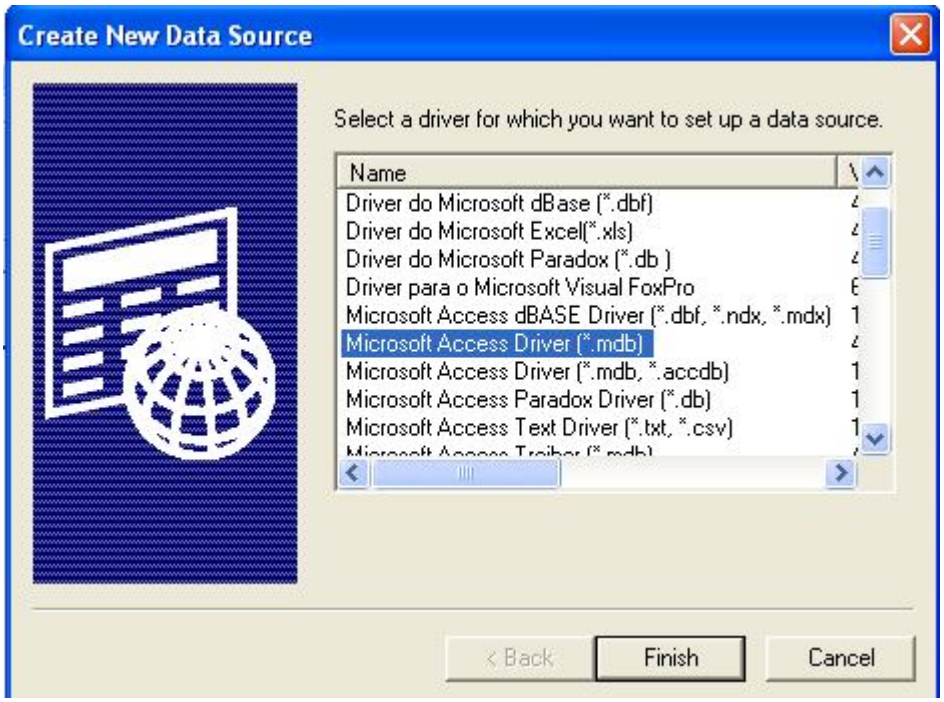
Table name : student
Fields : name (general), rollno (number), mark1 (number), mark2 (number), mark3 (number)

ODBC Connection:

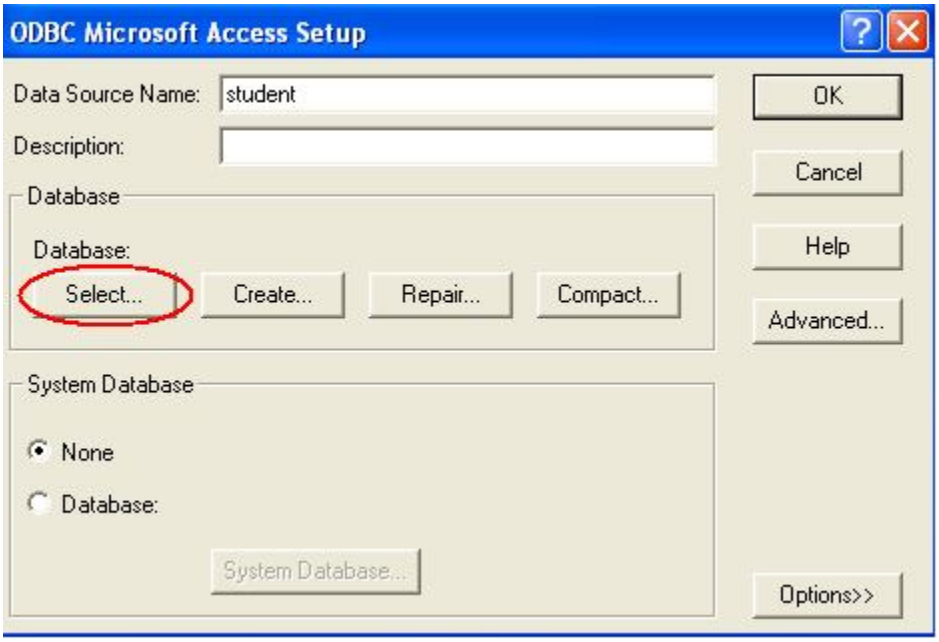
Step 2:



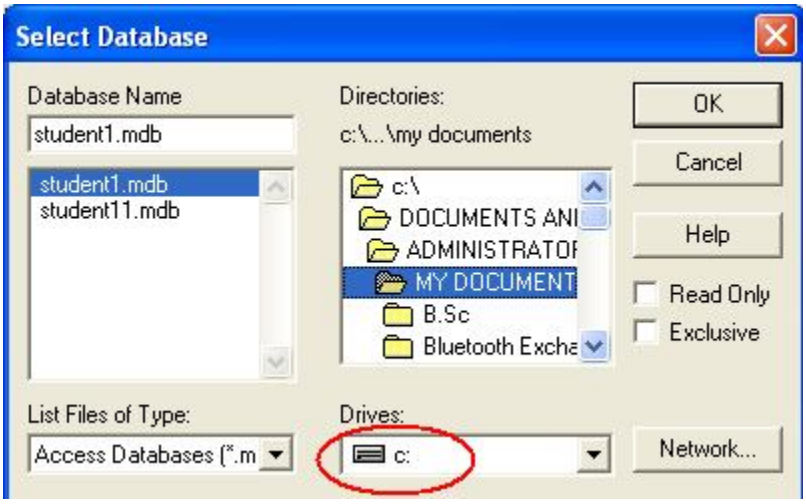
Step 2:



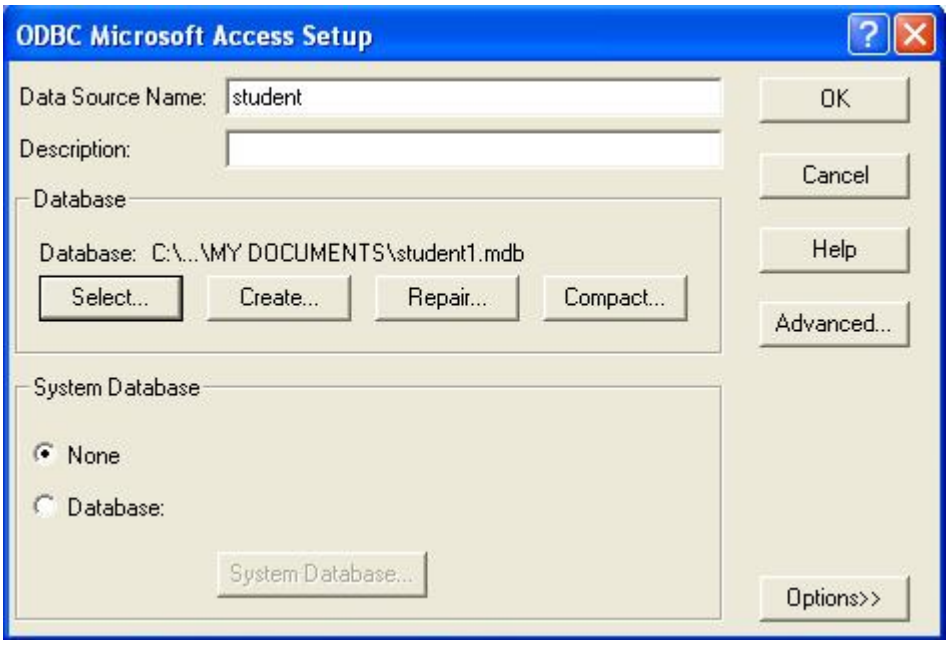
Step 3:



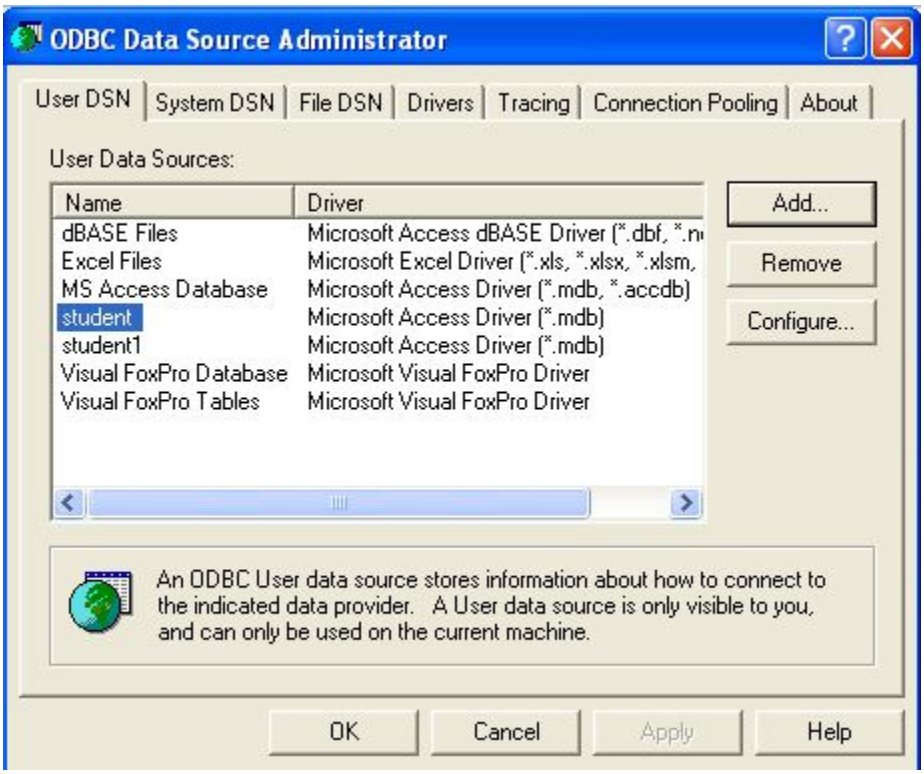
Step 4:


Select Database
Database Name
student1.mdb
student1.mdb
student11.mdb
Directories:
c:\...\my documents
c:\
DOCUMENTS ANI
ADMINISTRATOF
MY DOCUMENT
B.Sc
Bluetooth Excha
OK
Cancel
Help
Read Only
Exclusive
List Files of Type:
Access Databases (*.m
Drives:
c:
Network...

Step 5:

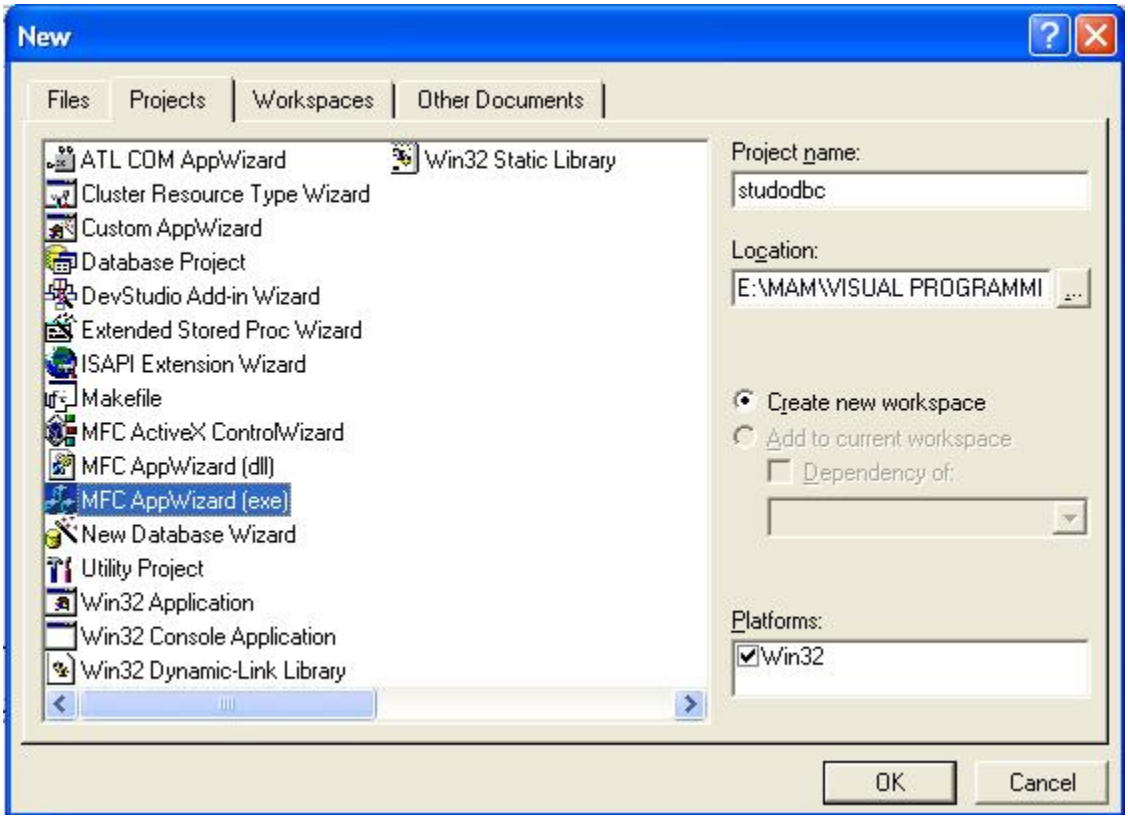

ODBC Microsoft Access Setup
Data Source Name:
student
Description:
Database
Database: C:\...\MY DOCUMENTS\student1.mdb
Select...
Create...
Repair...
Compact...
System Database
None
Database:
System Database...
OK
Cancel
Help
Advanced...
Options>>

Step 6:

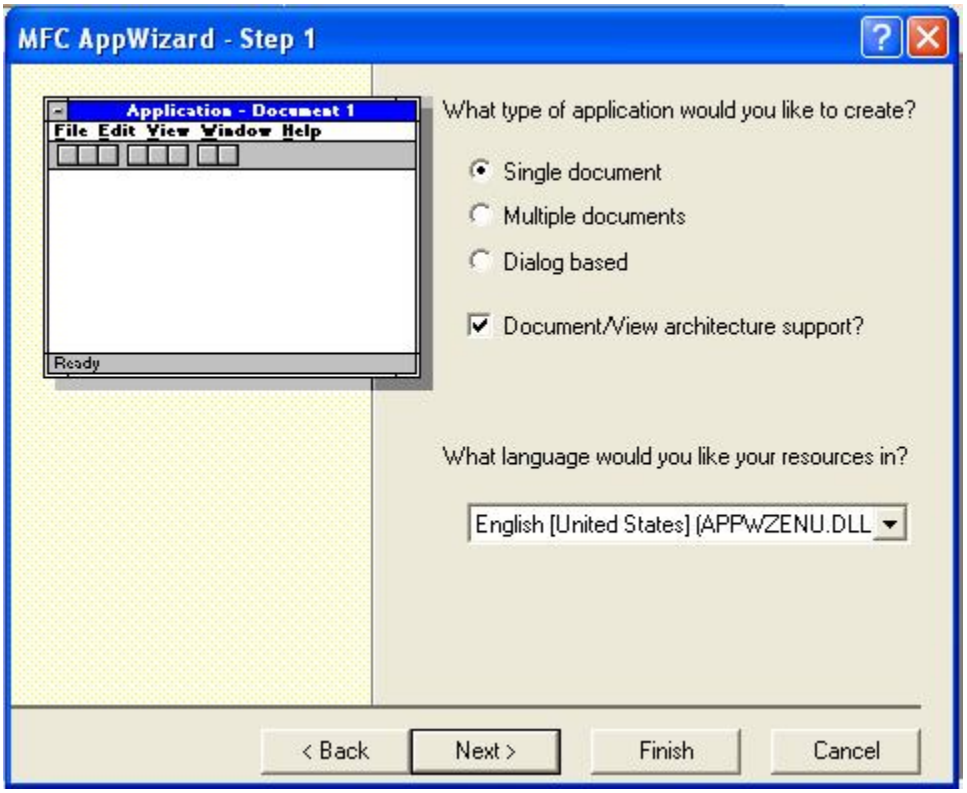


Application:

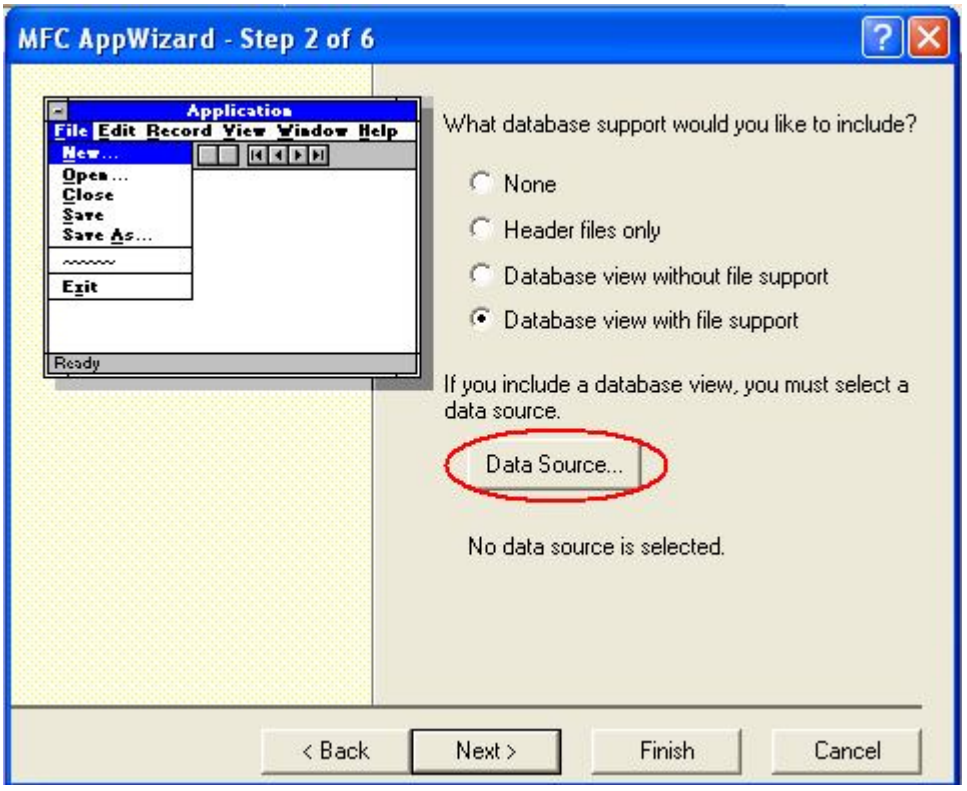
Step 1:


New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
studodbc
Location:
E:\MAM\VISUAL PROGRAMMI
Create new workspace
Add to current workspace
Dependency of:
Platforms:
Win32
OK
Cancel

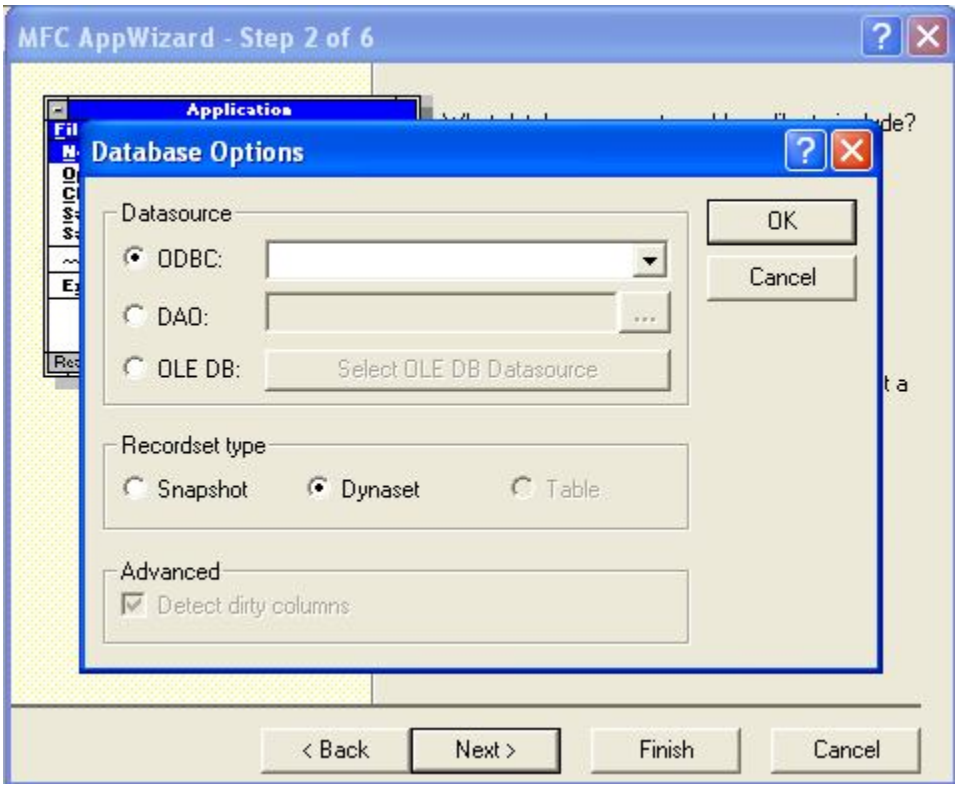
Step 2:


MFC AppWizard - Step 1
Application - Document 1
File Edit View Window Help
Ready
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

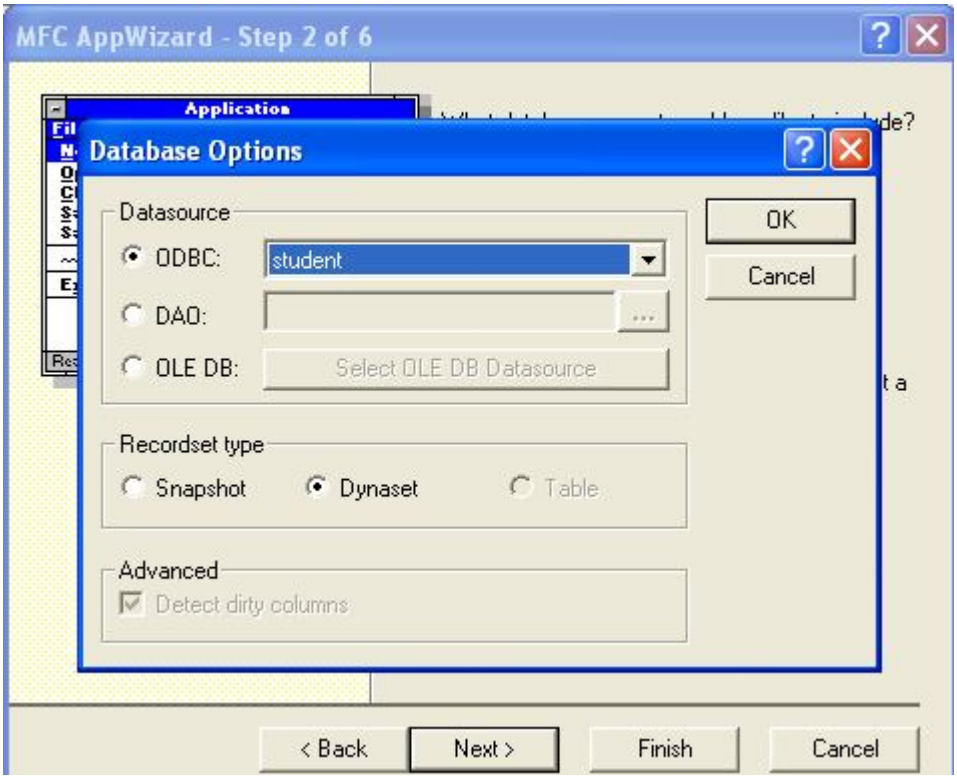
Step 3:


MFC AppWizard - Step 2 of 6
Application
File Edit Record View Window Help
New...
Open...
Close
Save
Save As...
Exit
Ready
What database support would you like to include?
None
Header files only
Database view without file support
Database view with file support
If you include a database view, you must select a data source.
Data Source...
No data source is selected.
< Back
Next >
Finish
Cancel

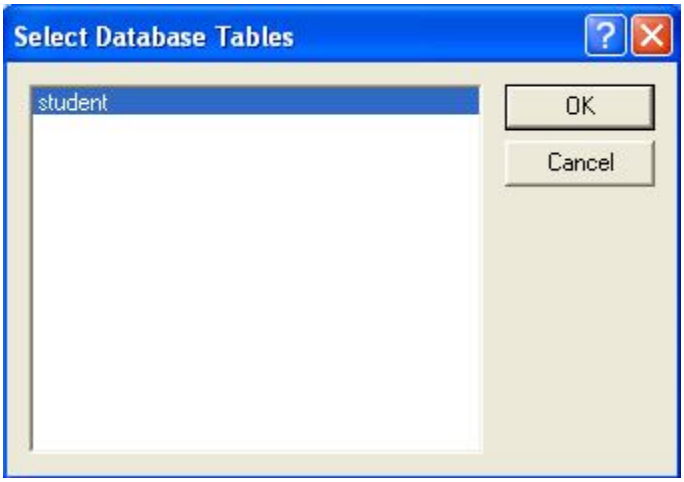
Step 4:


MFC AppWizard - Step 2 of 6
Application
Database Options
Datasource
ODBC:
DAO:
OLE DB:
Select OLE DB Datasource
OK
Cancel
Recordset type
Snapshot
Dynaset
Table
Advanced
Detect dirty columns
< Back
Next >
Finish
Cancel

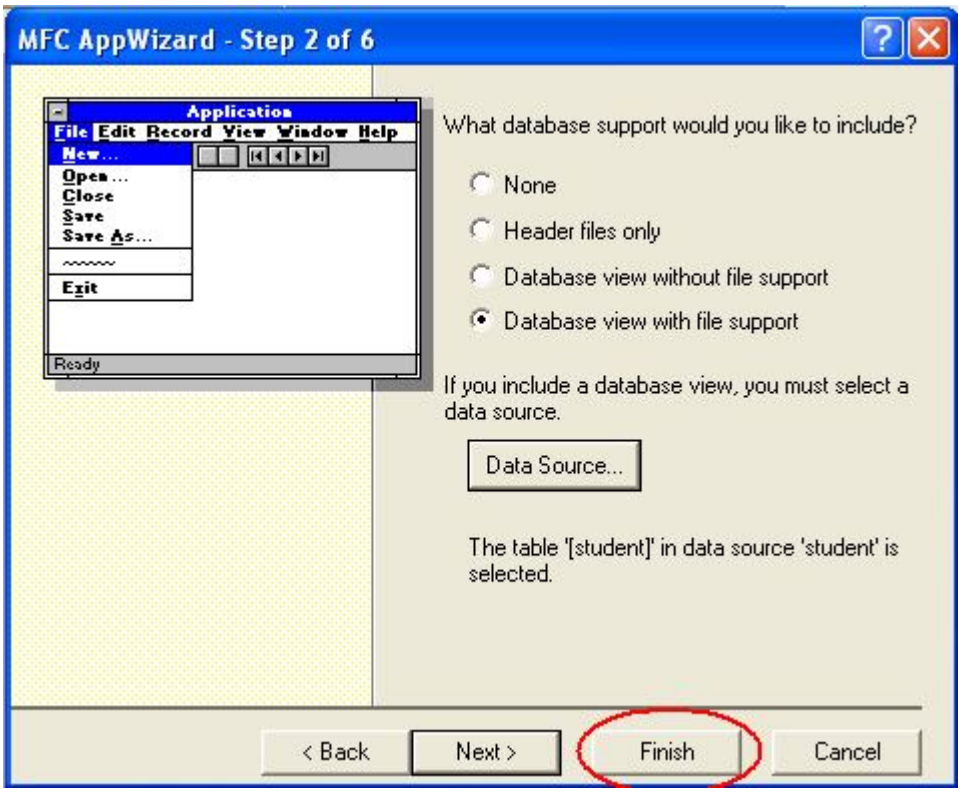
Step 5:


MFC AppWizard - Step 2 of 6
Application
Database Options
Datasource
ODBC:
student
DAO:
OLE DB:
Select OLE DB Datasource
OK
Cancel
Recordset type
Snapshot
Dynaset
Table
Advanced
Detect dirty columns
< Back
Next >
Finish
Cancel

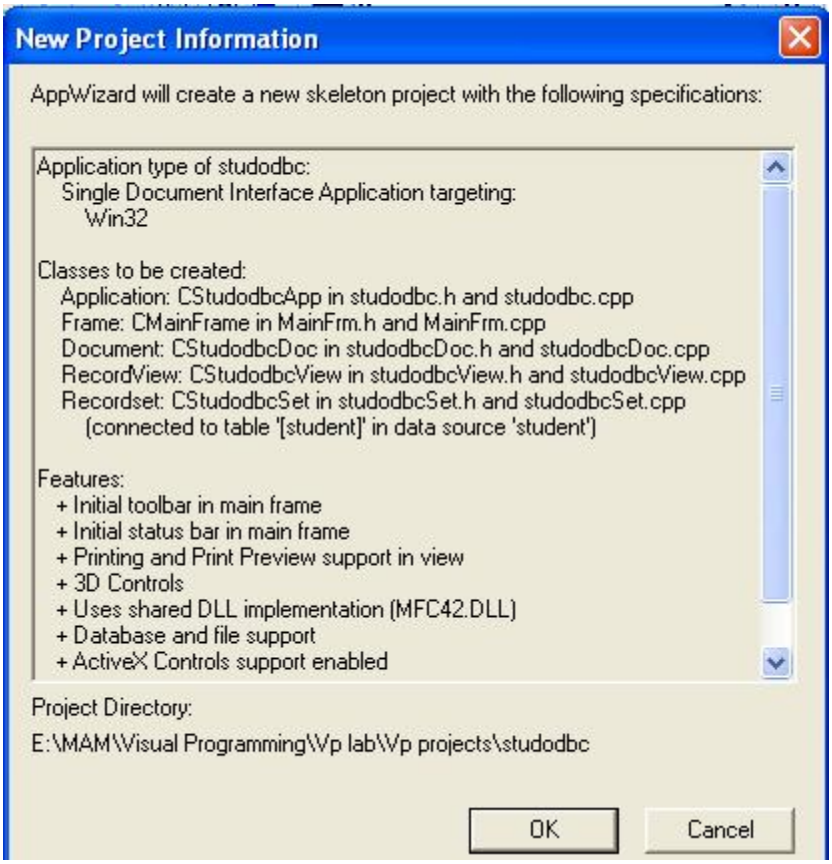
Step 6:


Select Database Tables
student
OK
Cancel

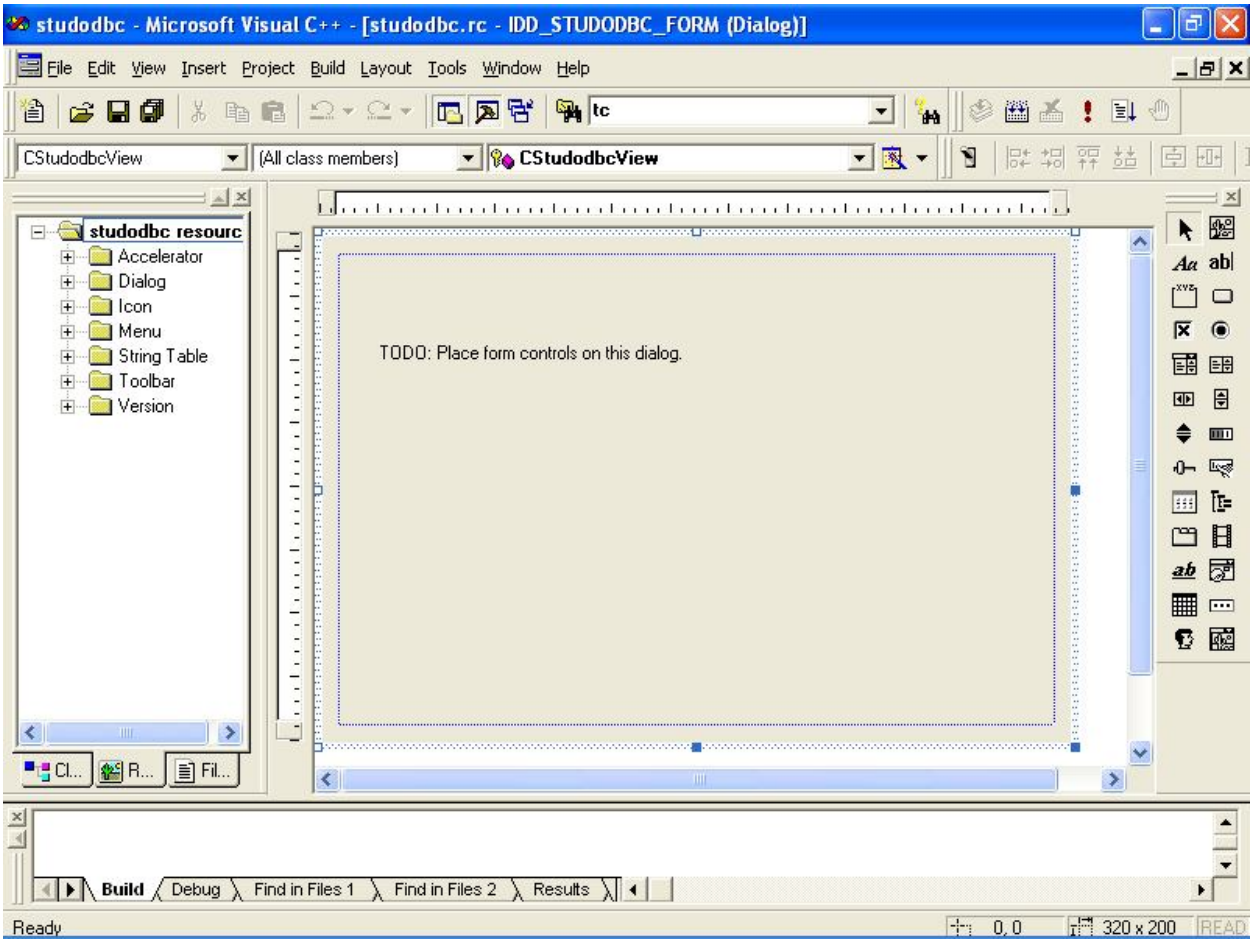
Step 7:


MFC AppWizard - Step 2 of 6
Application
File Edit Record View Window Help
New...
Open...
Close
Save
Save As...
Exit
Ready
What database support would you like to include?
None
Header files only
Database view without file support
Database view with file support
If you include a database view, you must select a data source.
Data Source...
The table '[student]' in data source 'student' is selected.
< Back
Next >
Finish
Cancel

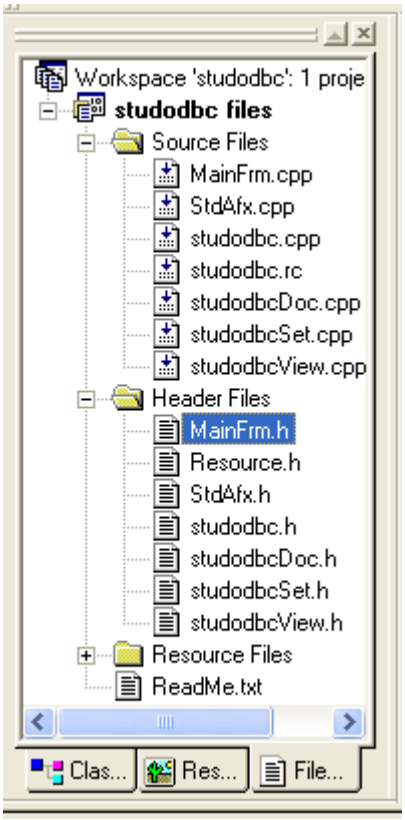
Step 8:


New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of studodbc:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CStudodbcApp in studodbc.h and studodbc.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CStudodbcDoc in studodbcDoc.h and studodbcDoc.cpp
RecordView: CStudodbcView in studodbcView.h and studodbcView.cpp
Recordset: CStudodbcSet in studodbcSet.h and studodbcSet.cpp
(connected to table '[student]' in data source 'student')
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ Database and file support
+ ActiveX Controls support enabled
Project Directory:
E:\MAM\Visual Programming\Vp lab\Vp projects\studodbc
OK
Cancel

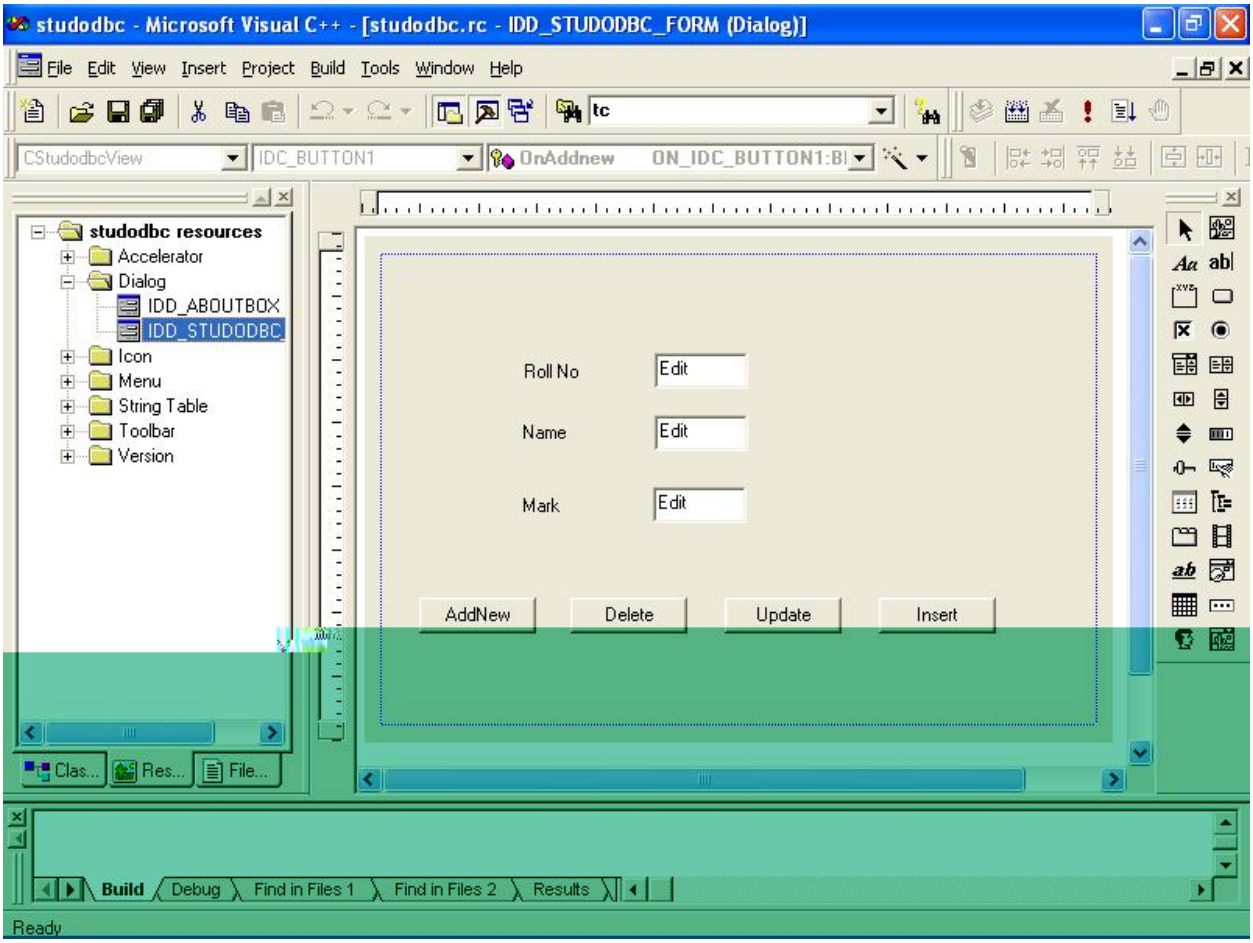
Step 9:


studodbc - Microsoft Visual C++ - [studodbc.rc - IDD_STUDODBC_FORM (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
tc
CStudodbcView
(All class members)
CStudodbcView
studodbc resourc
Accelerator
Dialog
Icon
Menu
String Table
Toolbar
Version
TODO: Place form controls on this dialog.
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
0, 0
320 x 200

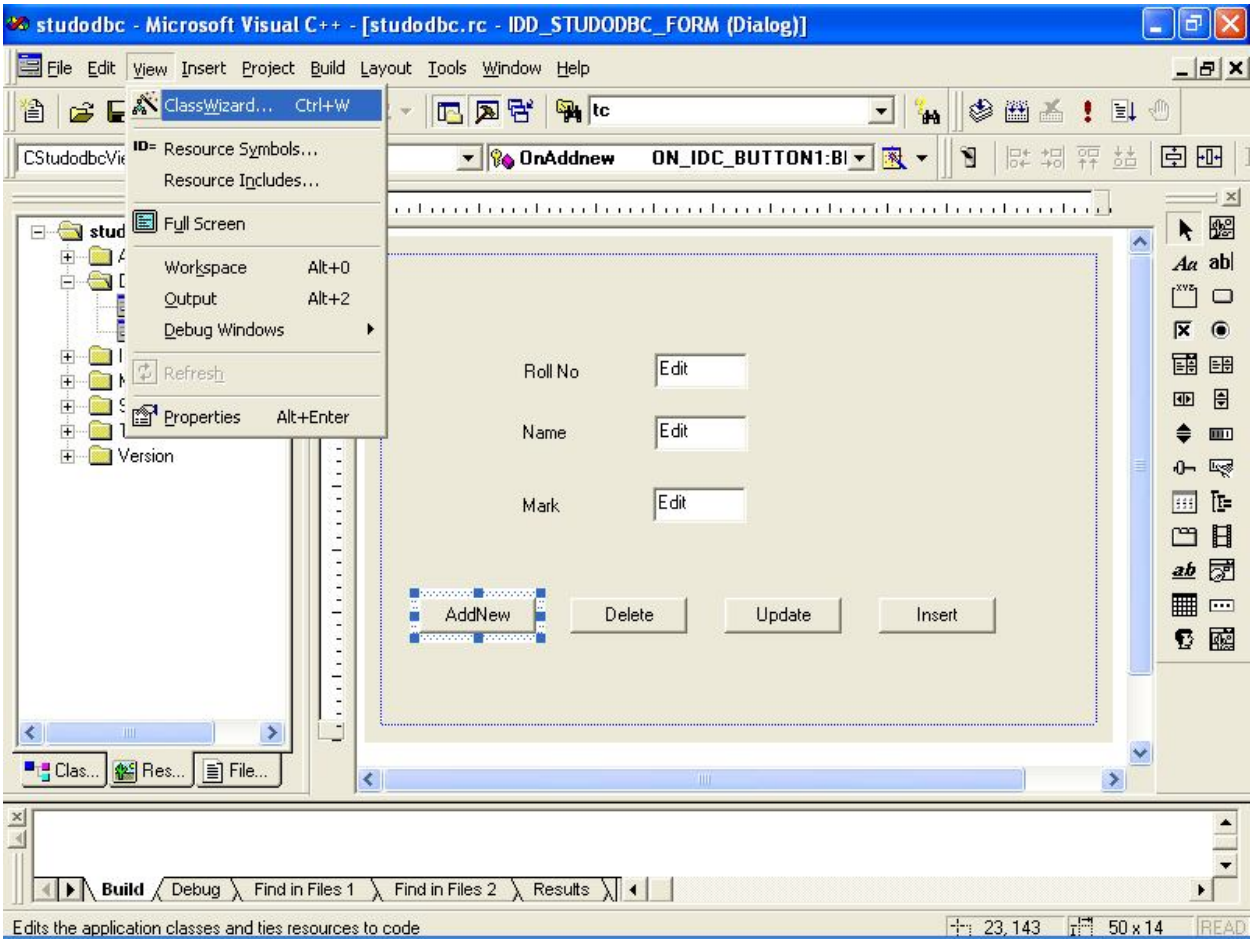
Step 10:


Workspace 'studodbc': 1 proje
studodbc files
Source Files
MainFrm.cpp
StdAfx.cpp
studodbc.cpp
studodbc.rc
studodbcDoc.cpp
studodbcSet.cpp
studodbcView.cpp
Header Files
MainFrm.h
Resource.h
StdAfx.h
studodbc.h
studodbcDoc.h
studodbcSet.h
studodbcView.h
Resource Files
ReadMe.txt
Clas...
Res...
File...

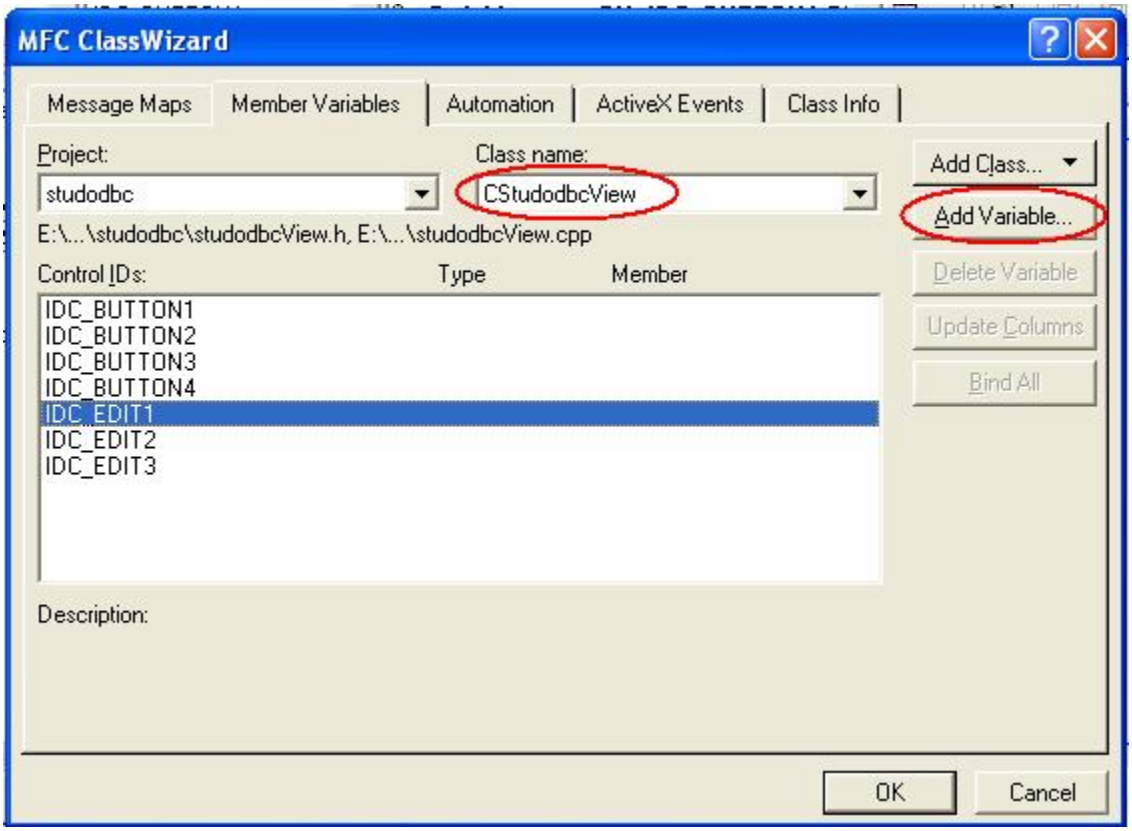
Step 11:


studodbc - Microsoft Visual C++ - [studodbc.rc - IDD_STUDODBC_FORM (Dialog)]
File Edit View Insert Project Build Tools Window Help
CStudodbcView
IDC_BUTTON1
OnAddnew
ON_IDC_BUTTON1:BI
studodbc resources
Accelerator
Dialog
IDD_ABOUTBOX
IDD_STUDODBC
Icon
Menu
String Table
Toolbar
Version
Clas...
Res...
File...
Roll No
Edit
Name
Edit
Mark
Edit
AddNew
Delete
Update
Insert
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready

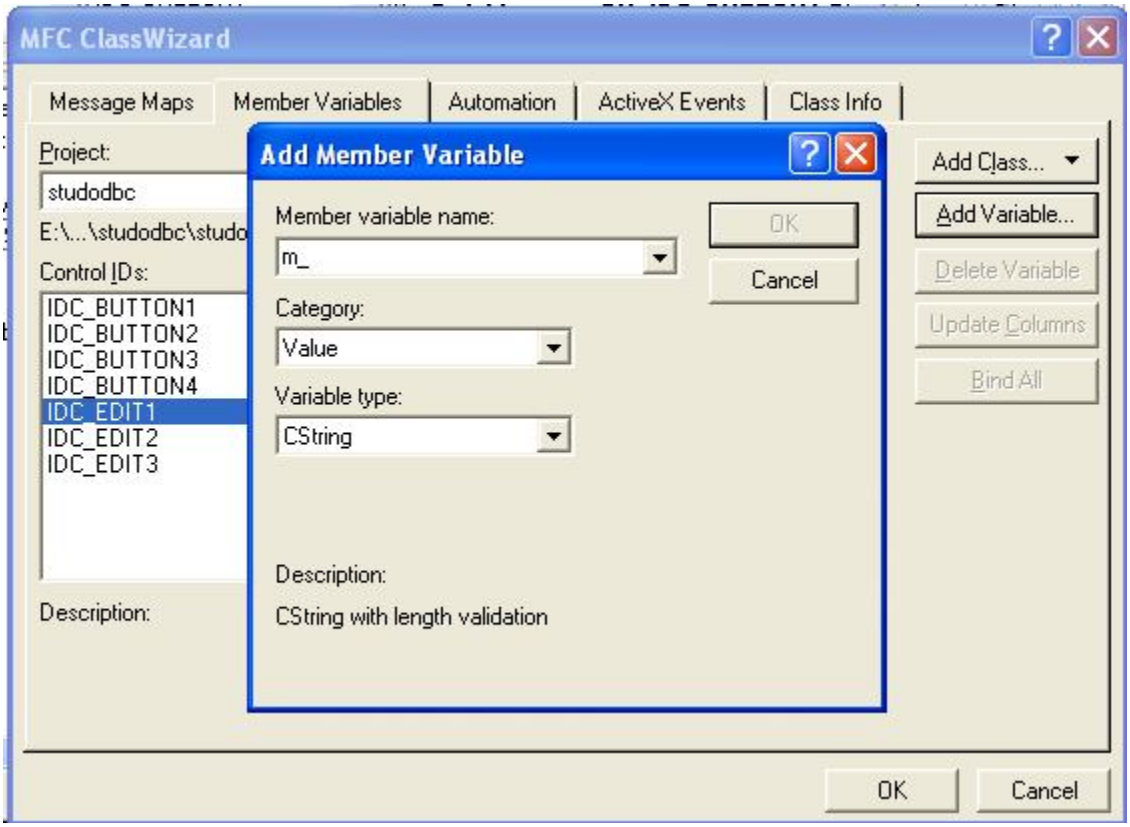
Step 12:


studodbc - Microsoft Visual C++ - [studodbc.rc - IDD_STUDODBC_FORM (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
ClassWizard... Ctrl+W
Resource Symbols...
Resource Includes...
Full Screen
Workspace Alt+0
Output Alt+2
Debug Windows
Refresh
Properties Alt+Enter
OnAddnew
ON_IDC_BUTTON1:BI
Version
Roll No
Name
Mark
Edit
Edit
Edit
AddNew
Delete
Update
Insert
Clas...
Res...
File...
Build
Debug
Find in Files 1
Find in Files 2
Results
Edits the application classes and ties resources to code
23, 143
50 x 14
READ

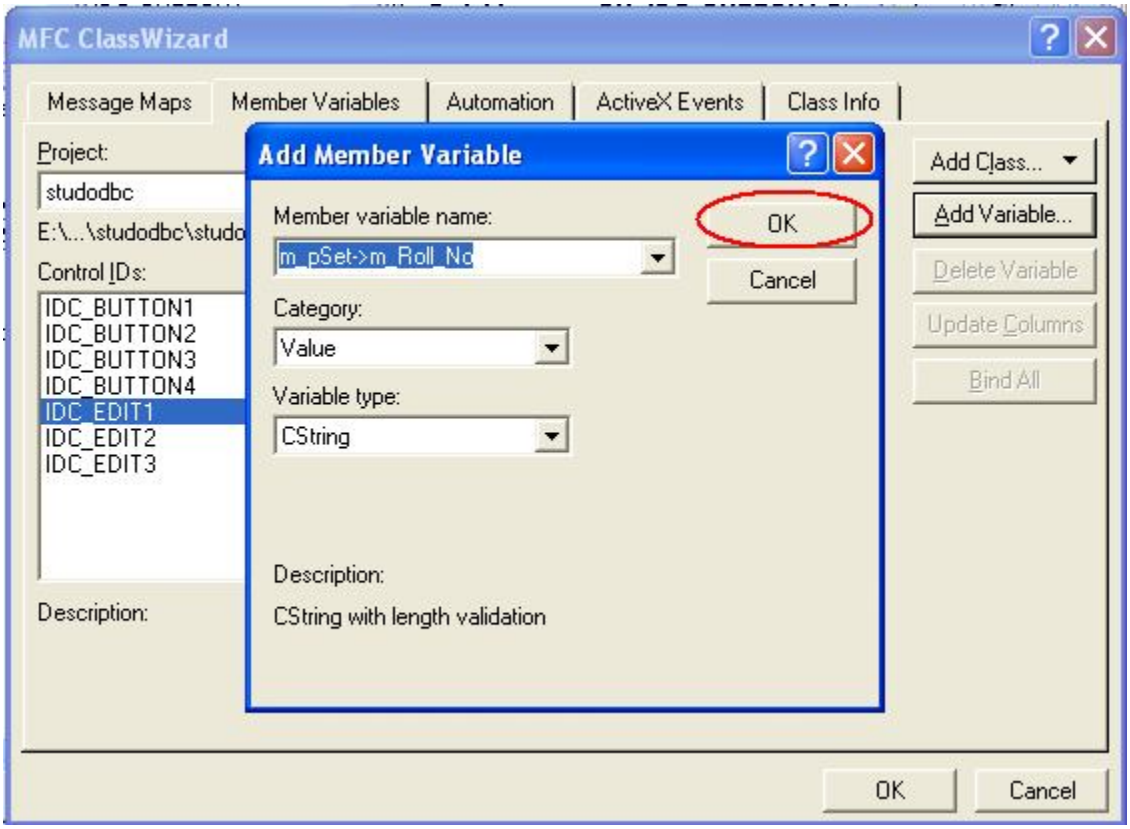
Step 13:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
studodbc
Class name:
CStudodbcView
Add Class...
Add Variable...
E:\...\studodbc\studodbcView.h, E:\...\studodbcView.cpp
Control IDs:
Type
Member
IDC_BUTTON1
IDC_BUTTON2
IDC_BUTTON3
IDC_BUTTON4
IDC_EDIT1
IDC_EDIT2
IDC_EDIT3
Delete Variable
Update Columns
Bind All
Description:
OK
Cancel

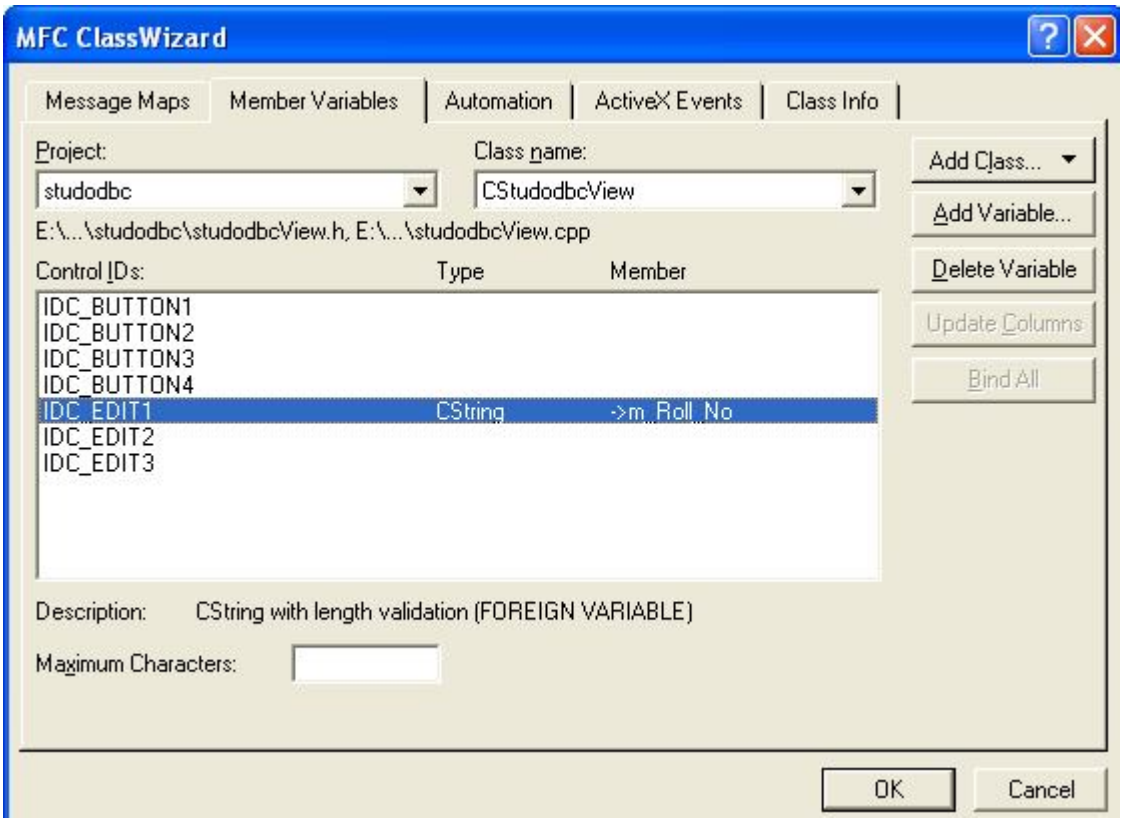
Step 14:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
studodbc
E:\...\studodbc\studo
Control IDs:
IDC_BUTTON1
IDC_BUTTON2
IDC_BUTTON3
IDC_BUTTON4
IDC_EDIT1
IDC_EDIT2
IDC_EDIT3
Description:
Add Member Variable
Member variable name:
m_
OK
Cancel
Category:
Value
Variable type:
CString
Description:
CString with length validation
Add Class...
Add Variable...
Delete Variable
Update Columns
Bind All
OK
Cancel

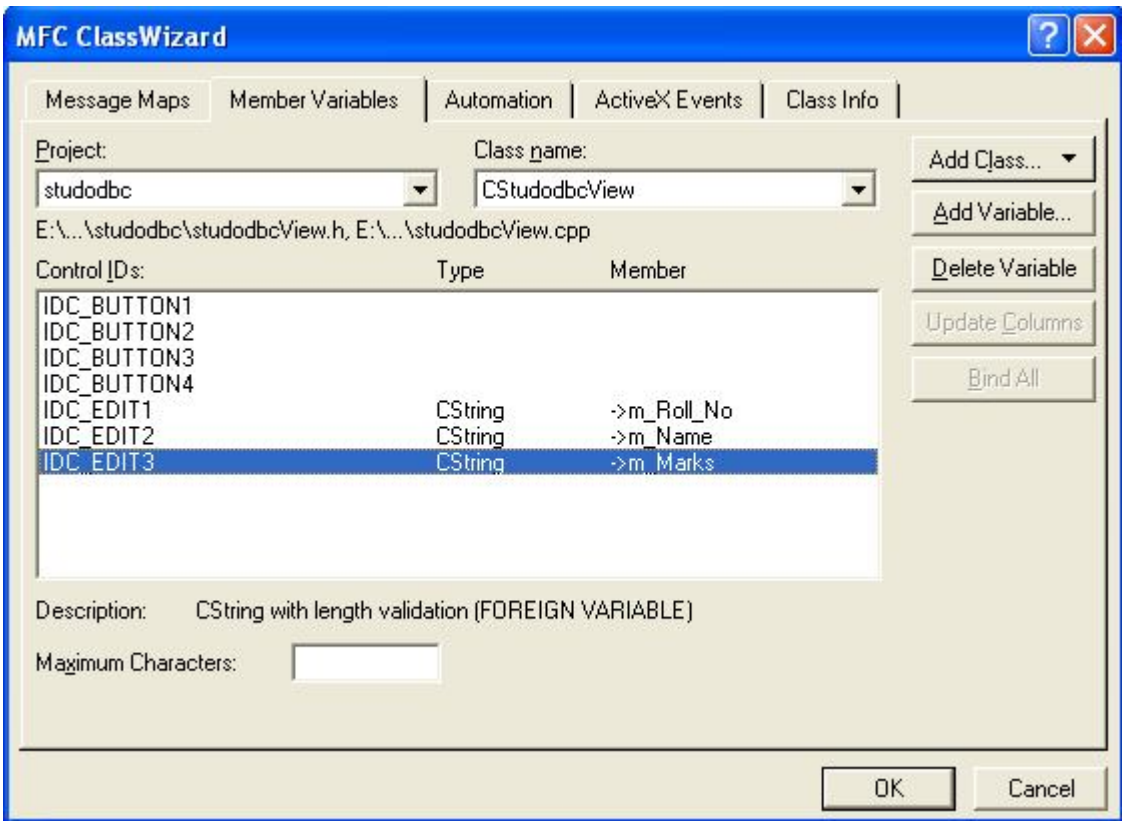
Step 15:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
studodbc
E:\...\studodbc\studo
Control IDs:
IDC_BUTTON1
IDC_BUTTON2
IDC_BUTTON3
IDC_BUTTON4
IDC_EDIT1
IDC_EDIT2
IDC_EDIT3
Description:
Add Member Variable
Member variable name:
m_pSet->m_Roll_No
Category:
Value
Variable type:
CString
Description:
CString with length validation
OK
Cancel
Add Class...
Add Variable...
Delete Variable
Update Columns
Bind All
OK
Cancel

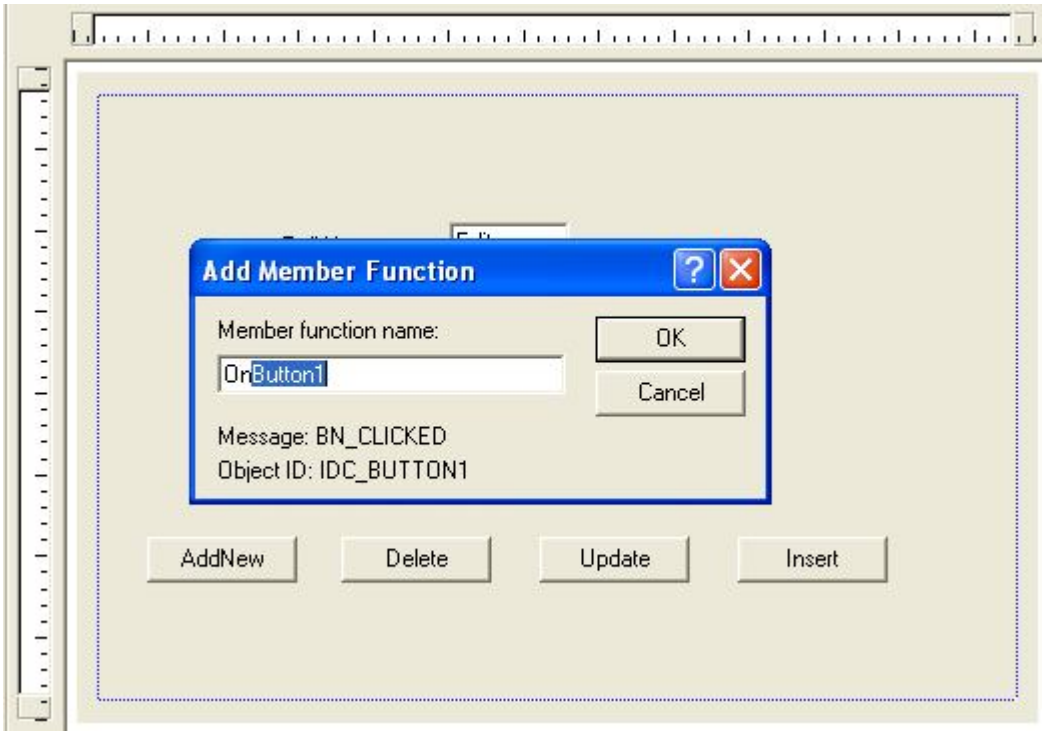
Step 16:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
studodbc
Class name:
CStudodbcView
E:\...\studodbc\studodbcView.h, E:\...\studodbcView.cpp
Control IDs:
Type
Member
IDC_BUTTON1
IDC_BUTTON2
IDC_BUTTON3
IDC_BUTTON4
IDC_EDIT1
CString
->m_Roll_No
IDC_EDIT2
IDC_EDIT3
Description:
CString with length validation (FOREIGN VARIABLE)
Maximum Characters:
Add Class...
Add Variable...
Delete Variable
Update Columns
Bind All
OK
Cancel

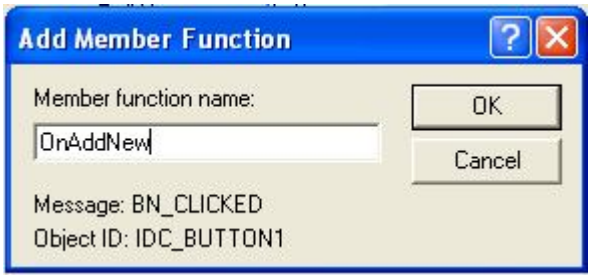
Step 17:


MFC ClassWizard
Message Maps
Member Variables
Automation
ActiveX Events
Class Info
Project:
studodbc
Class name:
CStudodbcView
E:\...\studodbc\studodbcView.h, E:\...\studodbcView.cpp
Control IDs:
Type
Member
IDC_BUTTON1
IDC_BUTTON2
IDC_BUTTON3
IDC_BUTTON4
IDC_EDIT1 CString ->m_Roll_No
IDC_EDIT2 CString ->m_Name
IDC_EDIT3 CString ->m_Marks
Add Class...
Add Variable...
Delete Variable
Update Columns
Bind All
Description: CString with length validation (FOREIGN VARIABLE)
Maximum Characters:
OK
Cancel

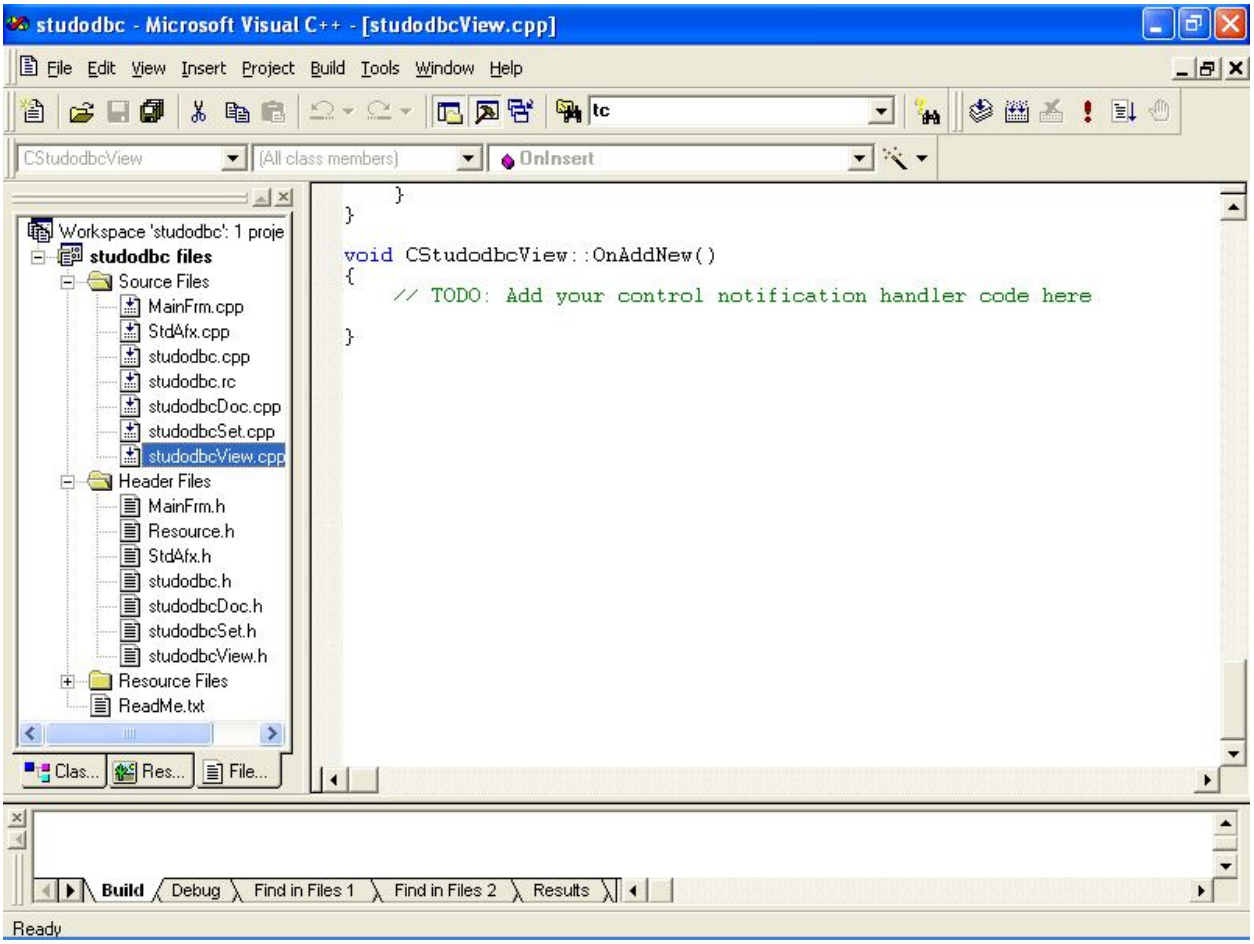
Step 18:


Add Member Function
Member function name:
OnButton1
OK
Cancel
Message: BN_CLICKED
Object ID: IDC_BUTTON1
AddNew
Delete
Update
Insert

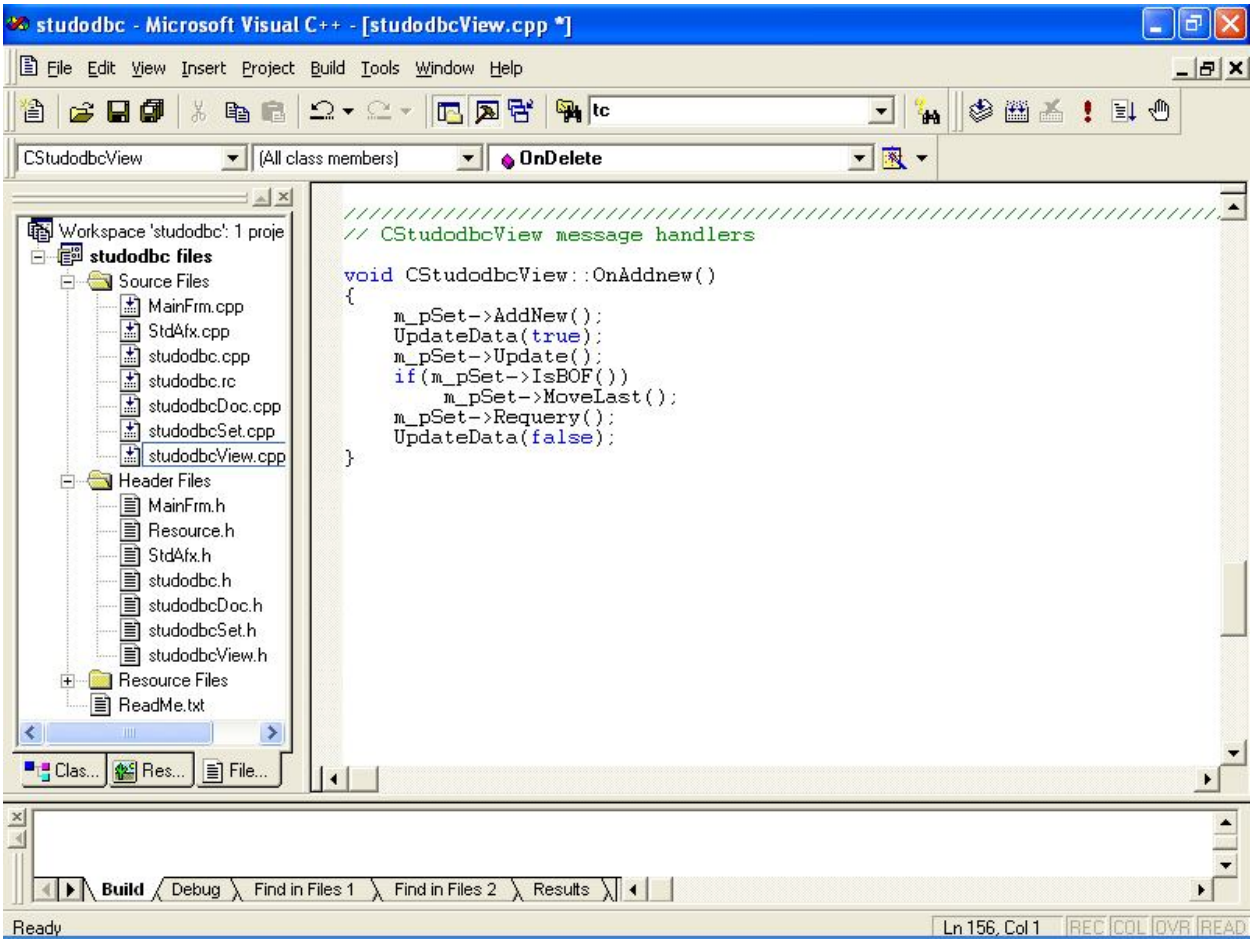
Step 19:



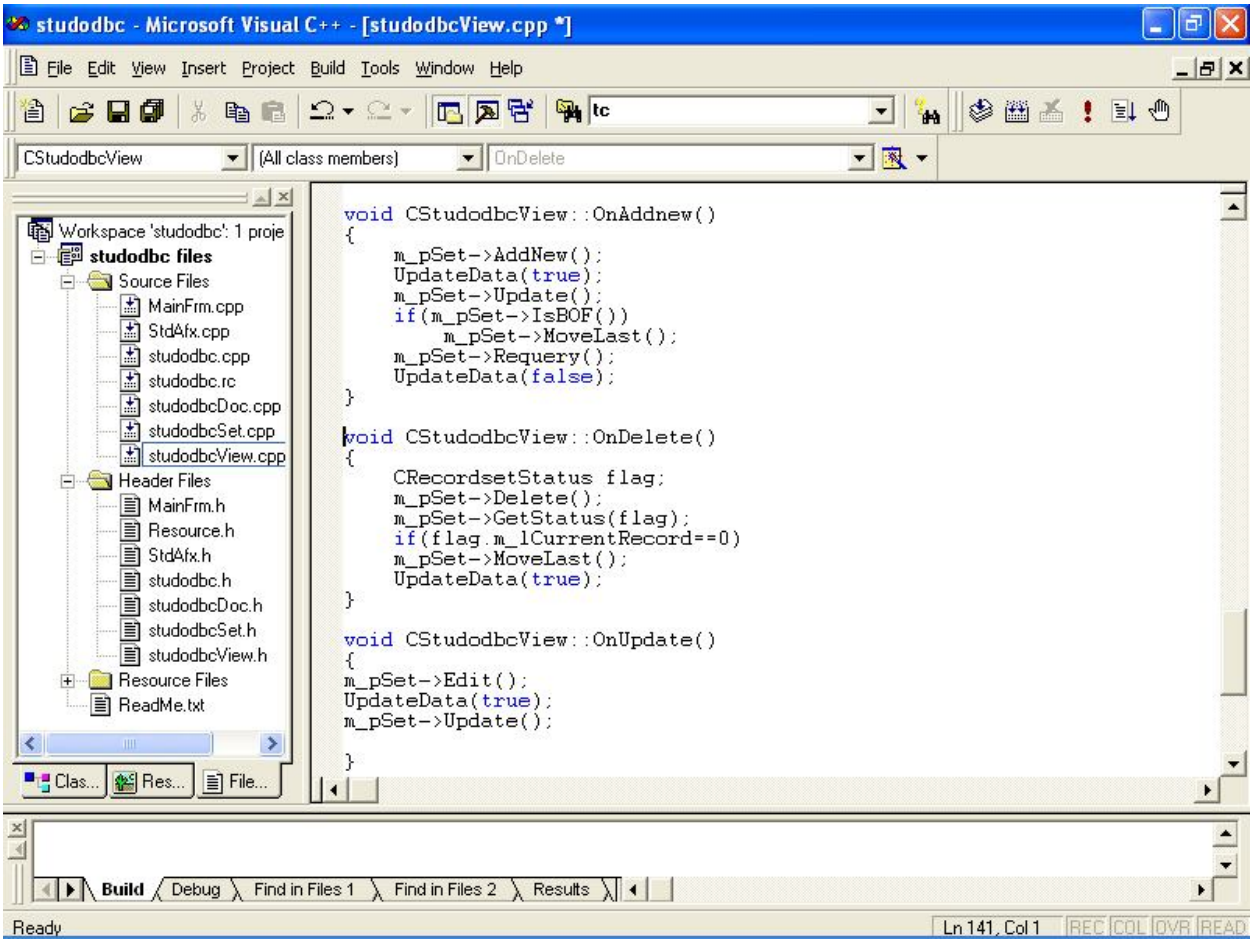
Step 20:



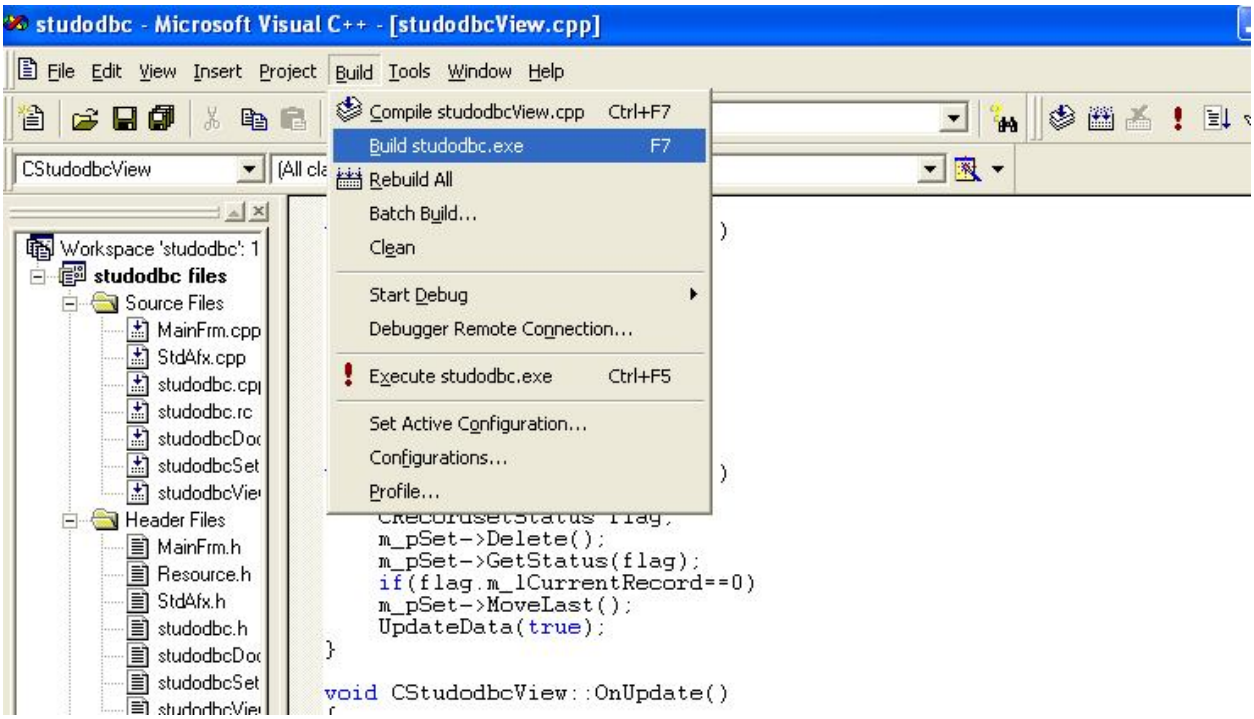
Step 21:


studodbc - Microsoft Visual C++ - [studodbcView.cpp *]
File Edit View Insert Project Build Tools Window Help
tc
CStudodbcView
(All class members)
OnDelete
Workspace 'studodbc': 1 proje
studodbc files
Source Files
MainFrm.cpp
StdAfx.cpp
studodbc.cpp
studodbc.rc
studodbcDoc.cpp
studodbcSet.cpp
studodbcView.cpp
Header Files
MainFrm.h
Resource.h
StdAfx.h
studodbc.h
studodbcDoc.h
studodbcSet.h
studodbcView.h
Resource Files
ReadMe.txt
Clas...
Res...
File...
//
// CStudodbcView message handlers
void CStudodbcView::OnAddnew()
{
m_pSet->AddNew();
UpdateData(true);
m_pSet->Update();
if(m_pSet->IsBOF())
m_pSet->MoveLast();
m_pSet->Requery();
UpdateData(false);
}
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
Ln 156, Col 1
REC COL OVR READ

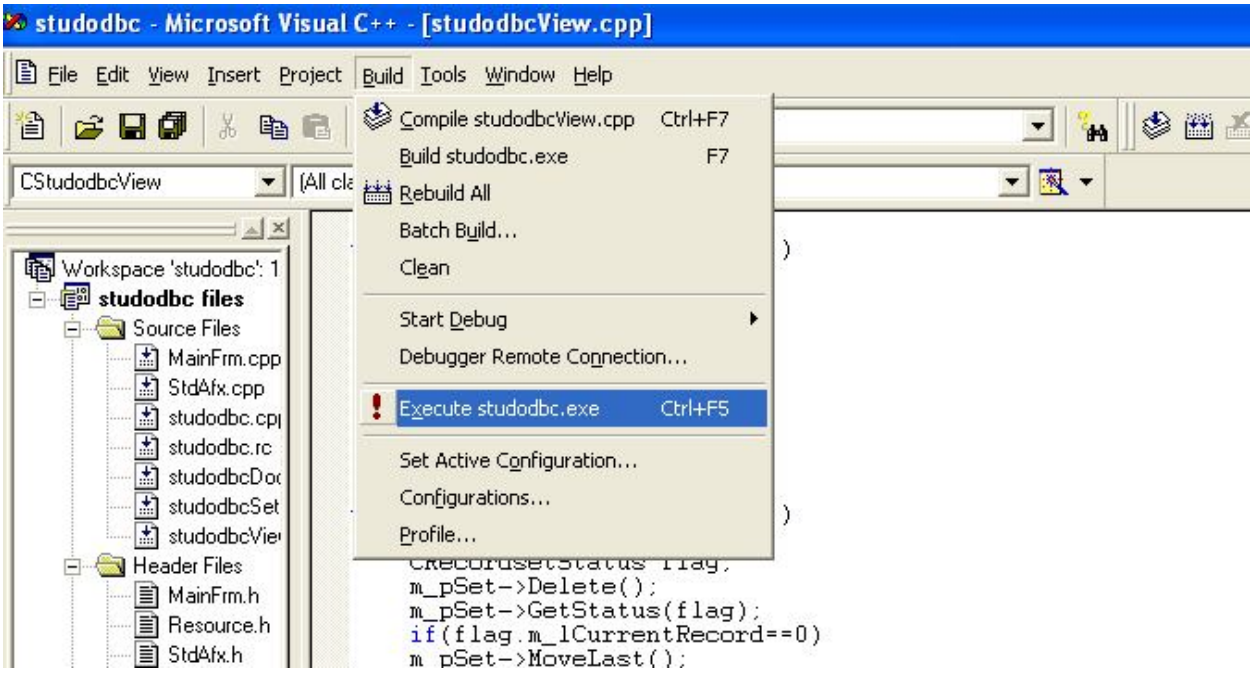
Step 22:


studodbc - Microsoft Visual C++ - [studodbcView.cpp *]
File Edit View Insert Project Build Tools Window Help
CStudodbcView
(All class members)
OnDelete
Workspace 'studodbc': 1 proje
studodbc files
Source Files
MainFrm.cpp
StdAfx.cpp
studodbc.cpp
studodbc.rc
studodbcDoc.cpp
studodbcSet.cpp
studodbcView.cpp
Header Files
MainFrm.h
Resource.h
StdAfx.h
studodbc.h
studodbcDoc.h
studodbcSet.h
studodbcView.h
Resource Files
ReadMe.txt
Clas...
Res...
File...
void CStudodbcView::OnAddnew()
{
m_pSet->AddNew();
UpdateData(true);
m_pSet->Update();
if(m_pSet->IsBOF())
m_pSet->MoveLast();
m_pSet->Requery();
UpdateData(false);
}
void CStudodbcView::OnDelete()
{
CRecordsetStatus flag;
m_pSet->Delete();
m_pSet->GetStatus(flag);
if(flag.m_lCurrentRecord==0)
m_pSet->MoveLast();
UpdateData(true);
}
void CStudodbcView::OnUpdate()
{
m_pSet->Edit();
UpdateData(true);
m_pSet->Update();
}
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready
Ln 141, Col 1
REC COL OVR READ

Step 23:



Step 24:



Program:

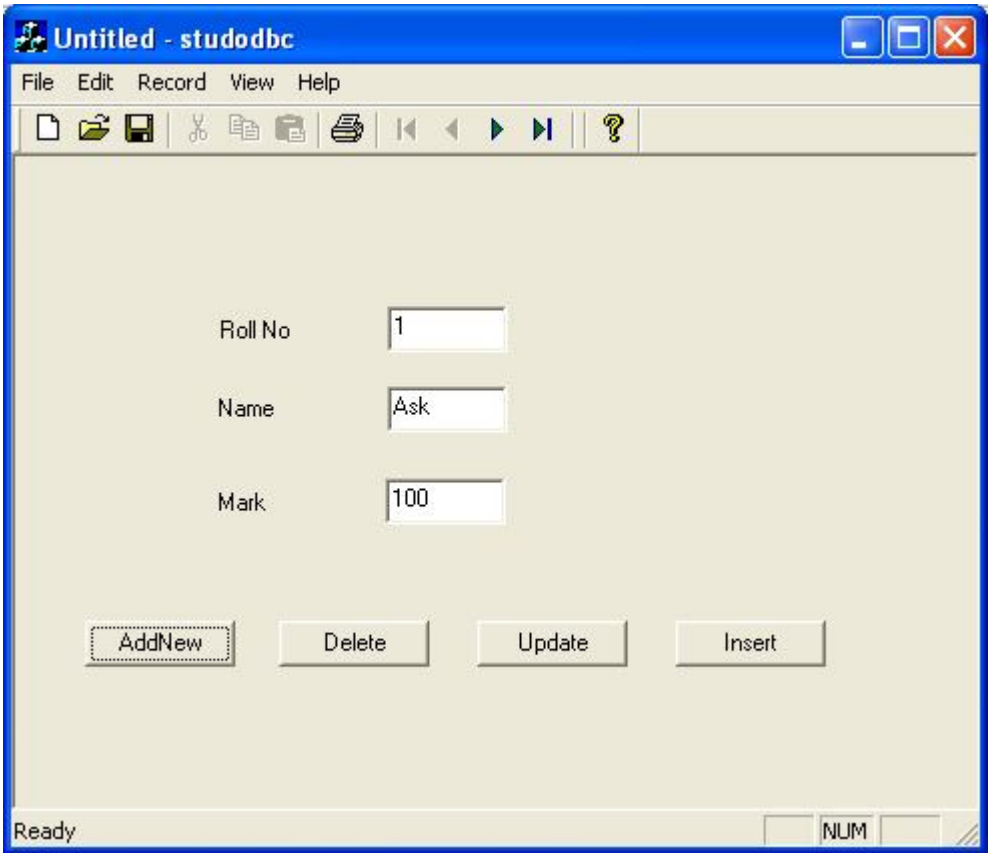
// studodbcView.cpp

void CStudodbcView::OnAddnew()

```
{
        m_pSet->AddNew();
        UpdateData(true);
        m_pSet->Update();
        if(m_pSet->IsBOF())          // detects empty recordset
                m_pSet->MoveLast();
        m_pSet->Requery();
        UpdateData(false);
}

void CStudodbcView::OnDelete()
{
        CRecordsetStatus flag;
        m_pSet->Delete();
        m_pSet->GetStatus(flag);
        if(flag.m_lCurrentRecord==0)
        m_pSet->MoveLast();
        UpdateData(true);
}

void CStudodbcView::OnUpdate()
{
m_pSet->Edit();
UpdateData(true);
m_pSet->Update();
```

```
}

void CStudodbcView::OnInsert()
{
	try{
		m_pSet->m_pDatabase->BeginTrans();
		m_pSet->m_pDatabase->ExecuteSQL("Insert into student values
(&m_text2,&m_text1,&m_text3)");
		if(m_pSet->m_pDatabase->CommitTrans())
			TRACE ("Transaction done");
		else
			TRACE("Error");
	}
	catch(CDBException *pEx)
	{
		pEx->ReportError();
		m_pSet->m_pDatabase->Rollback();
	}
}
```

Output:



Result:

Thus the VC++ program to access the Student database through MFC ODBC has been developed, built and verified.