



MFC

**tutorialspoint**  
SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

The Microsoft Foundation Class (MFC) library provides a set of functions, constants, data types, and classes to simplify creating applications for the Microsoft Windows operating systems. In this tutorial, you will learn all about how to start and create Windows-based applications using MFC.

## Audience

---

This tutorial is designed for all those developers who are keen on developing best-in-class applications using MFC. The tutorial provides a hands-on approach with step-by-step program examples, source codes, and illustrations that will assist the developers to learn and put the acquired knowledge into practice.

## Prerequisites

---

To gain advantage of this tutorial you need to be familiar with programming for Windows. You also need to know the basics of programming in C++ and understand the fundamentals of object-oriented programming.

## Disclaimer & Copyright

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com).

## Table of Contents

---

About the Tutorial .....	i
Audience.....	i
Prerequisites.....	i
Disclaimer & Copyright .....	i
<b>Table of Contents.....</b>	<b>ii</b>
<b>1. MFC - OVERVIEW .....</b>	<b>1</b>
Prerequisites.....	1
What is MFC?.....	1
MFC Framework .....	1
Why MFC? .....	2
<b>2. MFC - ENVIRONMENT SETUP .....</b>	<b>3</b>
<b>3. MFC - VC++ PROJECTS .....</b>	<b>8</b>
<b>4. MFC - GETTING STARTED .....</b>	<b>9</b>
Create Project Using Project Templates .....	9
Create Project from Scratch .....	14
<b>5. MFC - WINDOWS FUNDAMENTALS.....</b>	<b>19</b>
Window Creation.....	24
Main Window .....	25
Windows Styles .....	27
Windows Location .....	29
Windows Size .....	30
Windows Dimensions .....	31
Windows Parents.....	32

6. MFC - DIALOG BOXES .....	34
Dialog Box Creation .....	40
Dialog Location .....	43
Dialog Box Dimensions .....	45
Dialog Box Methods.....	45
Modal Dialog Boxes .....	46
Dialog-Based Applications .....	49
7. MFC - WINDOWS RESOURCES.....	58
Identifiers .....	59
Icons .....	60
Menus .....	63
Toolbars.....	68
Accelerators.....	75
8. MFC - PROPERTY SHEETS.....	86
9. MFC - WINDOWS LAYOUT .....	97
Adding controls .....	97
Control Grid .....	99
Controls Resizing .....	100
Controls Positions .....	101
Tab Ordering.....	107
10. MFC - CONTROLS MANAGEMENT .....	111
Control Variable/Instance.....	111
Control Value Variable.....	116
Controls Event Handlers .....	118
Controls Management .....	121

11. MFC - WINDOWS CONTROLS .....	128
Static Control .....	128
Animation Control .....	128
Button .....	134
Bitmap Button .....	137
Command Button .....	139
Static Text.....	142
List Box .....	144
Combo Boxes.....	153
Radio Buttons .....	159
Checkboxes.....	164
Image Lists.....	178
Edit Box .....	182
Rich Edit.....	187
Group Box.....	198
Spin Button.....	199
Managing the Updown Control.....	201
Progress Control .....	206
Progress Bars .....	210
Timer .....	213
Date & Time Picker .....	217
Picture .....	222
Image Editor .....	225
Slider Controls .....	228
Scrollbars.....	235
Tree Control.....	237
List Control .....	245

<b>12. MFC - MESSAGES AND EVENTS .....</b>	<b>254</b>
<b>Overview .....</b>	<b>254</b>
<b>Map of Messages.....</b>	<b>254</b>
<b>Windows Messages .....</b>	<b>258</b>
<b>Command Messages.....</b>	<b>264</b>
<b>Keyboard Messages .....</b>	<b>264</b>
<b>Mouse Messages .....</b>	<b>267</b>
<b>13. MFC - ACTIVEX CONTROL.....</b>	<b>270</b>
<b>14. MFC - FILE SYSTEM.....</b>	<b>276</b>
<b>Drives .....</b>	<b>276</b>
<b>Directories .....</b>	<b>280</b>
<b>File Processing .....</b>	<b>284</b>
<b>15. MFC - STANDARD I/O.....</b>	<b>288</b>
<b>16. MFC - DOCUMENT VIEW.....</b>	<b>294</b>
<b>View .....</b>	<b>294</b>
<b>Document.....</b>	<b>294</b>
<b>Frame .....</b>	<b>294</b>
<b>Single Document Interface (SDI) .....</b>	<b>294</b>
<b>Multiple Document Interface (MDI).....</b>	<b>296</b>
<b>17. MFC - STRINGS.....</b>	<b>300</b>
<b>Create String.....</b>	<b>302</b>
<b>Empty String .....</b>	<b>303</b>
<b>String Concatenation .....</b>	<b>305</b>
<b>String Length.....</b>	<b>306</b>
<b>String Comparison .....</b>	<b>308</b>

18.	MFC - CARRAY.....	310
	Create CArray Object .....	311
	Add items .....	311
	Retrieve Items .....	311
	Add Items in the Middle .....	313
	Update Item Value.....	315
	Copy Array.....	317
	Remove Items.....	319
19.	MFC - LINKED LISTS.....	322
	Singly Linked List.....	322
	Doubly Linked List.....	322
	CList Class .....	323
	Create CList Object .....	324
	Add items .....	324
	Retrieve Items .....	324
	Add Items in the Middle .....	326
	Update Item Value.....	328
	Remove Items.....	330
20.	MFC - DATABASE CLASSES.....	333
	CDatabase .....	333
	Insert Query.....	335
	Retrieve Record .....	337
	Update Record.....	342
	Delete Record .....	346
21.	MFC - SERIALIZATION.....	351

22. MFC - MULTITHREADING .....	358
23. MFC - INTERNET PROGRAMMING.....	367
24. MFC - GDI.....	381
Drawing .....	381
Lines .....	387
Polylines .....	390
Rectangles .....	391
Squares.....	393
Pies.....	394
Arcs.....	396
Chords .....	398
Colors .....	399
Fonts.....	401
Pens.....	403
Brushes.....	404
25. MFC - LIBRARIES.....	407
Static Library.....	407
Dynamic Library.....	421

# 1. MFC - Overview

The Microsoft Foundation Class (MFC) library provides a set of functions, constants, data types, and classes to simplify creating applications for the Microsoft Windows operating systems. In this tutorial, you will learn all about how to start and create windows based applications using MFC.

## Prerequisites

---

We have assumed that you know the following:

- A little about programming for Windows.
- The basics of programming in C++.
- Understand the fundamentals of object-oriented programming.

## What is MFC?

---

The Microsoft Foundation Class Library (MFC) is an "application framework" for programming in Microsoft Windows. MFC provides much of the code, which are required for the following:

- Managing Windows.
- Menus and dialog boxes.
- Performing basic input/output.
- Storing collections of data objects, etc.

You can easily extend or override the basic functionality the MFC framework in your C++ applications by adding your application-specific code into MFC framework.

## MFC Framework

---

- The MFC framework provides a set of reusable classes designed to simplify Windows programming.
- MFC provides classes for many basic objects, such as strings, files, and collections that are used in everyday programming.
- It also provides classes for common Windows APIs and data structures, such as windows, controls, and device contexts.
- The framework also provides a solid foundation for more advanced features, such as ActiveX and document view processing.
- In addition, MFC provides an application framework, including the classes that make up the application architecture hierarchy.

## Why MFC?

---

The MFC framework is a powerful approach that lets you build upon the work of expert programmers for Windows. MFC framework has the following advantages.

- It shortens development time.
- It makes code more portable.
- It also provides tremendous support without reducing programming freedom and flexibility.
- It gives easy access to "hard to program" user-interface elements and technologies.
- MFC simplifies database programming through Data Access Objects (DAO) and Open Database Connectivity (ODBC), and network programming through Windows Sockets.

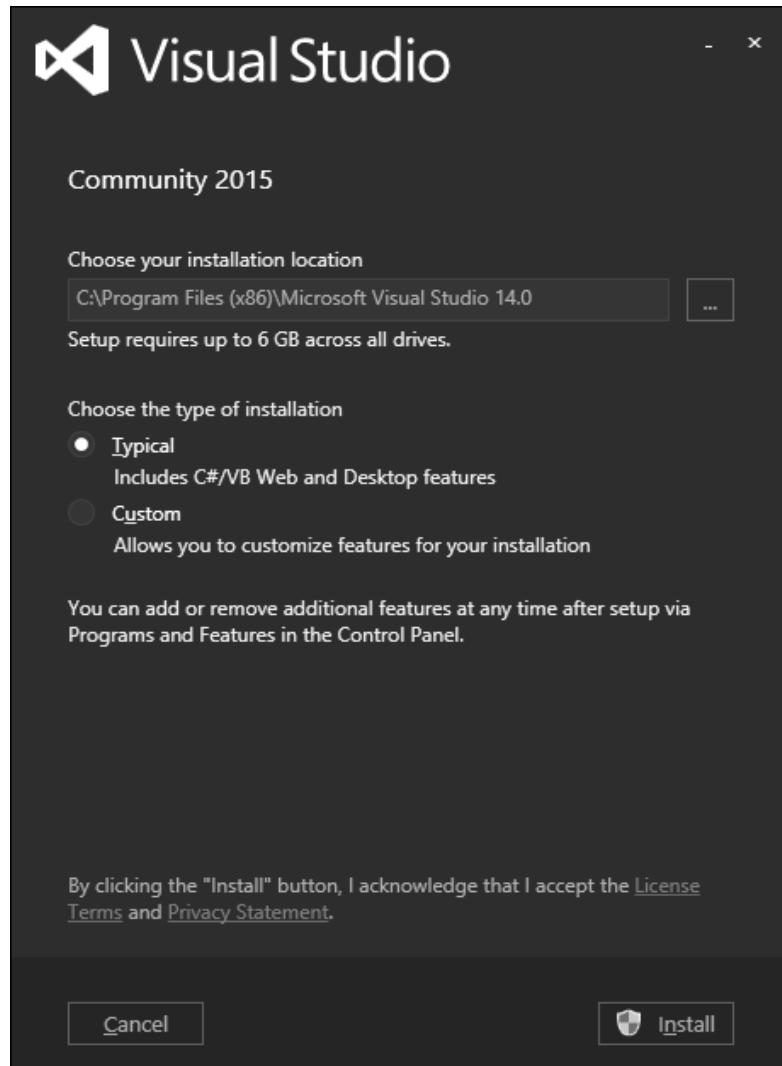
## 2. MFC - Environment Setup

Microsoft Visual C++ is a programming environment used to create applications for the Microsoft Windows operating systems. To use MFC framework in your C++ application, you must have installed either Microsoft Visual C++ or Microsoft Visual Studio. Microsoft Visual Studio also contains the Microsoft Visual C++ environment.

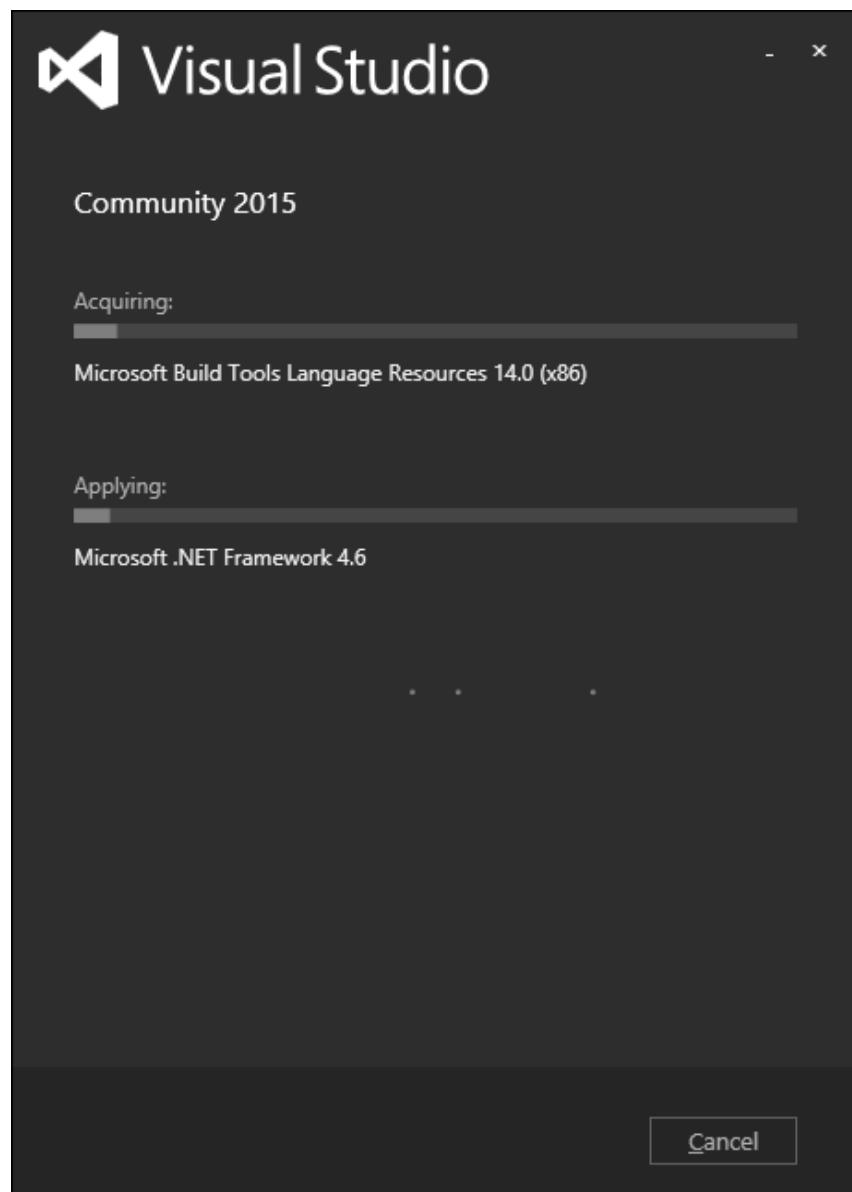
Microsoft provides a free version of visual studio which also contains SQL Server and it can be downloaded from <https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx>.

Following are the installation steps.

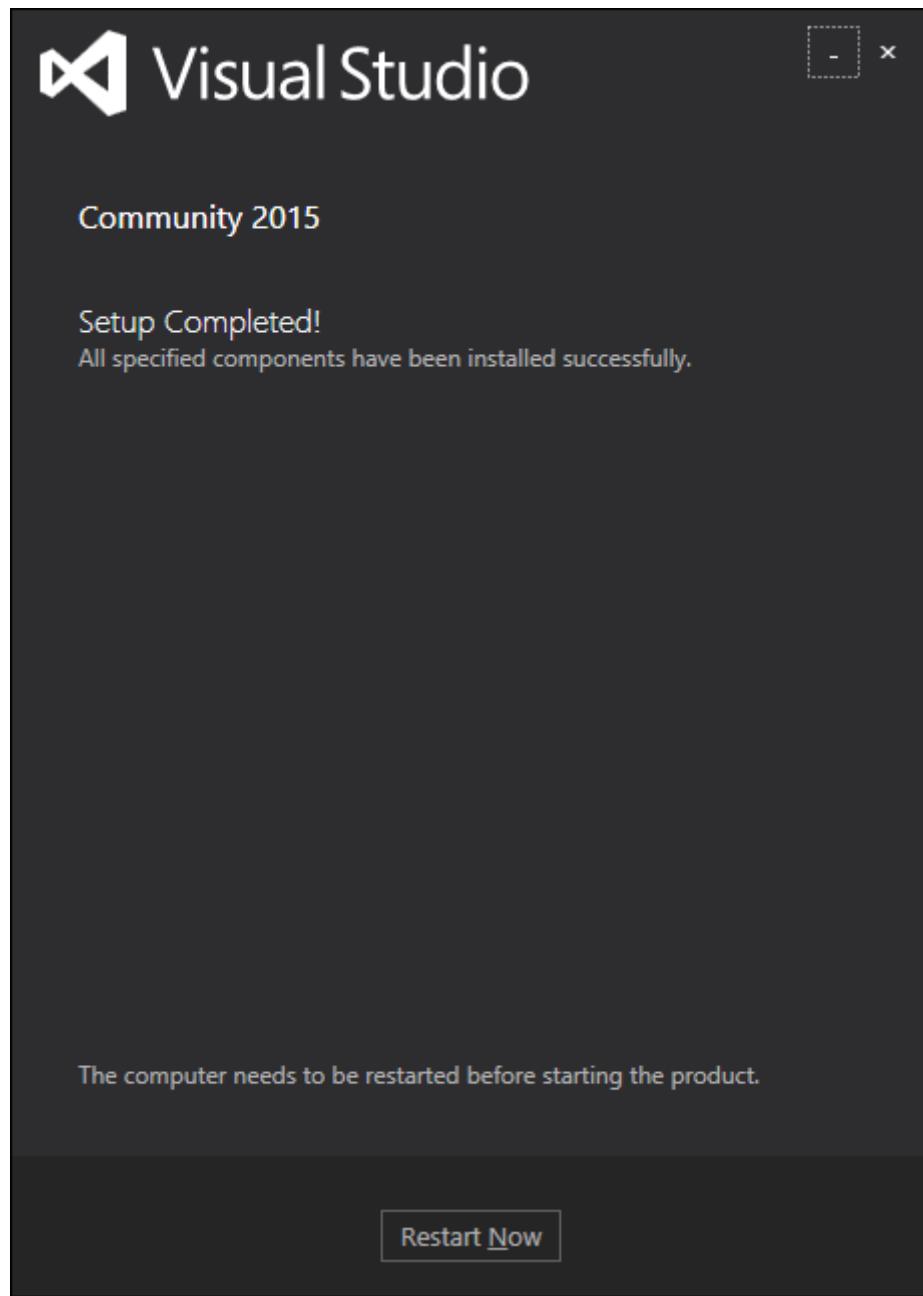
**Step 1:** Once Visual Studio is downloaded, run the installer. The following dialog box will be displayed.



**Step 2:** Click Install to start the installation process.

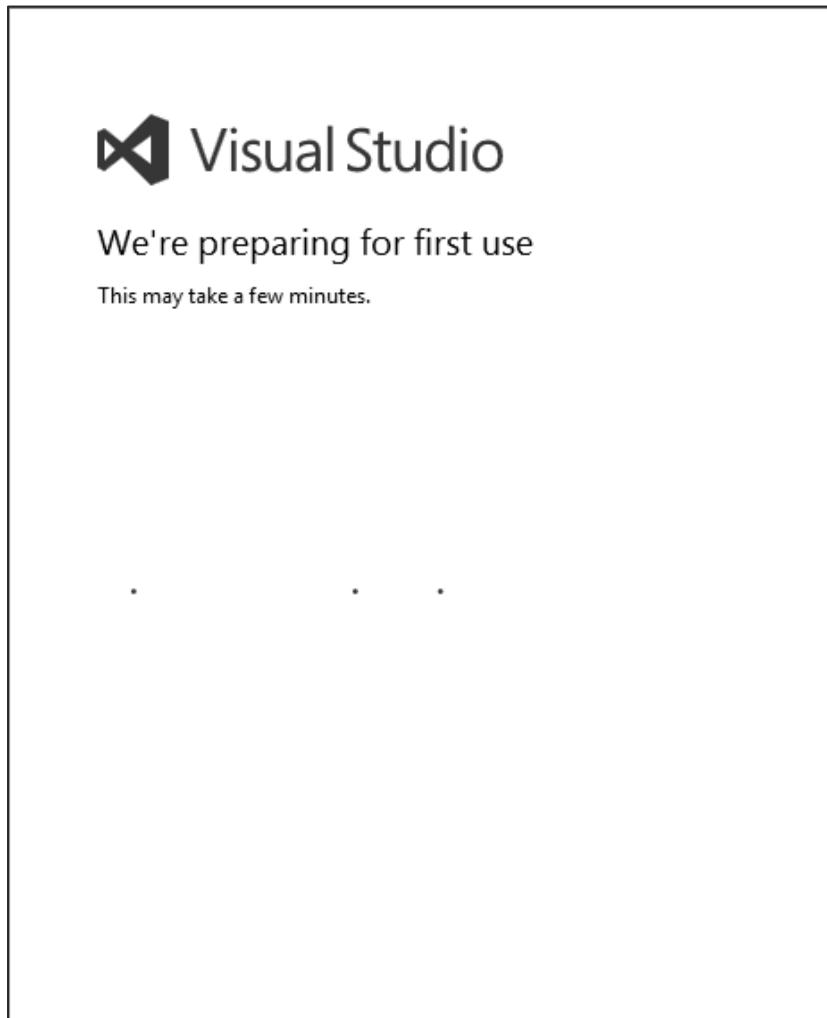


**Step 3:** Once Visual Studio is installed successfully, you will see the following dialog box.

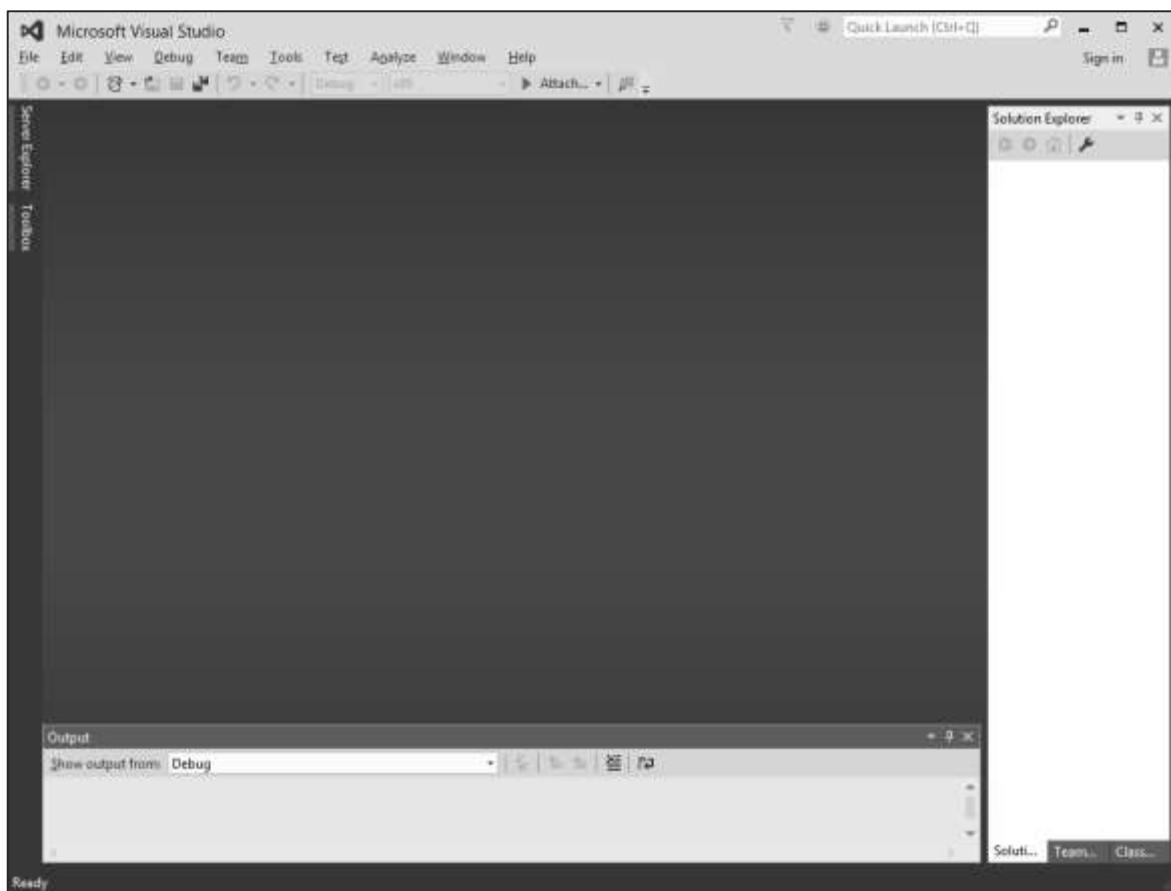


**Step 4:** Close this dialog box and restart your computer if required.

**Step 5:** Open Visual studio from the Start menu, which will open the following dialog box. It will take some time for preparation, while starting for the first time.



**Step 6:** Next, you will see the main window of Visual Studio.



**Step 7:** You are now ready to start your application.

### 3. MFC - VC++ Projects

In this chapter, we will be covering the different types of VC++ projects. Visual Studio includes several kinds of Visual C++ project templates. These templates help to create the basic program structure, menus, toolbars, icons, references, and include statements that are appropriate for the kind of project you want to create. Following are some of the salient features of the templates.

- It provides wizards for many of these project templates and helps you customize your projects as you create them.
- Once the project is created, you can build and run the application.
- You don't have to use a template to create a project, but in most cases it's more efficient to use project templates.
- It's easier to modify the provided project files and structure than it is to create them from scratch.

In MFC, you can use the following project templates.

Project Template	Description
<b>MFC Application</b>	An MFC application is an executable application for Windows that is based on the Microsoft Foundation Class (MFC) Library. The easiest way to create an MFC application is to use the MFC Application Wizard.
<b>MFC ActiveX Control</b>	ActiveX control programs are modular programs designed to give a specific type of functionality to a parent application. For example, you can create a control such as a button for use in a dialog, or toolbar or on a Web page.
<b>MFC DLL</b>	An MFC DLL is a binary file that acts as a shared library of functions that can be used simultaneously by multiple applications. The easiest way to create an MFC DLL project is to use the MFC DLL Wizard.

Following are some General templates which can also be used to create MFC application:

Project Template	Description
<b>Empty Project</b>	Projects are the logical containers for everything that's needed to build your application. You can then add more new or existing projects to the solution if necessary.
<b>Custom Wizard</b>	The Visual C++ Custom Wizard is the tool to use when you need to create a new custom wizard. The easiest way to create a custom wizard is to use the Custom Wizard.

# 4. MFC - Getting Started

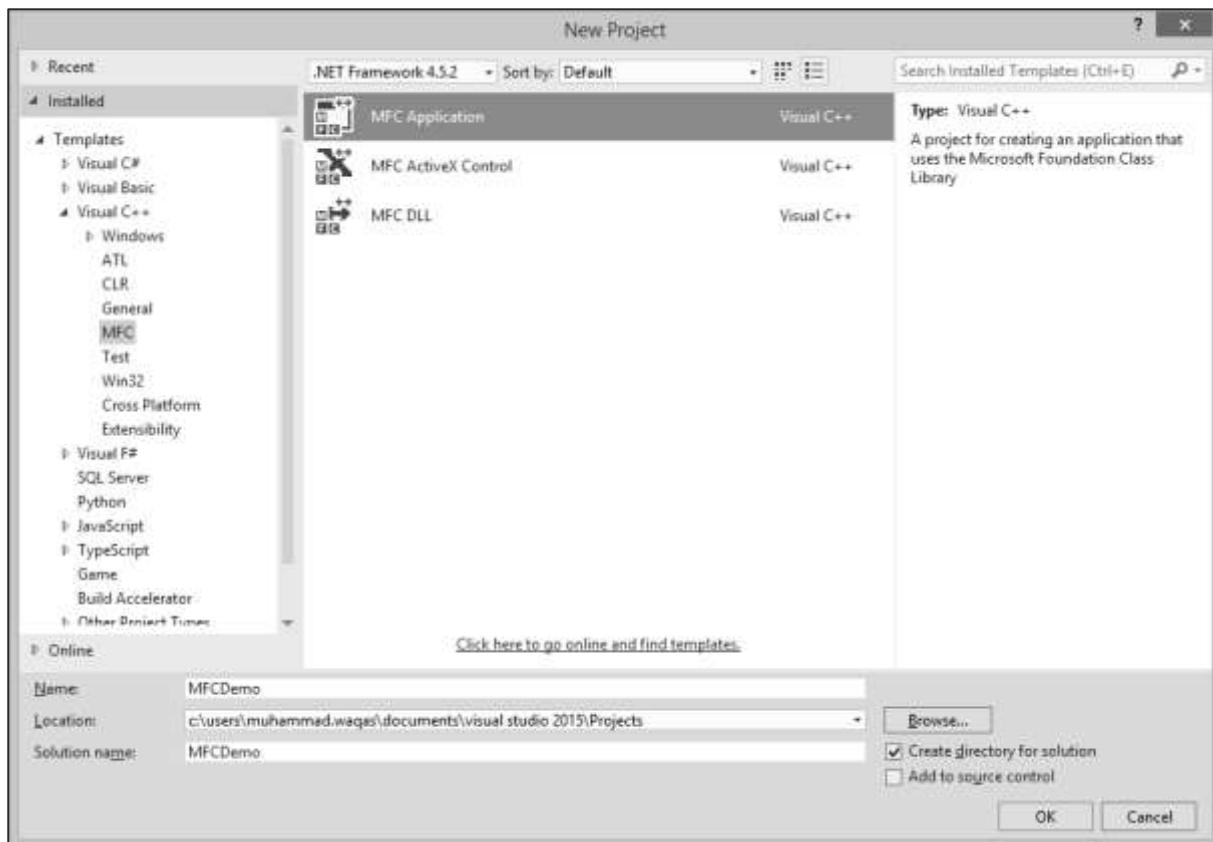
In this chapter, we will look at a working MFC example. To create an MFC application, you can use wizards to customize your projects. You can also create an application from scratch.

## Create Project Using Project Templates

Following are the steps to create a project using project templates available in Visual Studio.

**Step 1:** Open the Visual studio and click on the File -> New -> Project menu option.

**Step 2:** You can now see that the New Project dialog box is open.

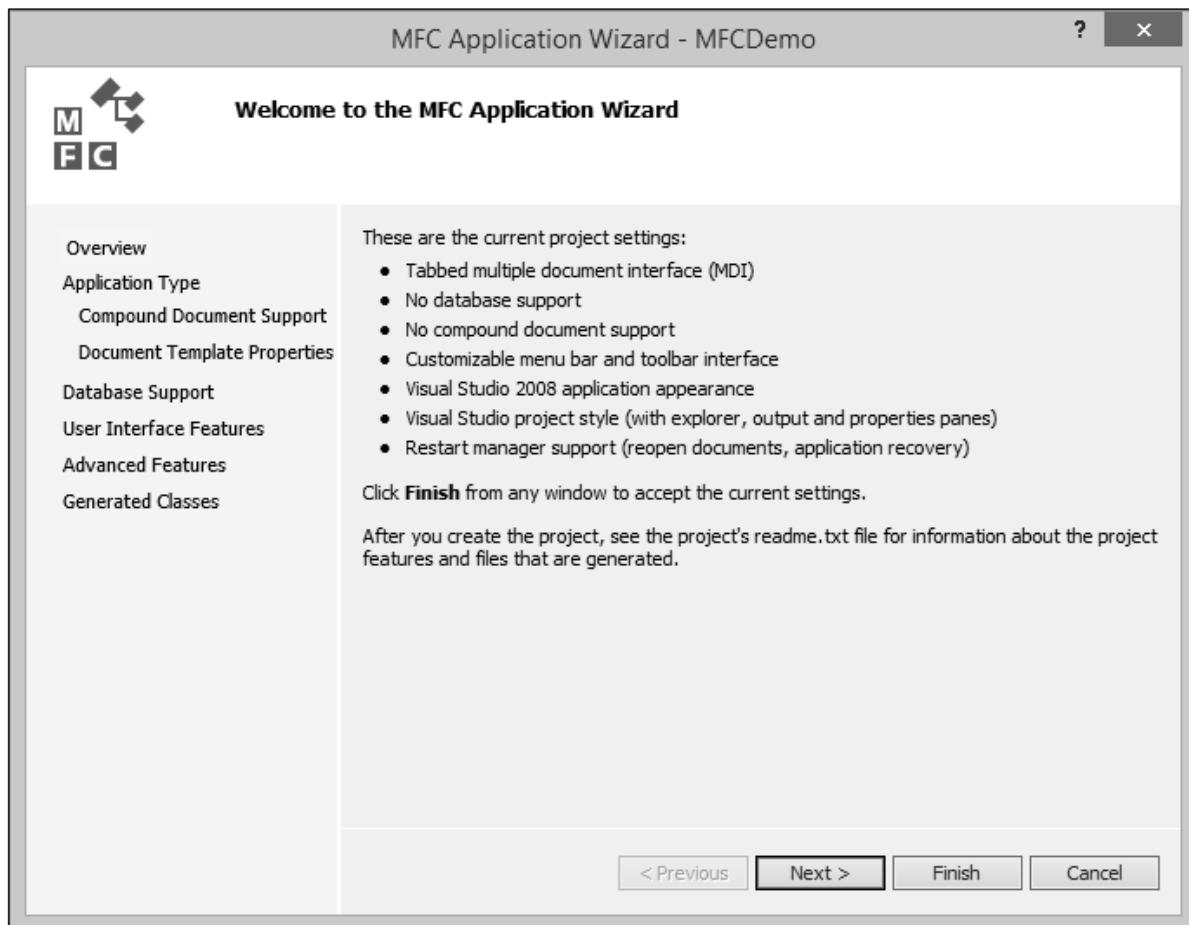


**Step 3:** From the left pane, select Templates -> Visual C++ -> MFC

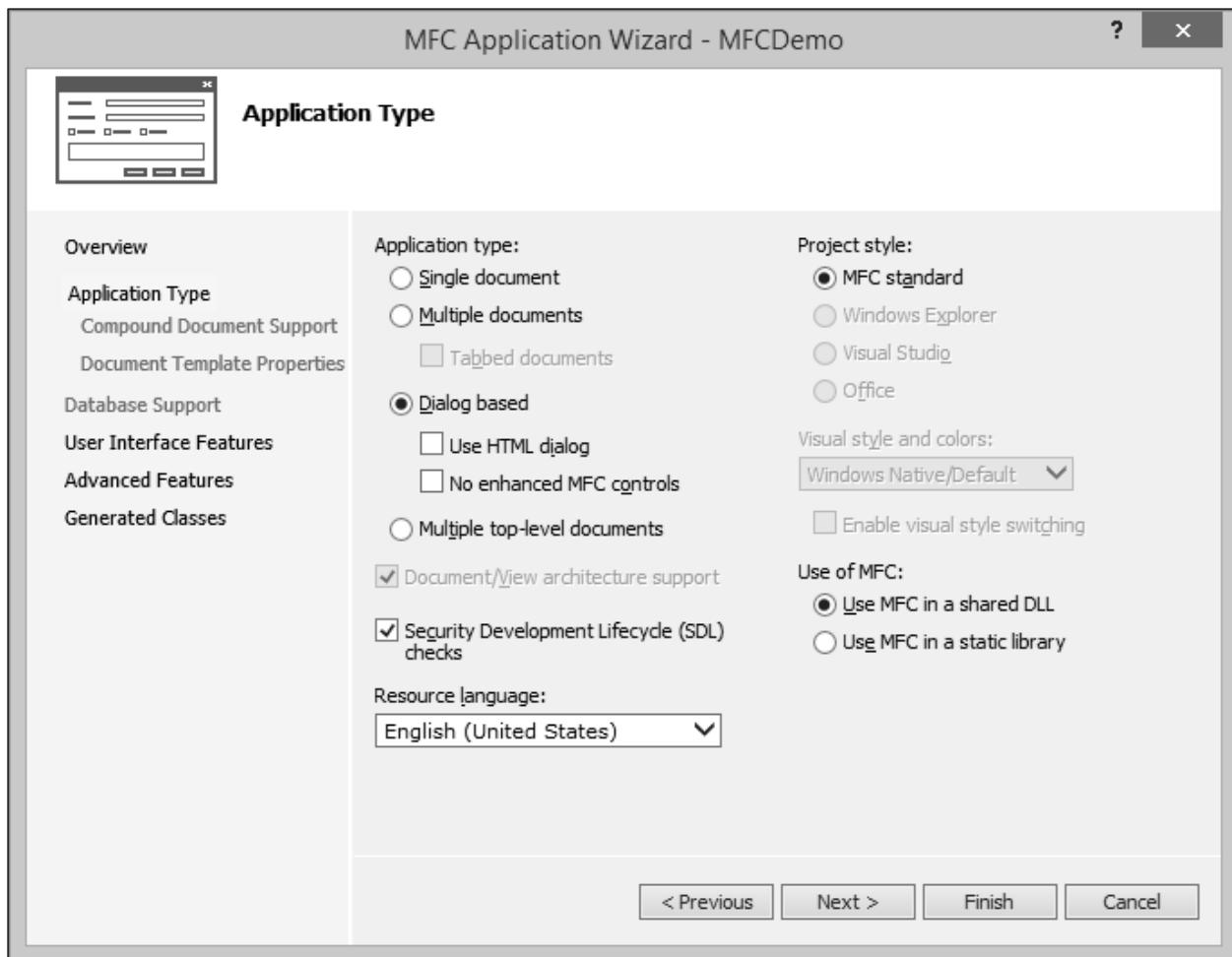
**Step 4:** In the middle pane, select MFC Application.

**Step 5:** Enter the project name 'MFCDemo' in the Name field and click OK to continue.

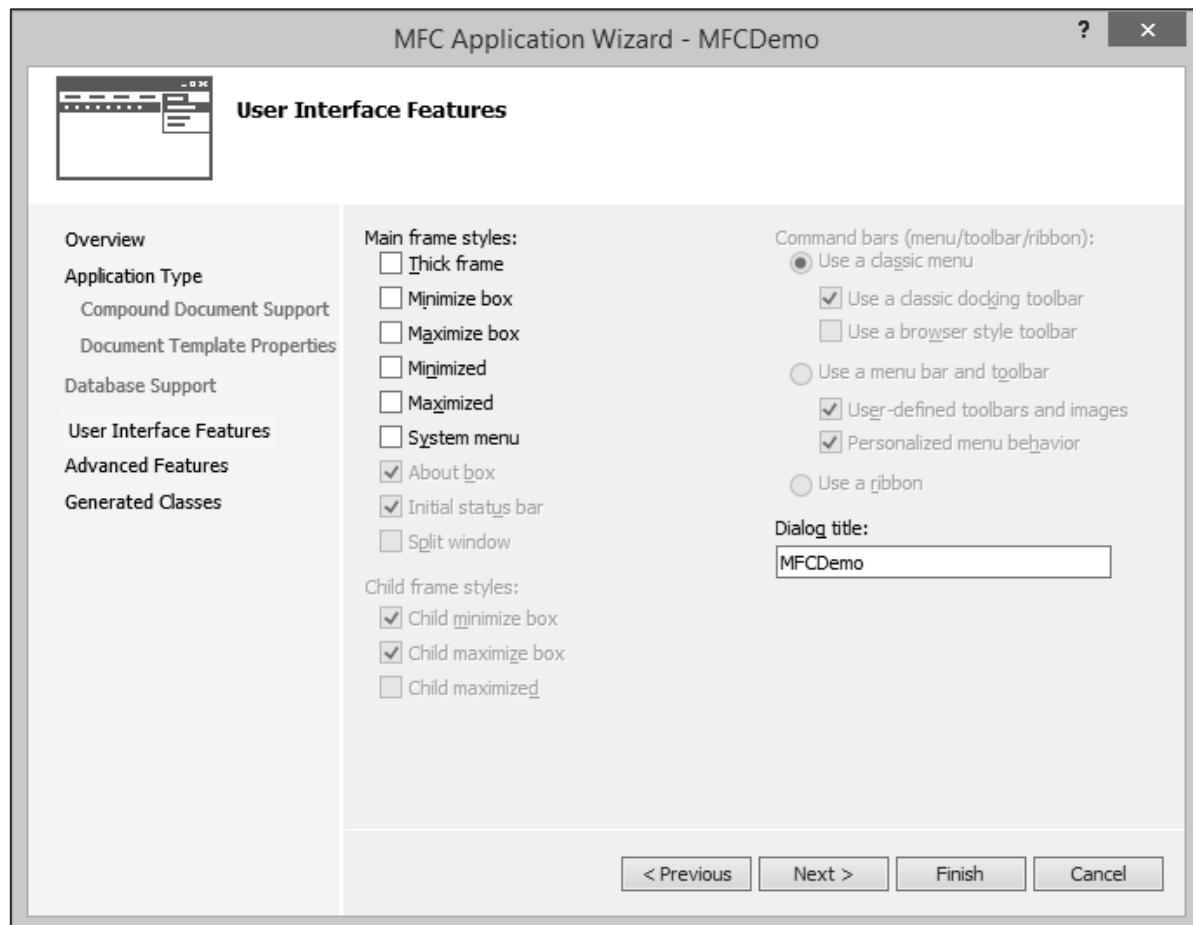
You will see the following dialog.



**Step 6:** Click Next.

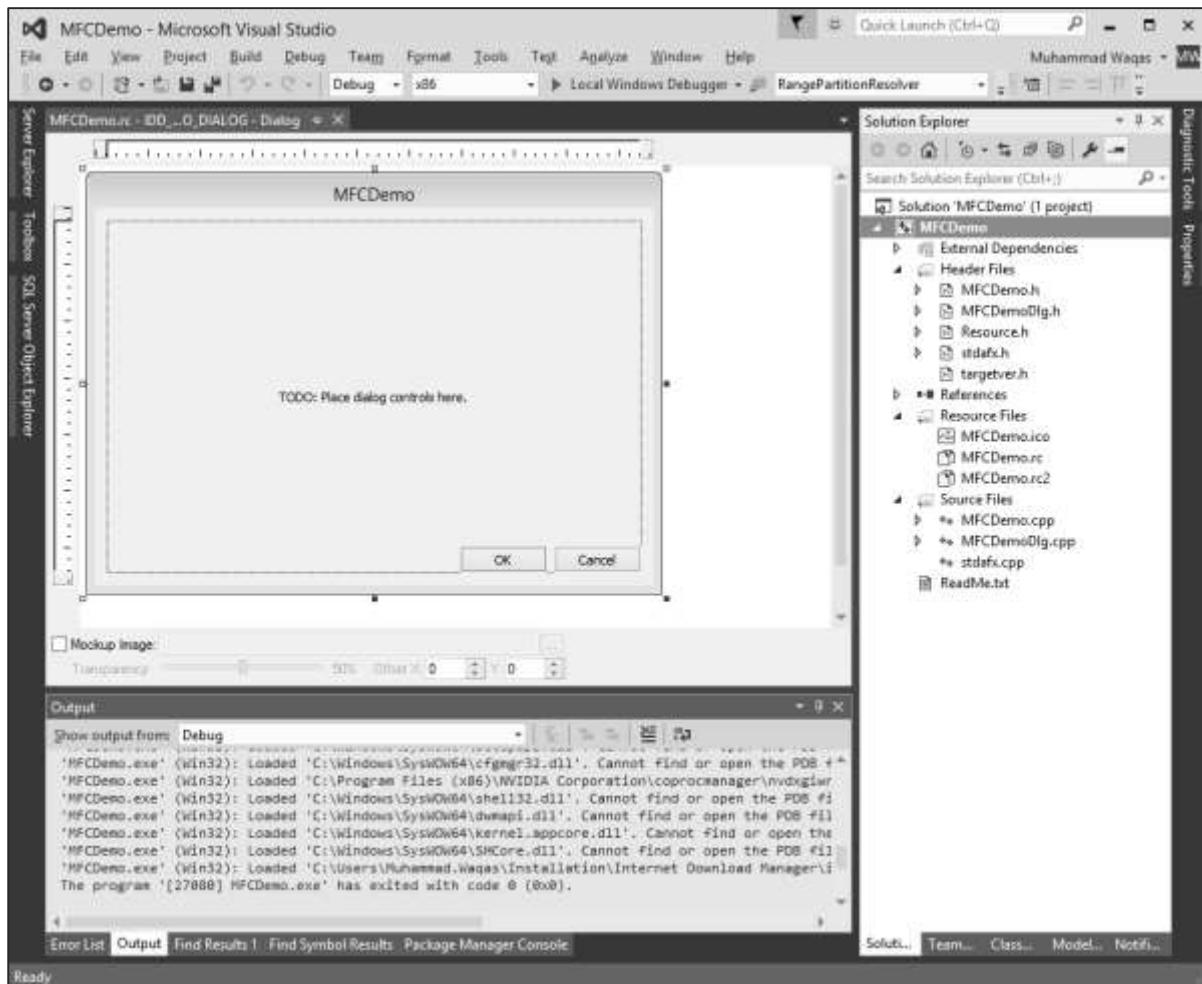


**Step 7:** Select the options which are shown in the dialog box given above and click Next.

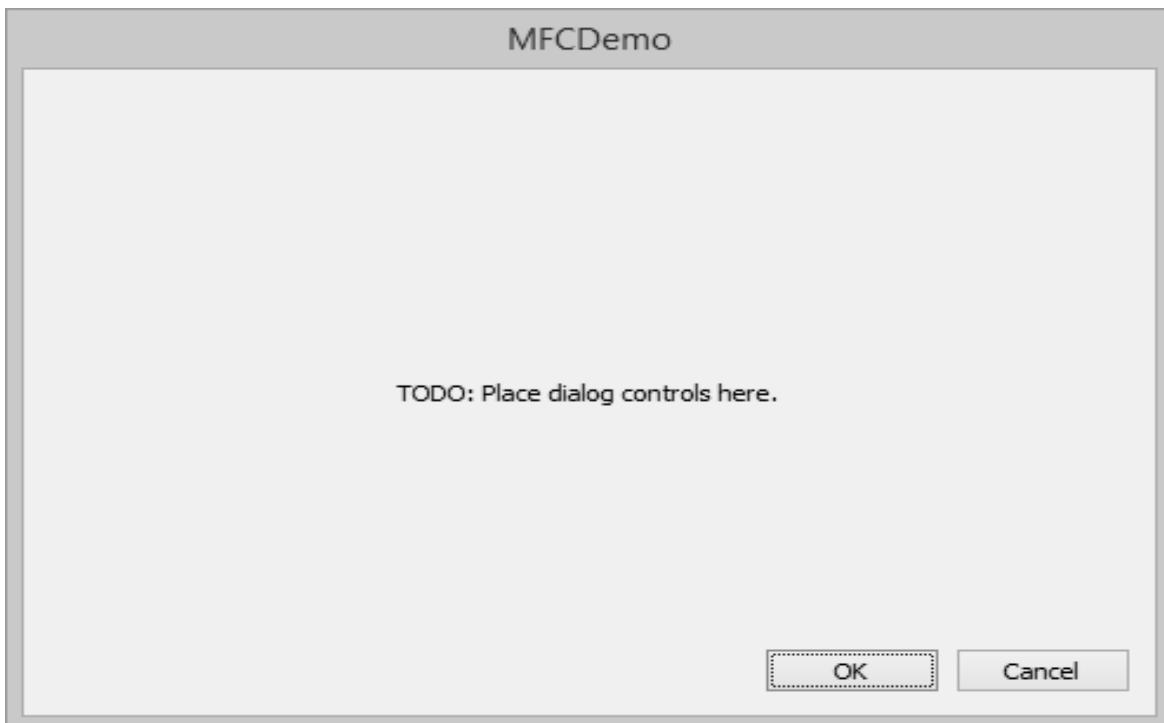


**Step 8:** Uncheck all options and click Finish button.

You can now see that the MFC wizard creates this Dialog Box and the project files by default.



**Step 9:** Run this application, you will see the following output.



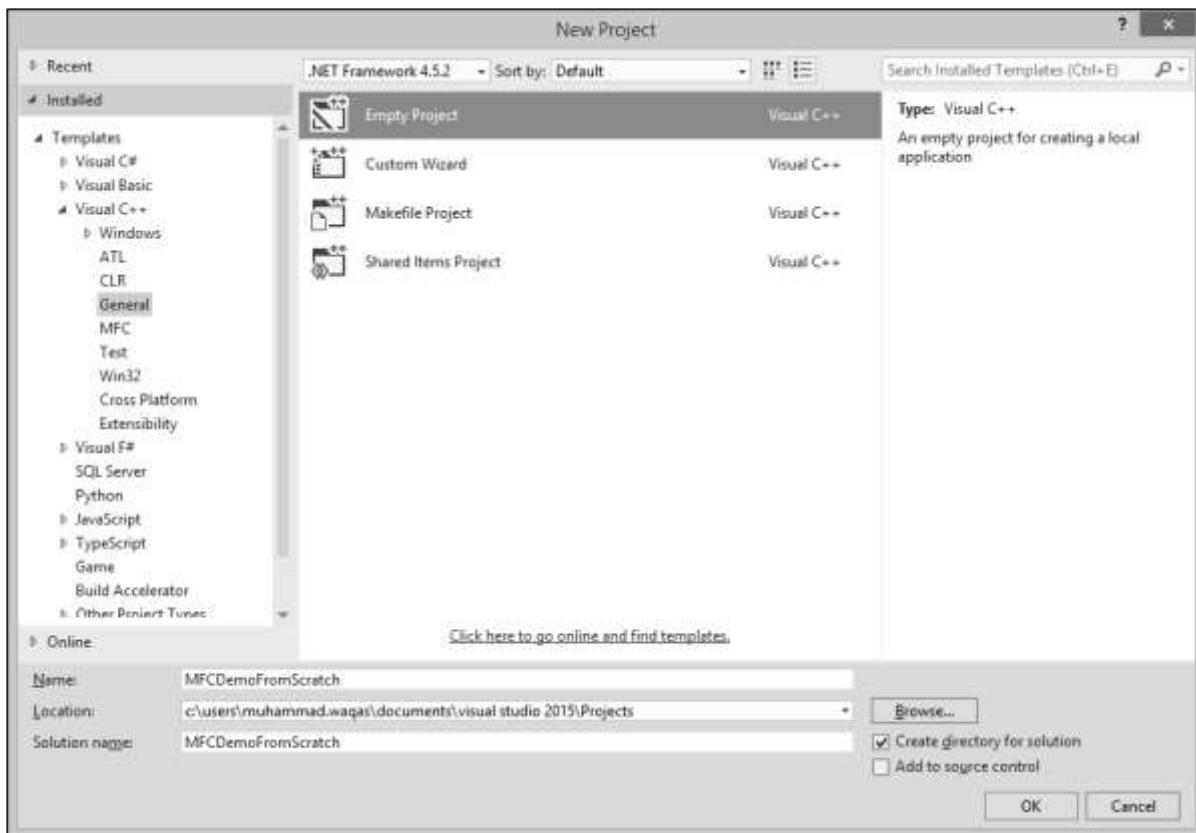
## Create Project from Scratch

---

You can also create an MFC application from scratch. To create an MFC application, you need to follow the following Steps.

**Step 1:** Open the Visual studio and click on the File - > New - > Project menu option.

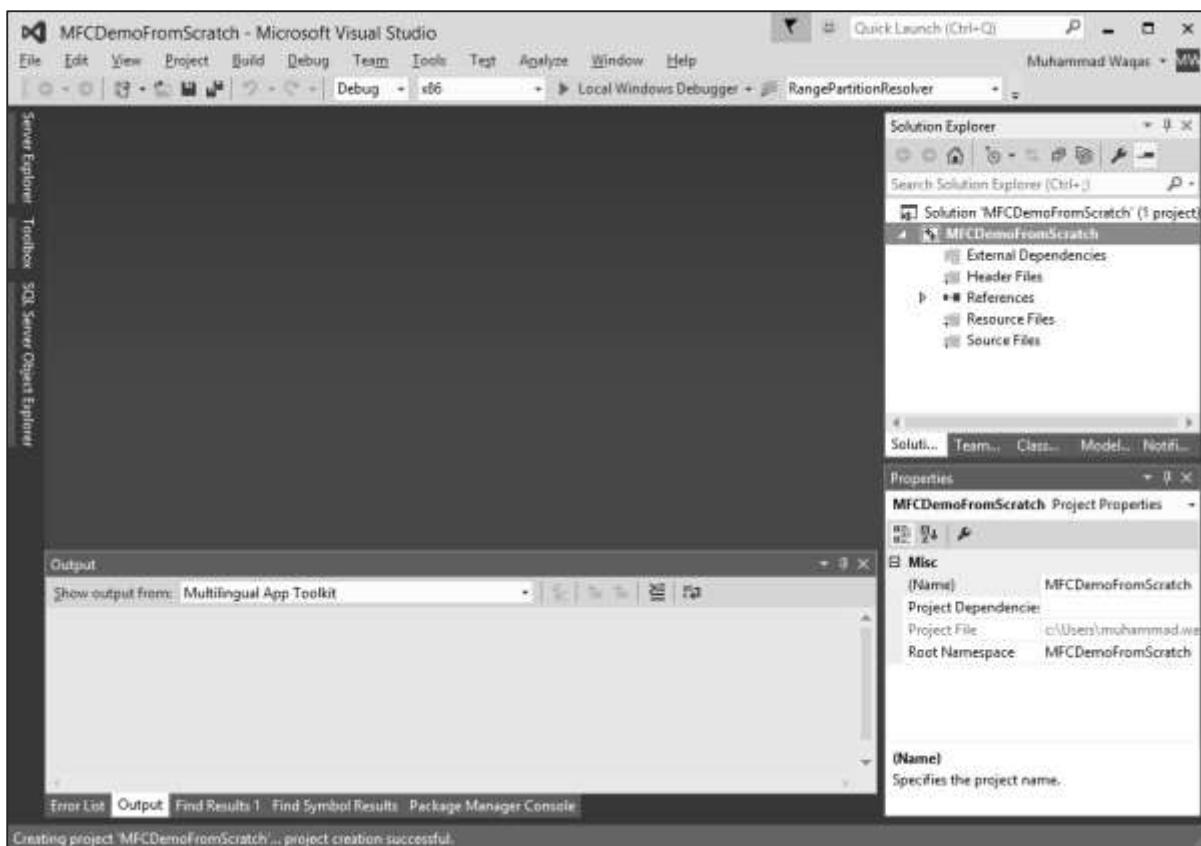
**Step 2:** You can now see the New Project dialog box.



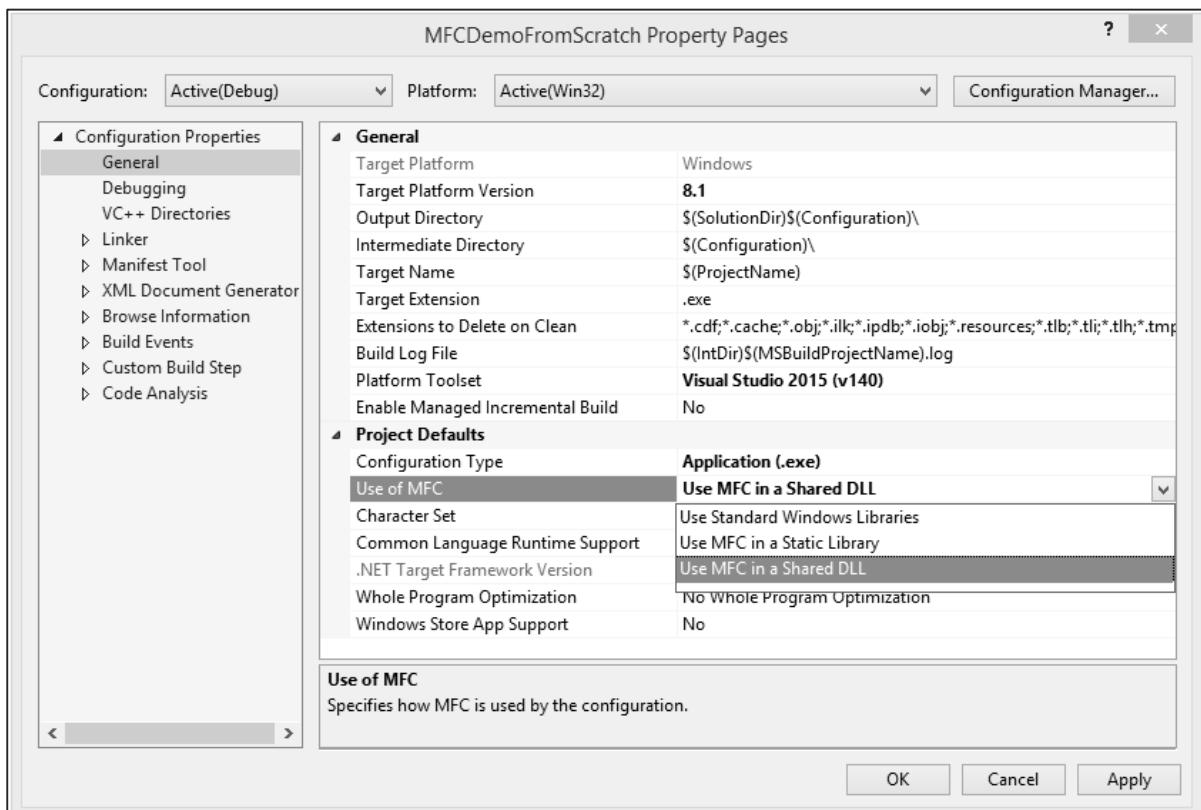
**Step 3:** From the left pane, select Templates -> Visual C++ -> General.

**Step 4:** In the middle pane, select Empty.

**Step 5:** Enter project name 'MFCDemoFromScratch' in the Name field and click OK to continue. You will see that an empty project is created.



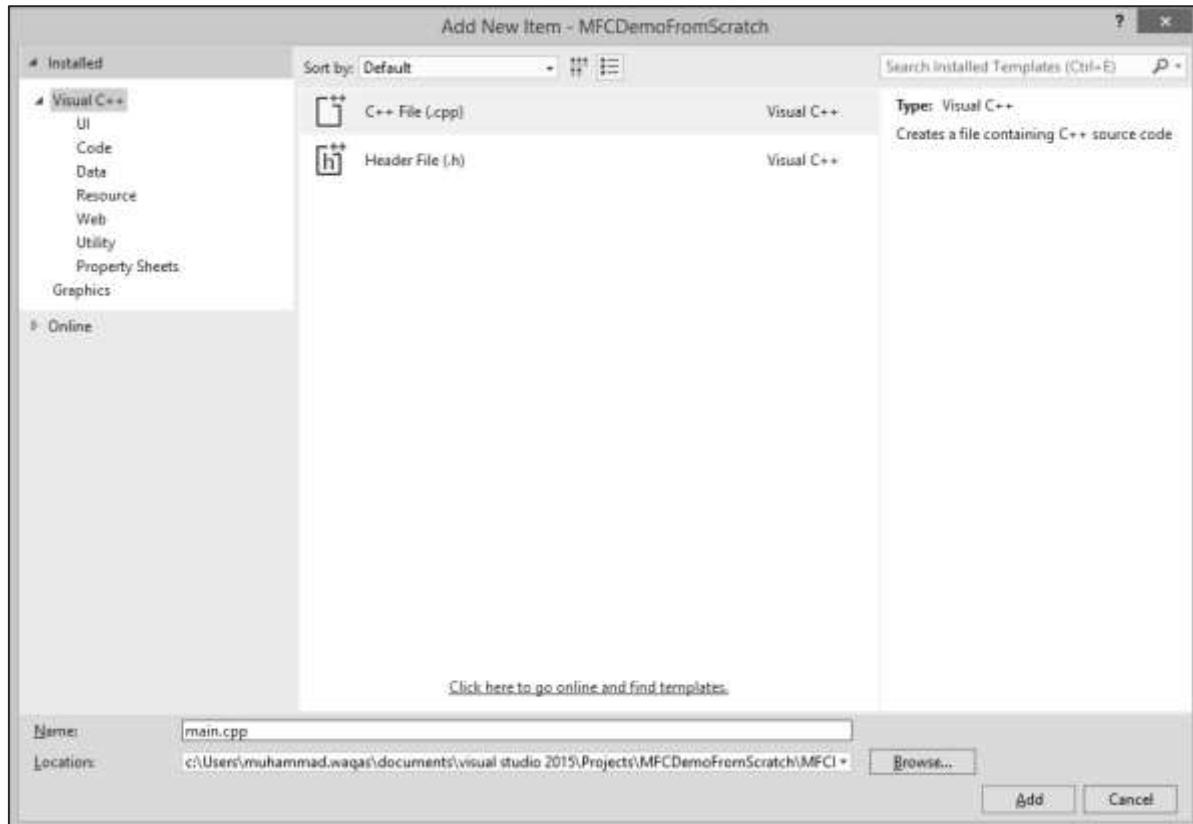
**Step 6:** To make it an MFC project, right-click on the project and select Properties.



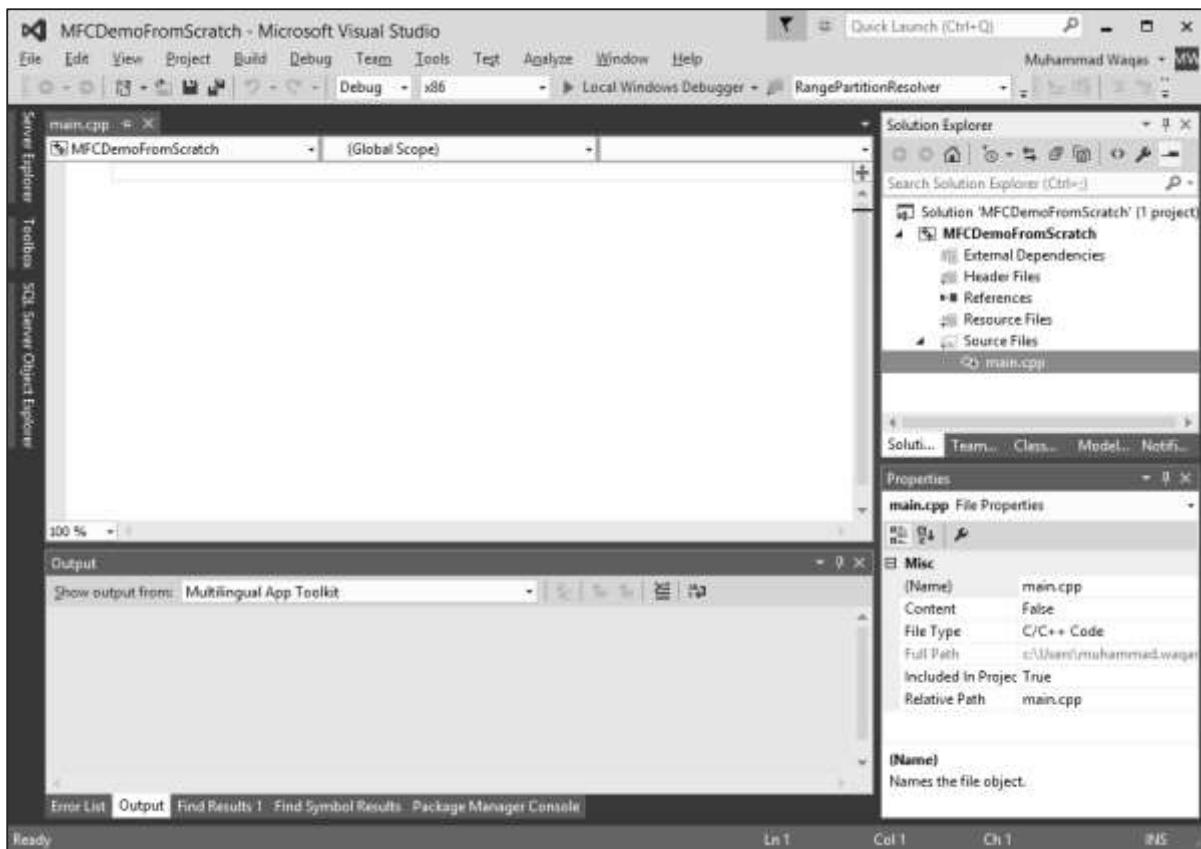
**Step 7:** In the left section, click Configuration Properties -> General.

**Step 8:** Select the Use MFC in Shared DLL option in Project Defaults section and click OK.

**Step 9:** As it is an empty project now; we need to add a C++ file. So, right-click on the project and select Add -> New Item...



**Step 10:** Select **C++ File (.cpp)** in the middle pane and enter file name in the Name field and click Add button.



**Step 11:** You can now see the **main.cpp** file added under the Source Files folder.

**Step 12:** Let us add the following code in this file.

```
#include <iostream>
using namespace std;

void main()
{
    cout << "*****\n";
    cout << "MFC Application Tutorial";
    cout << "\n*****";
    getchar();
}
```

**Step 13:** When you run this application, you will see the following output on console.

```
*****
MFC Application Tutorial
*****
```

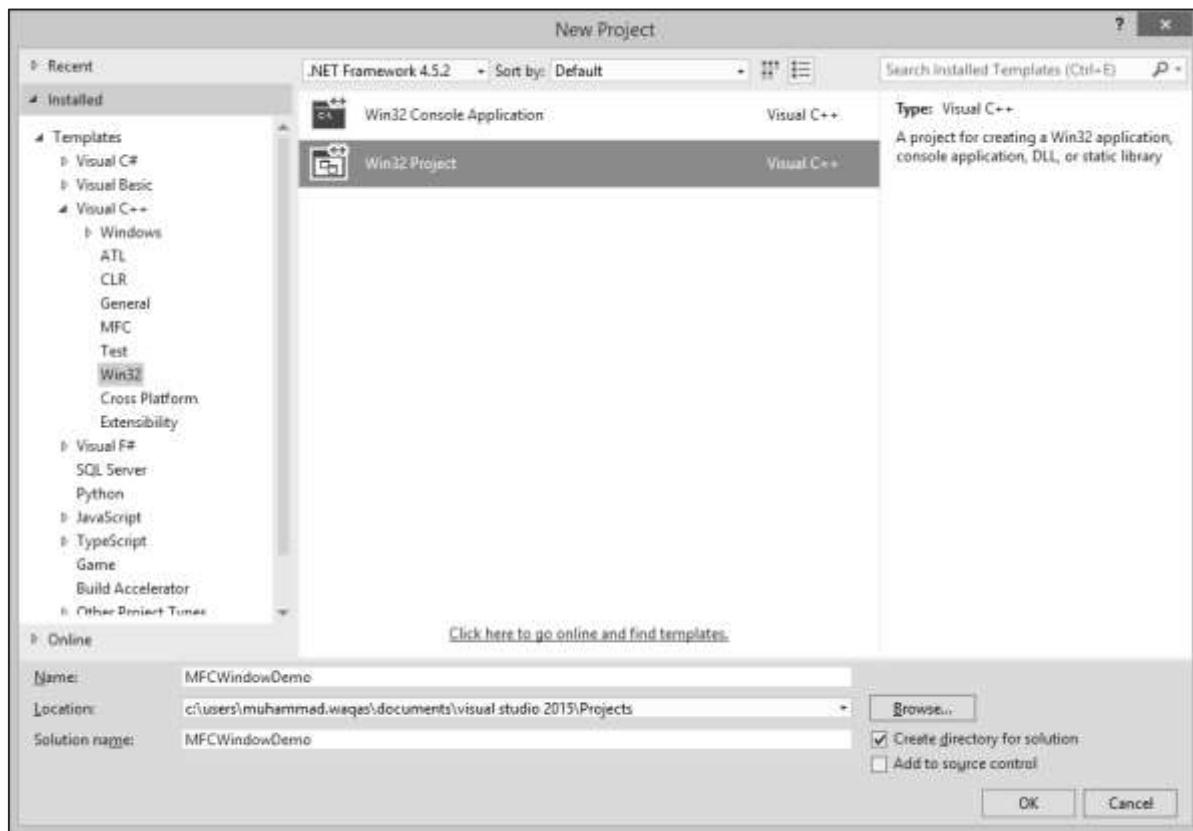
# 5. MFC - Windows Fundamentals

In this chapter, we will be covering the fundamentals of Windows. To create a program, also called an application, you derive a class from the MFC's CWinApp. **CWinApp** stands for **Class for a Windows Application**.

Let us look into a simple example by creating a new Win32 project

**Step 1:** Open the Visual studio and click on the File -> New -> Project menu option.

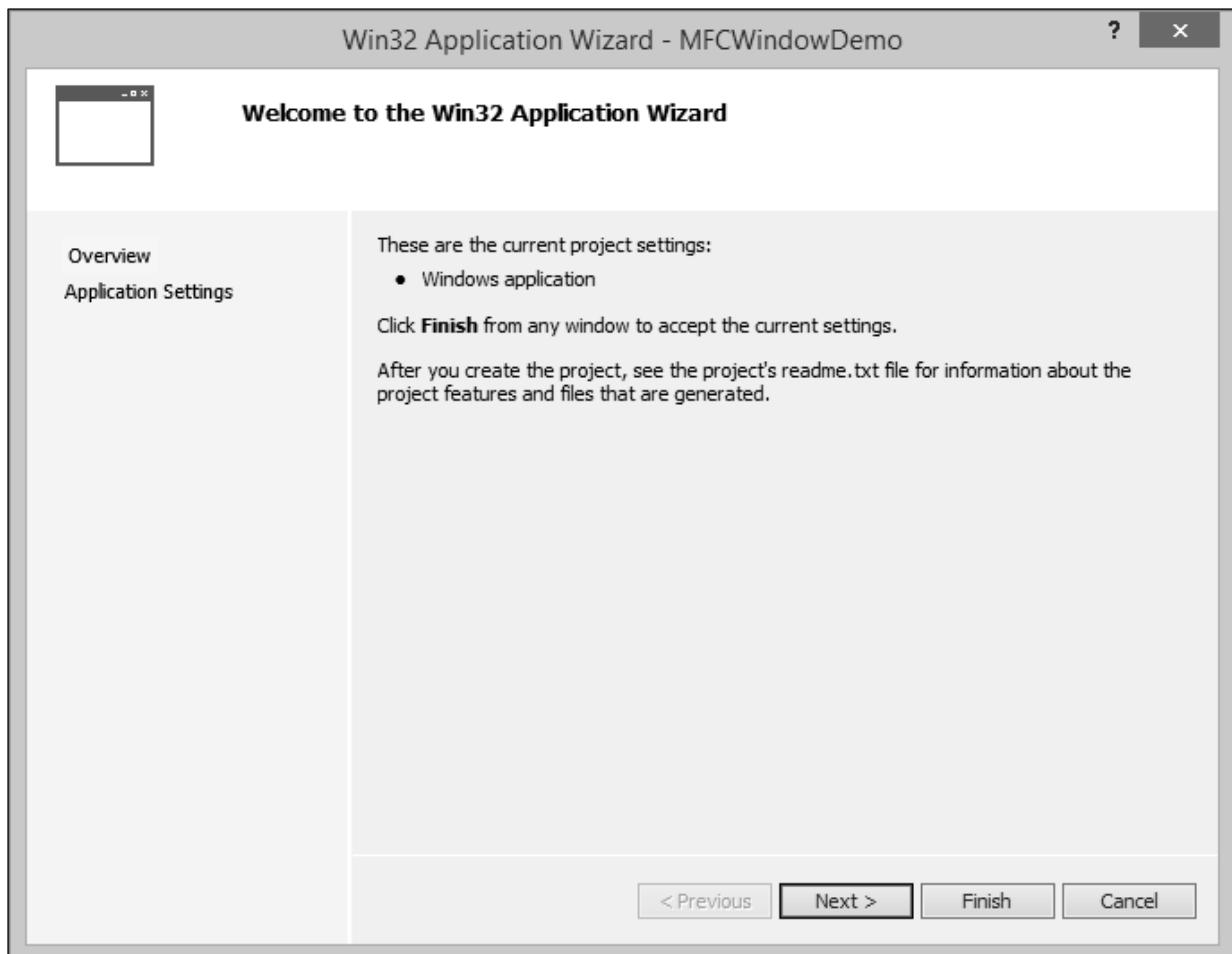
**Step 2:** You can now see the New Project dialog box.



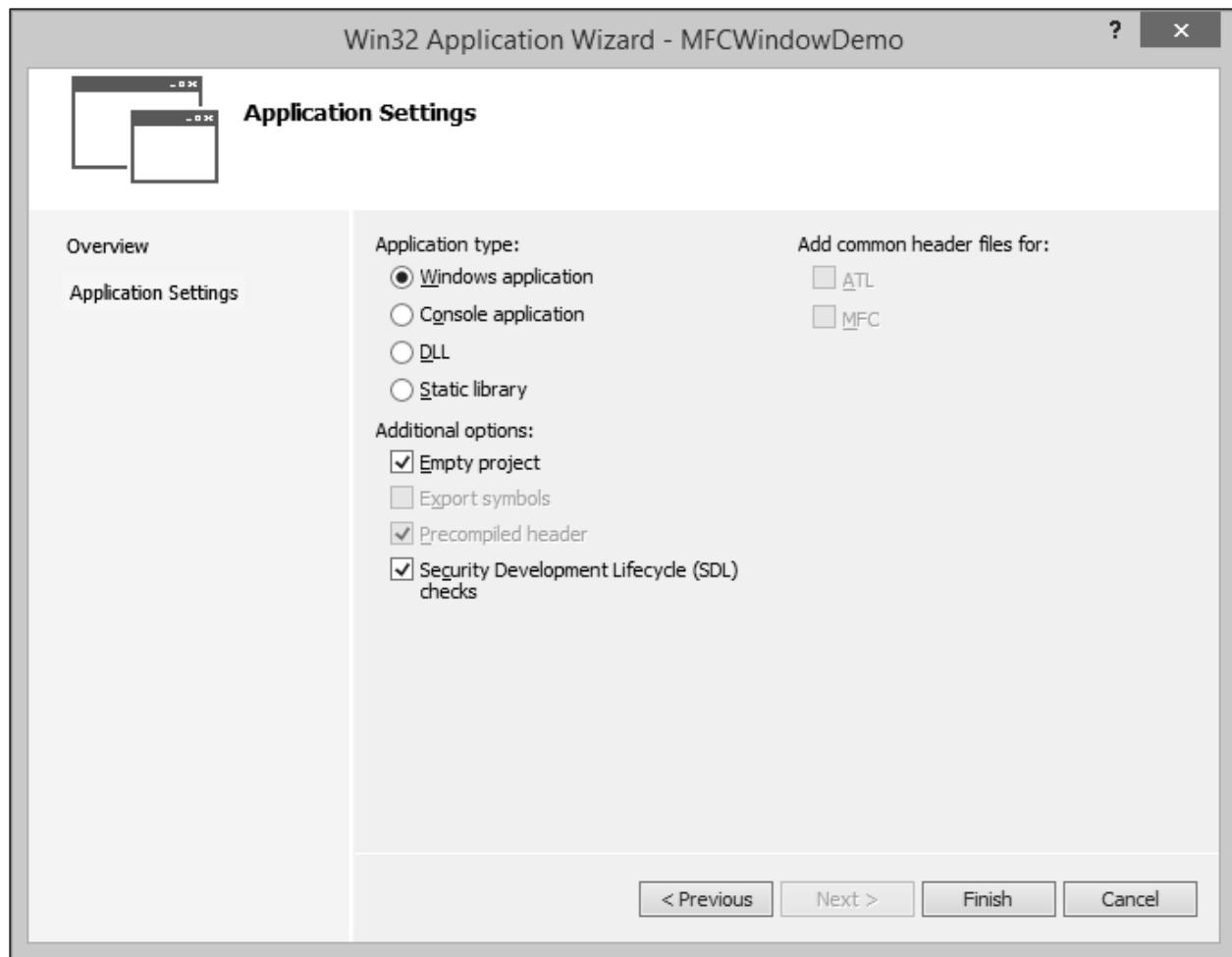
**Step 3:** From the left pane, select Templates -> Visual C++ -> Win32.

**Step 4:** In the middle pane, select Win32 Project.

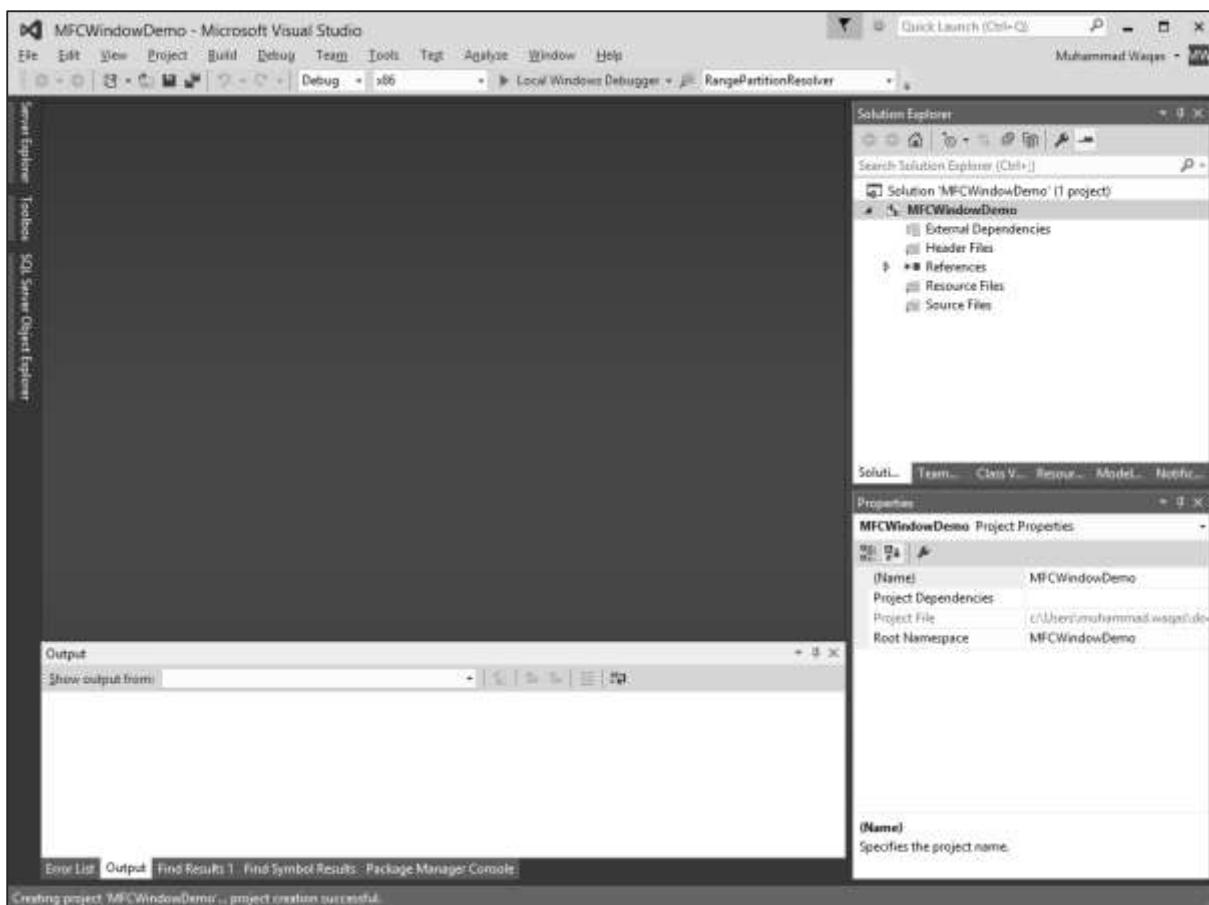
**Step 5:** Enter the project name 'MFCWindowDemo' in the Name field and click OK to continue. You will see the following dialog box.



**Step 6:** Click Next.

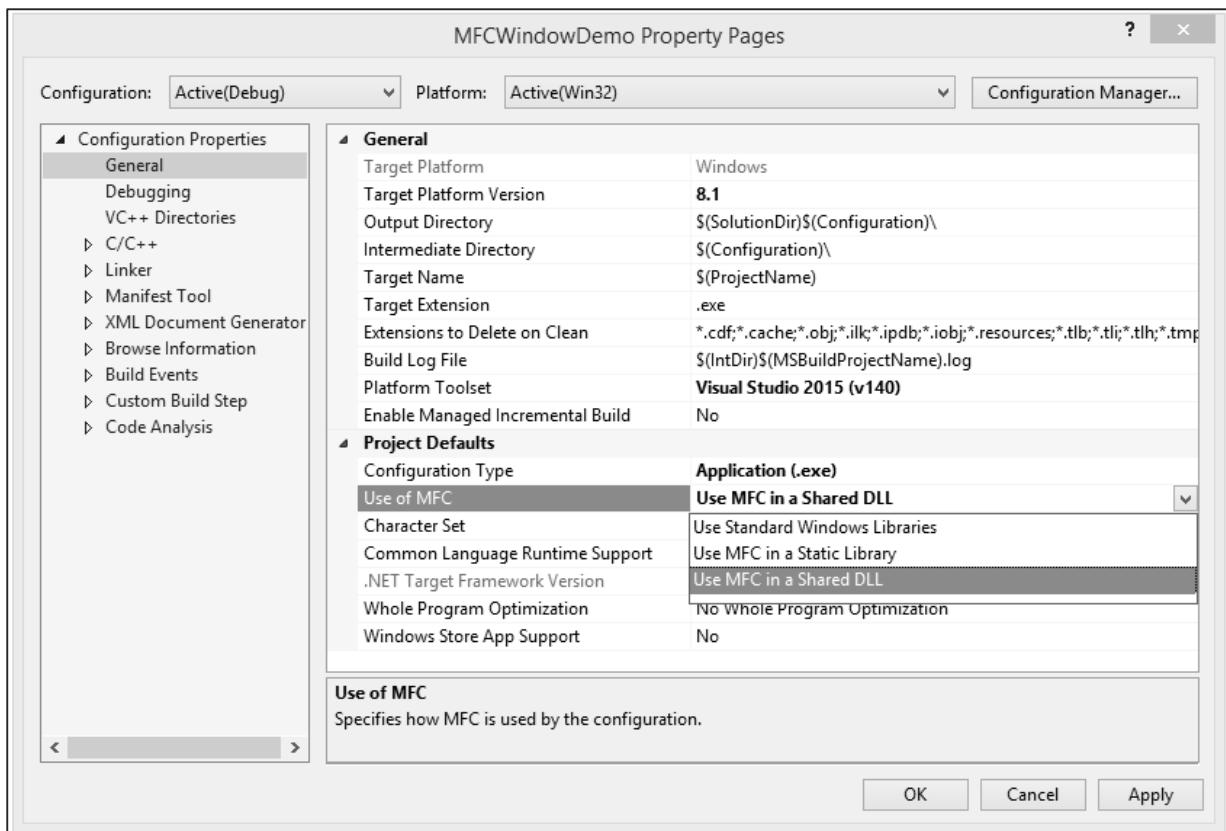


**Step 7:** Select the options as shown in the dialog box given above and click Finish.



**Step 8:** An empty project is created.

**Step 9:** To make it an MFC project, right-click on the project and select Properties.



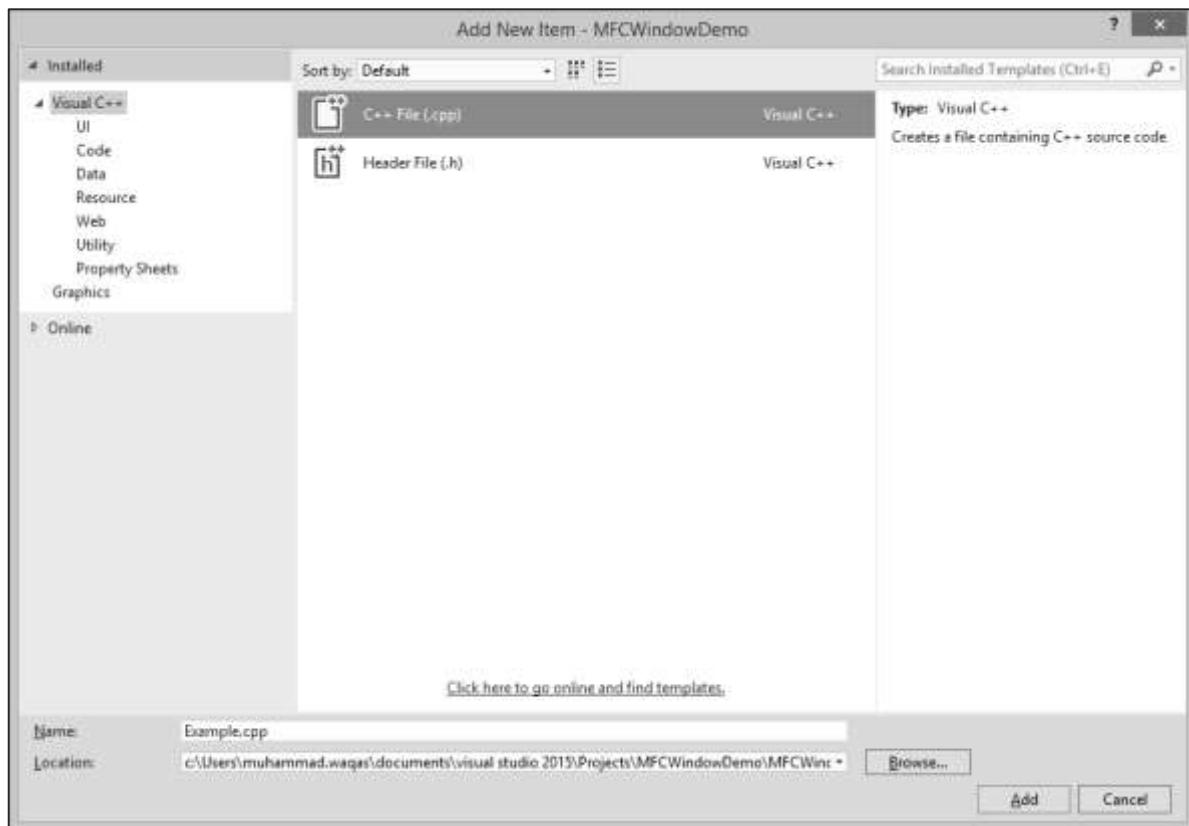
**Step 10:** In the left section, click Configuration Properties -> General.

**Step 11:** Select the Use MFC in Shared DLL option in Project Defaults section and click OK.

**Step 12:** Add a new source file.

**Step 13:** Right-click on your Project and select Add -> New Item...

**Step 14:** In the Templates section, click C++ File (.cpp)



**Step 15:** Set the Name as Example and click Add.

## Window Creation

Any application has two main sections:

- Class
- Frame or Window

Let us create a window using the following steps:

**Step 1:** To create an application, we need to derive a class from the MFC's CWinApp.

```
class CExample : public CWinApp
{
    BOOL InitInstance()
    {
        return TRUE;
    }
};
```

**Step 2:** We also need a frame/window to show the content of our application.

**Step 3:** For this, we need to add another class and derive it from the MFC's **CFrameWnd** class and implement its constructor and a call the Create() method, which will create a frame/window as shown in the following code.

```
class CMyFrame : public CFrameWnd
{
public:
    CMyFrame()
    {
        Create(NULL, _T("MFC Application Tutorial"));
    }
};
```

**Step 4:** As you can see that Create() method needs two parameters, the name of the class, which should be passed as NULL, and the name of the window, which is the string that will be shown on the title bar.

## Main Window

After creating a window, to let the application use it, you can use a pointer to show the class used to create the window. In this case, the pointer would be CFrameWnd. To use the frame window, assign its pointer to the **CWinThread::m\_pMainWnd** member variable. This is done in the **InitInstance()** implementation of your application.

**Step 1:** Here is the implementation of **InitInstance()** in **CExample** class.

```
class CExample : public CWinApp
{
    BOOL InitInstance()
    {
        CMyFrame *Frame = new CMyFrame();
        m_pMainWnd = Frame;

        Frame->ShowWindow(SW_NORMAL);
        Frame->UpdateWindow();

        return TRUE;
    }
};
```

**Step 2:** Following is the complete implementation of Example.cpp file.

```
#include <afxwin.h>

class CMyFrame : public CFrameWnd
{
public:
    CMyFrame()
    {
        Create(NULL, _T("MFC Application Tutorial"));
    }
};

class CExample : public CWinApp
{
    BOOL InitInstance()
    {
        CMyFrame *Frame = new CMyFrame();
        m_pMainWnd = Frame;

        Frame->ShowWindow(SW_NORMAL);
        Frame->UpdateWindow();

        return TRUE;
    }
};

CExample theApp;
```

**Step 3:** When we run the above application, the following window is created.



## Windows Styles

Windows styles are characteristics that control features such as window appearance, borders, minimized or maximized state, or other resizing states, etc. Following is a list of styles which you can use while creating a Window.

Style	Description
<b>WS_BORDER</b>	Creates a window that has a border.
<b>WS_CAPTION</b>	Creates a window that has a title bar (implies the WS_BORDER style). Cannot be used with the WS_DLGFRA ME style.
<b>WS_CHILD</b>	Creates a child window. Cannot be used with the WS_POPUP style.
<b>WS_CHILDWINDOW</b>	Same as the WS_CHILD style.
<b>WS_CLIPCHILDREN</b>	Excludes the area occupied by child windows when you draw within the parent window. Used when you create the parent window.
<b>WS_CLIPSIBLINGS</b>	Clips child windows relative to each other; that is, when a particular child window receives a paint message, the WS_CLIPSIBLINGS style clips all other overlapped child windows out of the region of the child window to be updated. (If WS_CLIPSIBLINGS is not given and child windows overlap, when you draw within the client area of a child window, it is possible to draw within the client area of a neighboring child window.) For use with the WS_CHILD style only.
<b>WS_DISABLED</b>	Creates a window that is initially disabled.
<b>WS_DLGFRA ME</b>	Creates a window with a double border but no title.
<b>WS_GROUP</b>	Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined with the WS_GROUP style FALSE after

	the first control belong to the same group. The next control with the WS_GROUP style starts the next group (that is, one group ends where the next begins).
<b>WS_HSCROLL</b>	Creates a window that has a horizontal scroll bar.
<b>WS_ICONIC</b>	Creates a window that is initially minimized. Same as the WS_MINIMIZE style.
<b>WS_MAXIMIZE</b>	Creates a window of maximum size.
<b>WS_MAXIMIZEBOX</b>	Creates a window that has a Maximize button.
<b>WS_MINIMIZE</b>	Creates a window that is initially minimized. For use with the WS_OVERLAPPED style only.
<b>WS_MINIMIZEBOX</b>	Creates a window that has a Minimize button.
<b>WS_OVERLAPPED</b>	Creates an overlapped window. An overlapped window usually has a caption and a border.
<b>WS_OVERLAPPEDWINDOW</b>	Creates an overlapped window with the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles.
<b>WS_POPUP</b>	Creates a pop-up window. Cannot be used with the WS_CHILD style.
<b>WS_POPUPWINDOW</b>	Creates a pop-up window with the WS_BORDER, WS_POPUP, and WS_SYSMENU styles. The WS_CAPTION style must be combined with the WS_POPUPWINDOW style to make the Control menu visible.
<b>WS_SIZEBOX</b>	Creates a window that has a sizing border. Same as the WS_THICKFRAME style.
<b>WS_SYSMENU</b>	Creates a window that has a Control-menu box in its title bar. Used only for windows with title bars.
<b>WS_TABSTOP</b>	Specifies one of any number of controls through which the user can move by using the TAB key. The TAB key moves the user to the next control specified by the WS_TABSTOP style.
<b>WS_THICKFRAME</b>	Creates a window with a thick frame that can be used to size the window.
<b>WS_TILED</b>	Creates an overlapped window. An overlapped window has a title bar and a border. Same as the WS_OVERLAPPED style.
<b>WS_TILEDWINDOW</b>	Creates an overlapped window with the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. Same as the WS_OVERLAPPEDWINDOW style.
<b>WS_VISIBLE</b>	Creates a window that is initially visible.
<b>WS_VSCROLL</b>	Creates a window that has a vertical scroll bar.

**Step 1:** Let us look into a simple example in which we will add some styling. After creating a window, to display it to the user, we can apply the WS\_VISIBLE style to it and additionally, we will also add WS\_OVERLAPPED style. Here is an implementation:

```
class CMyFrame : public CFrameWnd
{
public:
    CMyFrame()
    {

```

```

        Create(NULL, _T("MFC Application Tutorial"), WS_VISIBLE |  

WS_OVERLAPPED);  

    }  

};
```

**Step 2:** When you run this application, the following window is created.



You can now see that the minimize, maximize, and close options do not appear anymore.

## Windows Location

To locate things displayed on the monitor, the computer uses a coordinate system similar to the Cartesian's, but the origin is located on the top left corner of the screen. Using this coordinate system, any point can be located by its distance from the top left corner of the screen of the horizontal and the vertical axes.

The **Win32 library** provides a structure called POINT defined as follows:

```

typedef struct tagPOINT {  

    LONG x;  

    LONG y;  

} POINT;
```

- The 'x' member variable is the distance from the left border of the screen to the point.

- The 'y' variable represents the distance from the top border of the screen to the point.
- Besides the Win32's POINT structure, the Microsoft Foundation Class (MFC) library provides the CPoint class.
- This provides the same functionality as the POINT structure. As a C++ class, it adds more functionality needed to locate a point. It provides two constructors.

```
CPoint();
CPoint(int X, int Y);
```

## Windows Size

---

While a point is used to locate an object on the screen, each window has a size. The size provides two measures related to an object.

- The width of an object.
- The height of an object.

The Win32 library uses the SIZE structure defined as follows:

```
typedef struct tagSIZE {
    int cx;
    int cy;
} SIZE;
```

Besides the Win32's SIZE structure, the MFC provides the CSize class. This class has the same functionality as SIZE but adds features of a C++ class. It provides five constructors that allow you to create a size variable in any way of your choice.

```
CSize();
CSize(int initCX, int initCY);
CSize(SIZE initSize);
CSize(POINT initPt);
CSize(DWORD dwSize);
```

## Windows Dimensions

When a Window displays, it can be identified on the screen by its location with regards to the borders of the monitor. A Window can also be identified by its width and height. These characteristics are specified or controlled by the *rect* argument of the **Create()** method. This argument is a rectangle that can be created through the Win32 RECT structure.

```
typedef struct _RECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT, *PRECT;
```

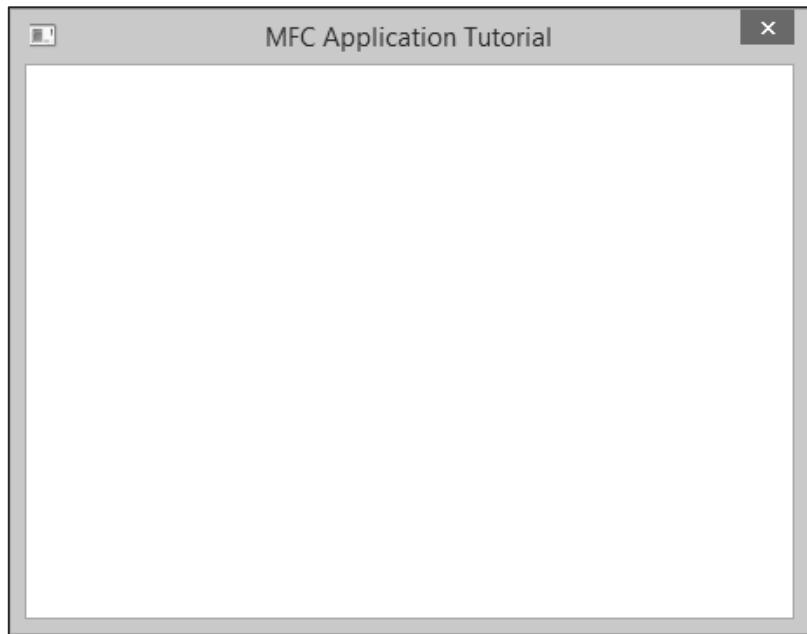
Besides the Win32's **RECT** structure, the MFC provides the CRect class which has the following constructors:

```
CRect();
CRect(int l, int t, int r, int b);
CRect(const RECT& srcRect);
CRect(LPCRECT lpSrcRect);
CRect(POINT point, SIZE size);
CRect(POINT topLeft, POINT bottomRight);
```

Let us look into a simple example in which we will specify the location and the size of the window

```
class CMyFrame : public CFrameWnd
{
public:
    CMyFrame()
    {
        Create(NULL, _T("MFC Application Tutorial"), WS_SYSMENU, CRect(90,
120, 550, 480));
    }
};
```

When you run this application, the following window is created on the top left corner of your screen as specified in CRect constructor in the first two parameters. The last two parameters are the size of the Window.



## Windows Parents

---

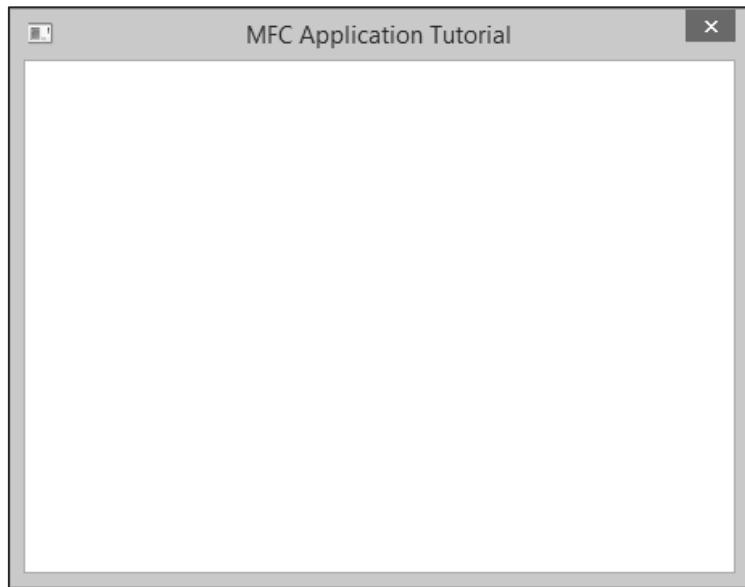
In the real world, many applications are made of different Windows. When an application uses various Windows, most of the objects depend on a particular one. It could be the first Window that was created or another window that you designated. Such a Window is referred to as the **Parent Window**. All the other windows depend on it directly or indirectly.

- If the Window you are creating is dependent of another, you can specify that it has a parent.
- This is done with the pParentWnd argument of the CFrameWnd::Create() method.
- If the Window does not have a parent, pass the argument with a NULL value.

Let us look into an example which has only one Window, and there is no parent Window available, so we will pass the argument with NULL value as shown in the following code:

```
class CMyFrame : public CFrameWnd
{
public:
    CMyFrame()
    {
        Create(NULL, _T("MFC Application Tutorial"), WS_SYSMENU, CRect(90,
120, 550, 480), NULL);
    }
};
```

When you run the above application, you see the same output.



# 6. MFC - Dialog Boxes

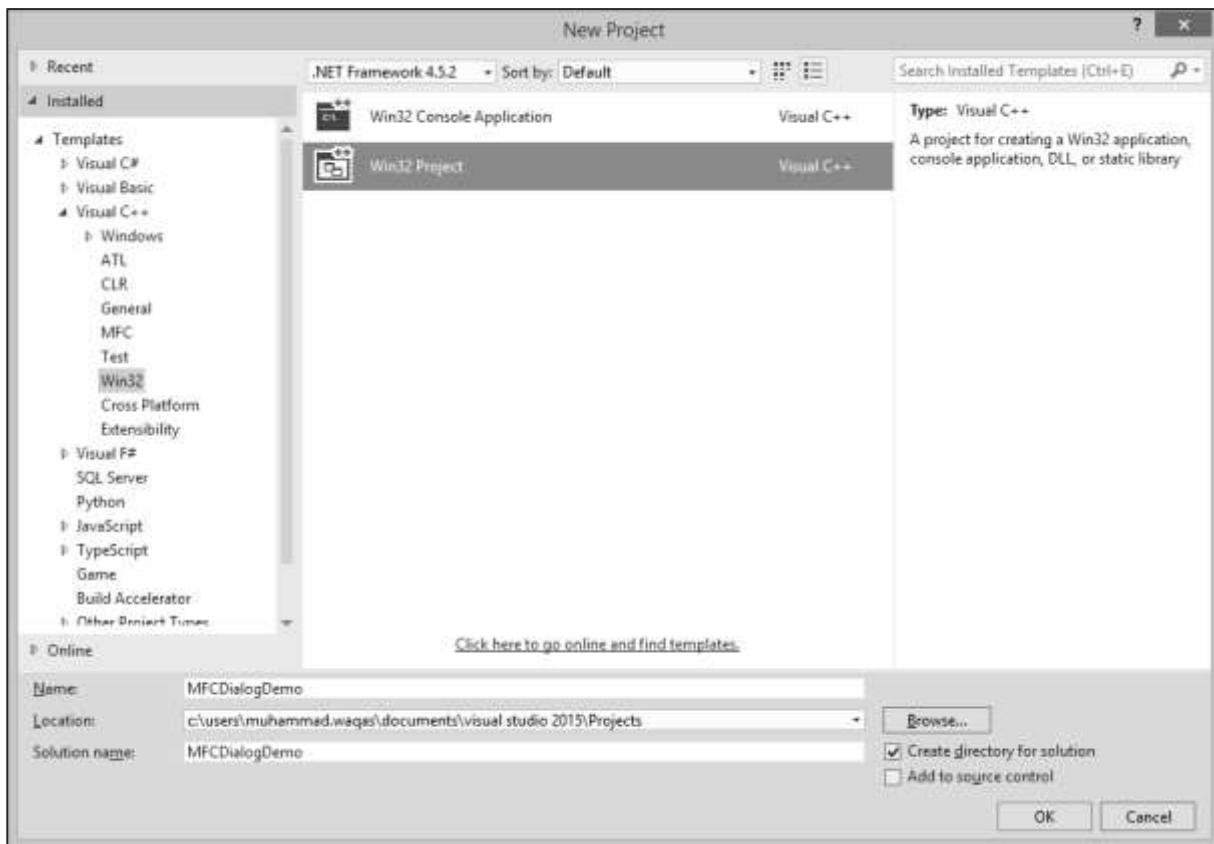
In this chapter, we will be covering the Dialog boxes. Applications for Windows frequently communicate with the user through dialog boxes. **CDialog class** provides an interface for managing dialog boxes. The Visual C++ dialog editor makes it easy to design dialog boxes and create their dialog-template resources.

- Creating a dialog object is a two-phase operation:
  - Construct the dialog object.
  - Create the dialog window.

Let us look into a simple example by creating a new Win32 project

**Step 1:** Open the Visual studio and click on the File -> New -> Project menu option.

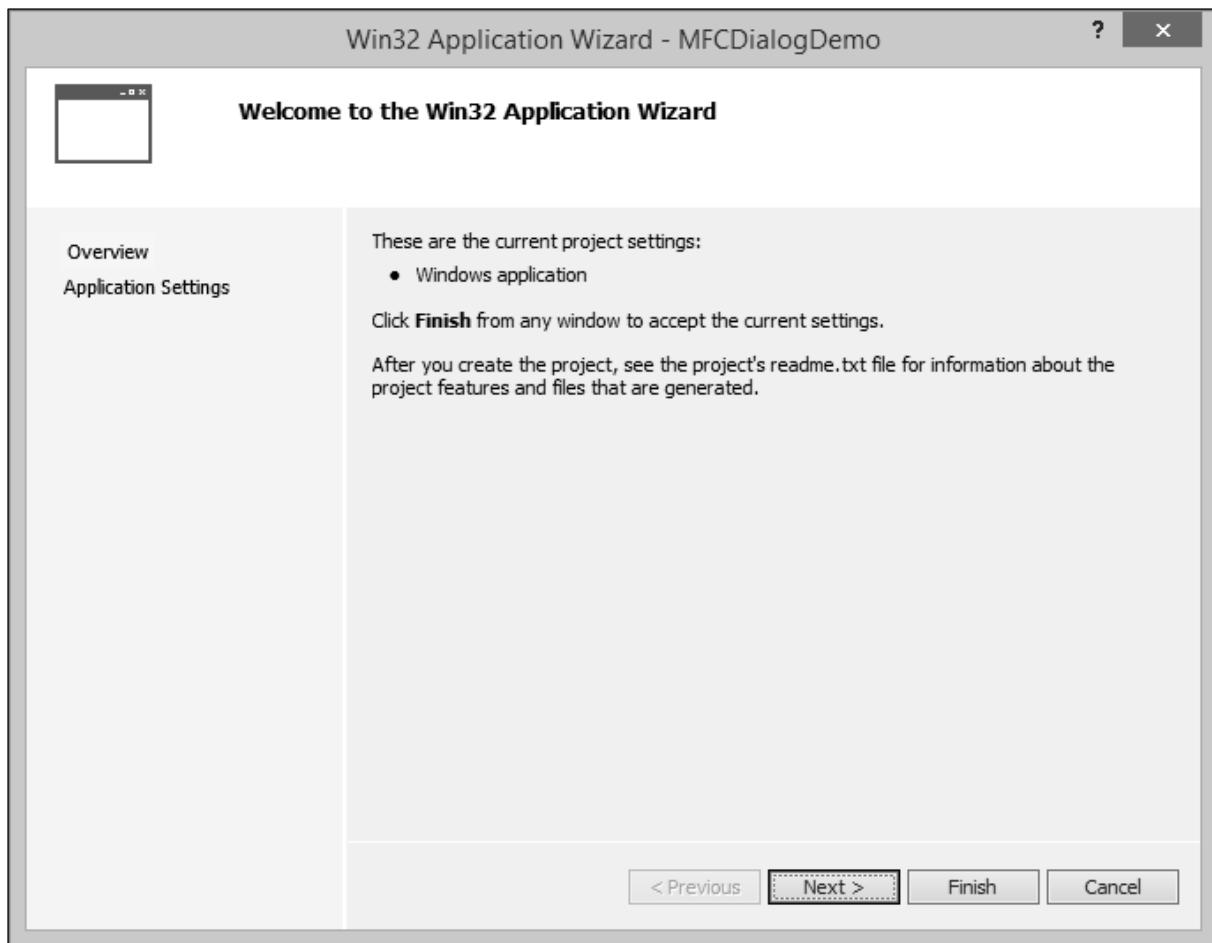
**Step 2:** You can now see the New Project dialog box.



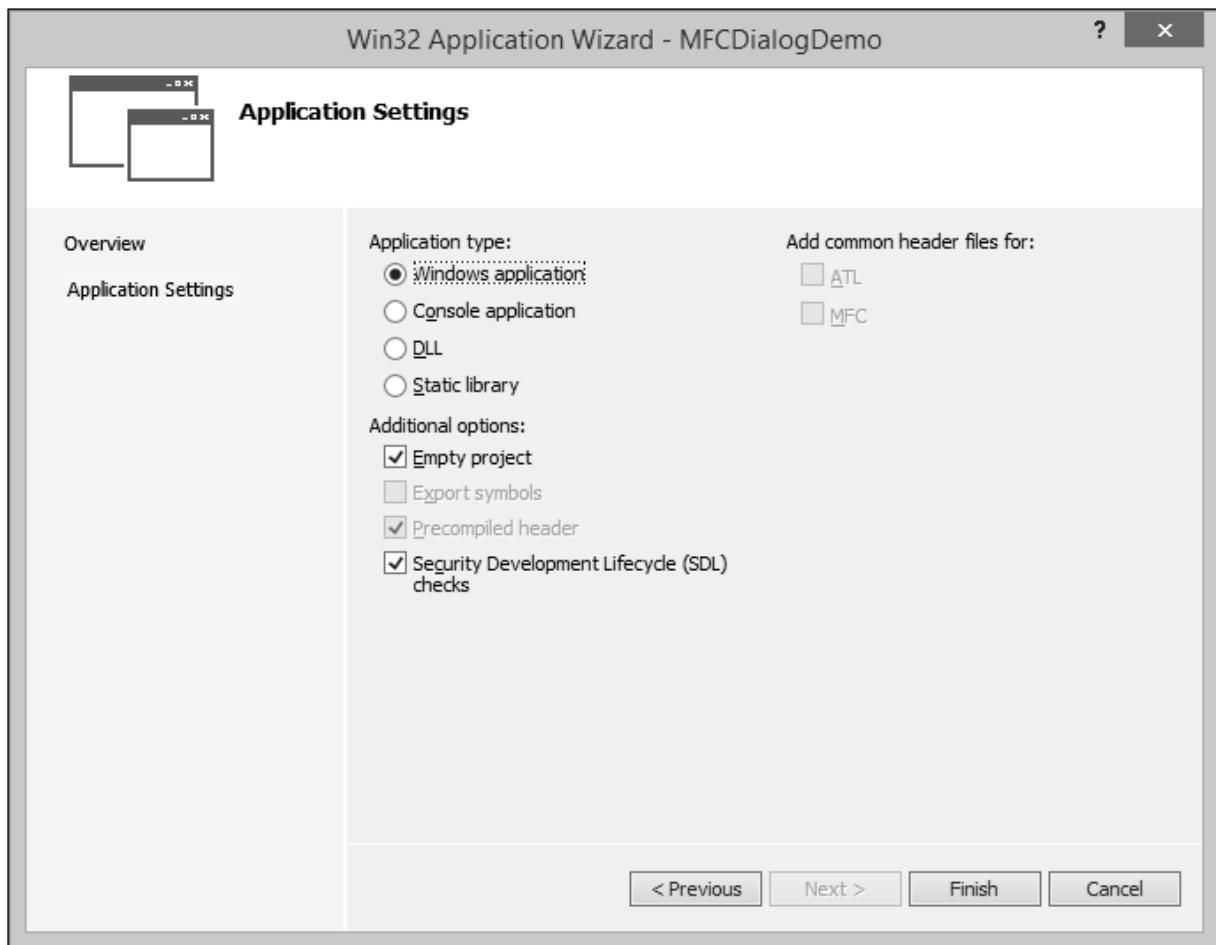
**Step 3:** From the left pane, select Templates -> Visual C++ -> Win32.

**Step 4:** In the middle pane, select Win32 Project.

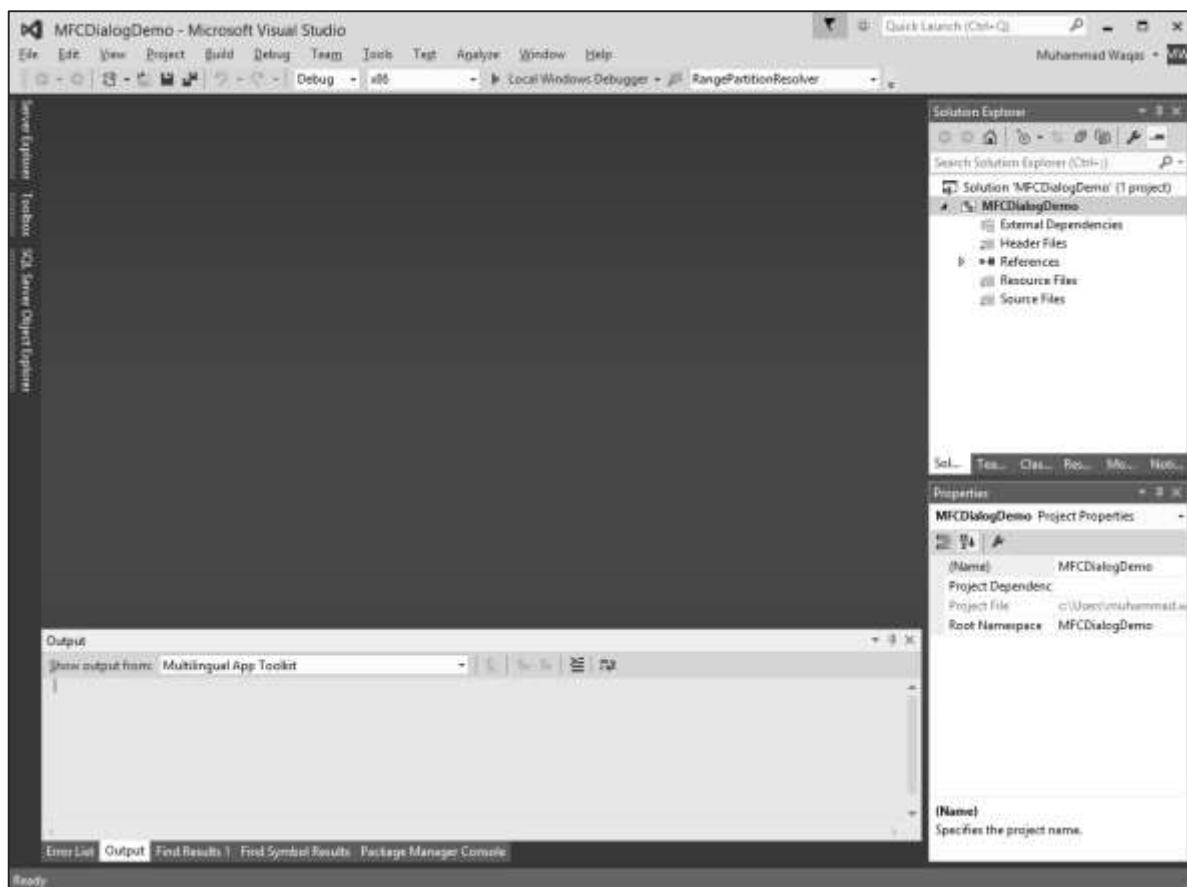
**Step 5:** Enter project name 'MFCDialogDemo' in the Name field and click OK to continue. You will see the following dialog.



**Step 6:** Click Next.

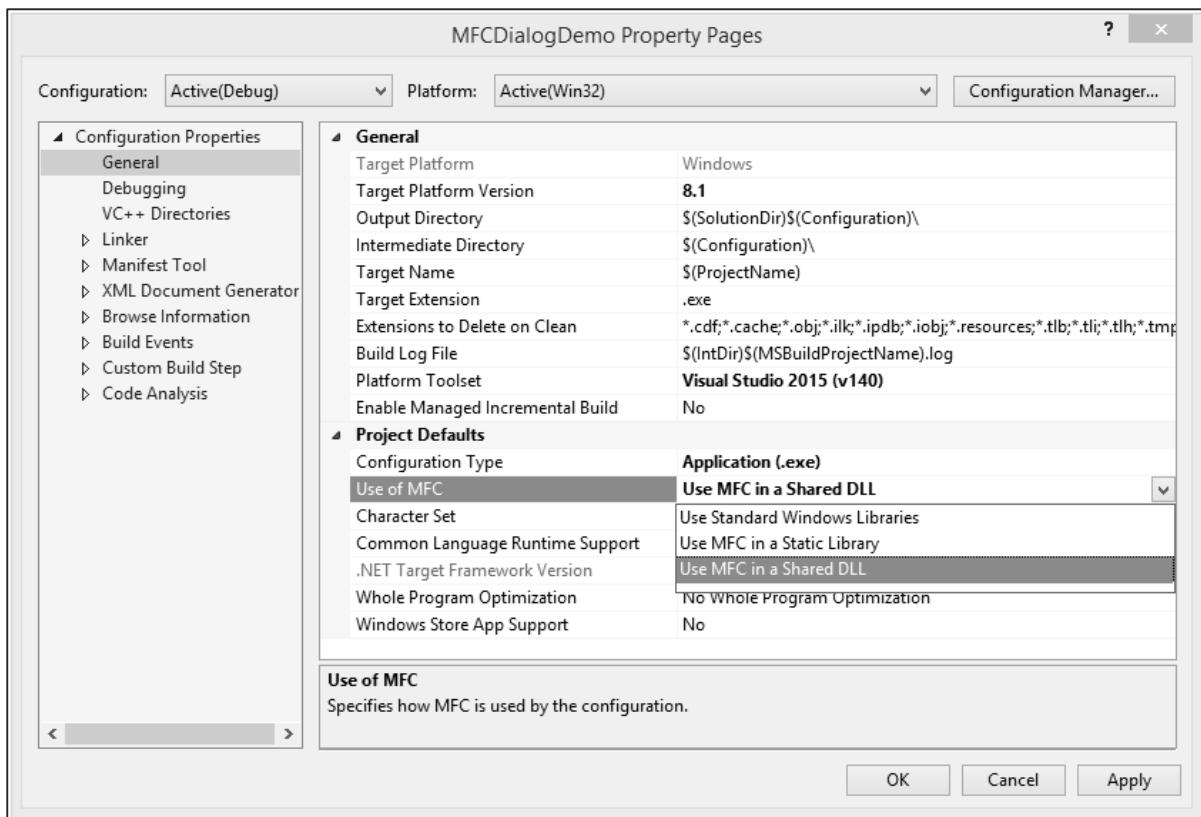


**Step 7:** Select the options shown in the dialog box given above and click Finish.



**Step 8:** An empty project is created.

**Step 9:** To make it a MFC project, right-click on the project and select Properties.



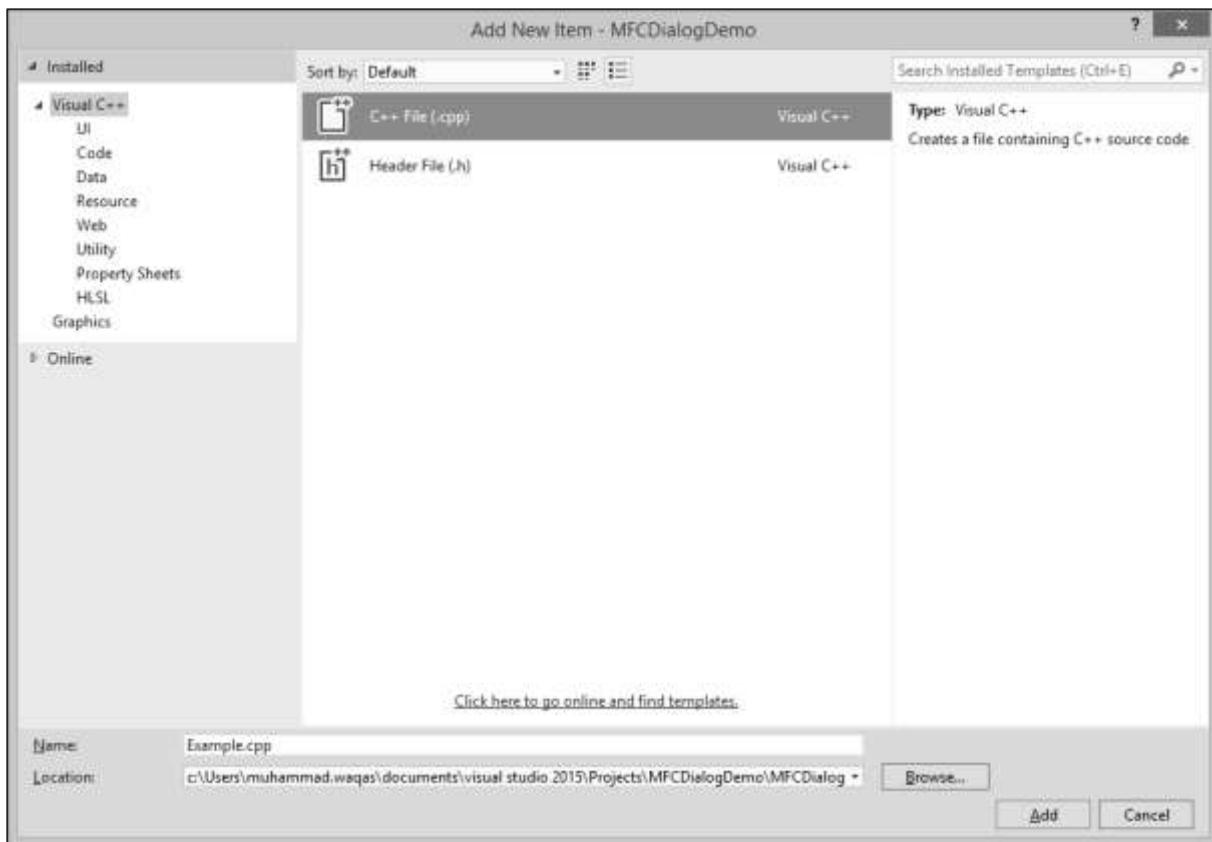
**Step 10:** In the left section, click Configuration Properties -> General.

**Step 11:** Select the Use MFC in Shared DLL option in Project Defaults section and click OK.

**Step 12:** Add a new source file.

**Step 13:** Right-click on your Project and select Add -> New Item.

**Step 14:** In the Templates section, click C++ File (.cpp)



**Step 15:** Set the Name as Example and click Add.

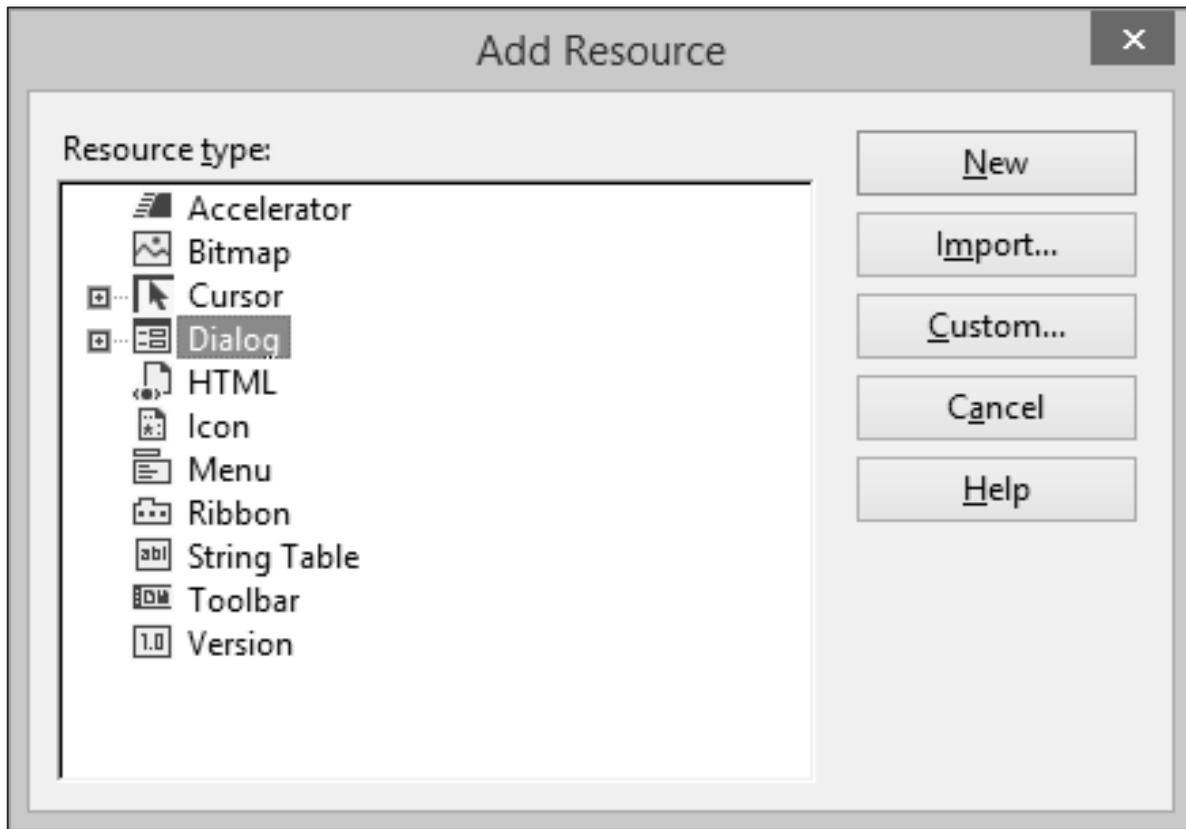
**Step 16:** To create an application, we need to add a class and derive it from the MFC's CWinApp.

```
#include <afxwin.h>

class CExample : public CWinApp
{
public:
    BOOL InitInstance();
};
```

## Dialog Box Creation

**Step 1:** To create a dialog box, right-click on the Resource Files folder in solution explorer and select Add -> Resource.

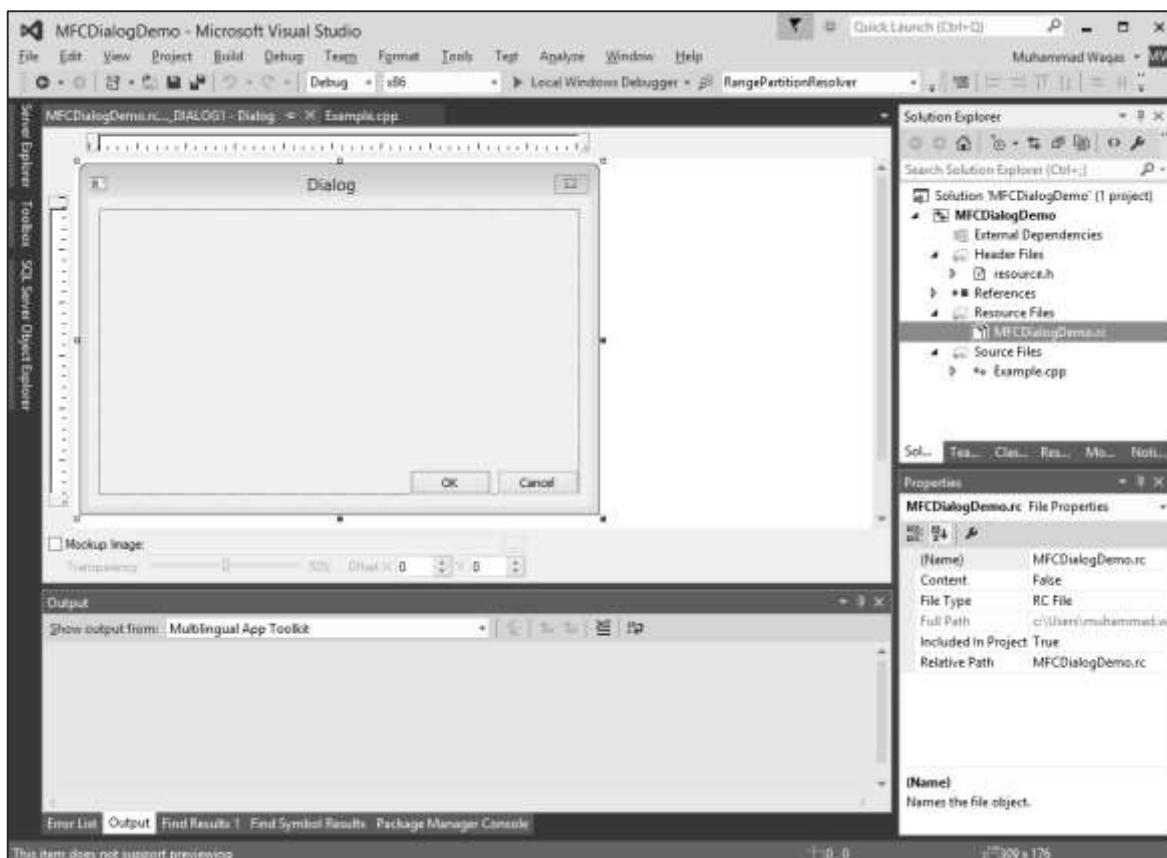


**Step 2:** In the Add Resource dialog box, select Dialog and click New.

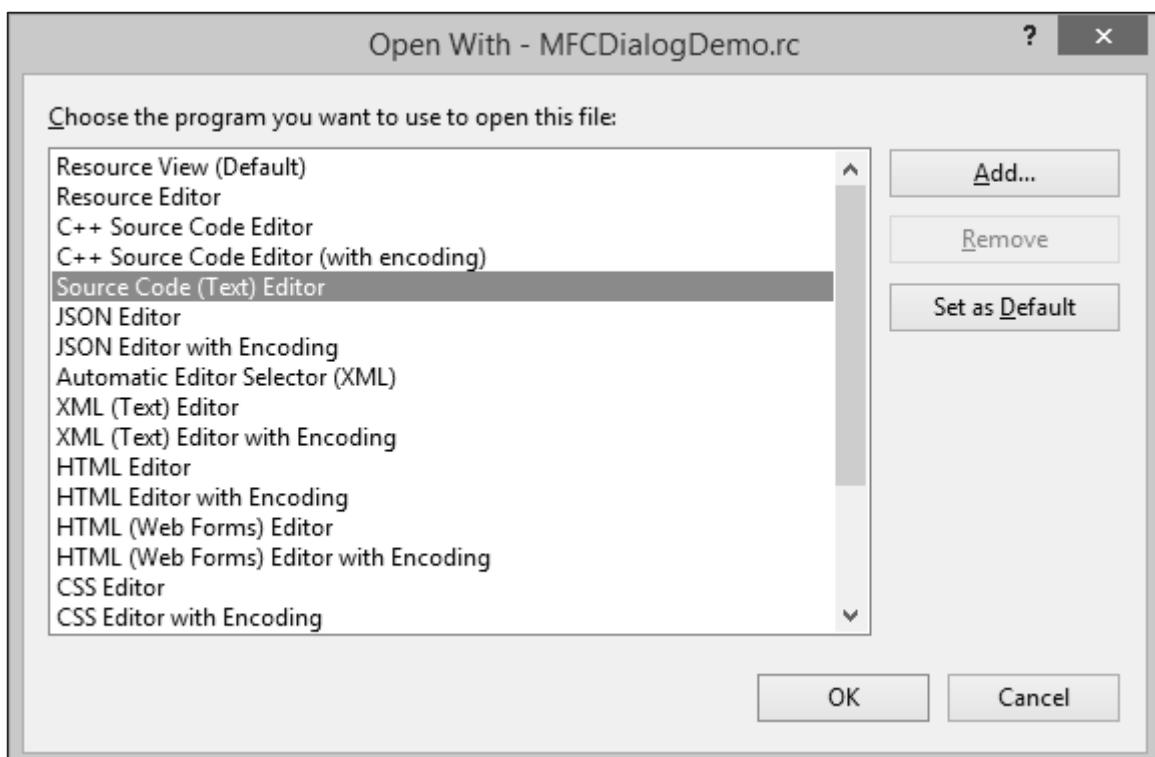
**Step 3:** A dialog box requires some preparation before actually programmatically creating it.

**Step 4:** A dialog box can first be manually created as a text file (in a resource file).

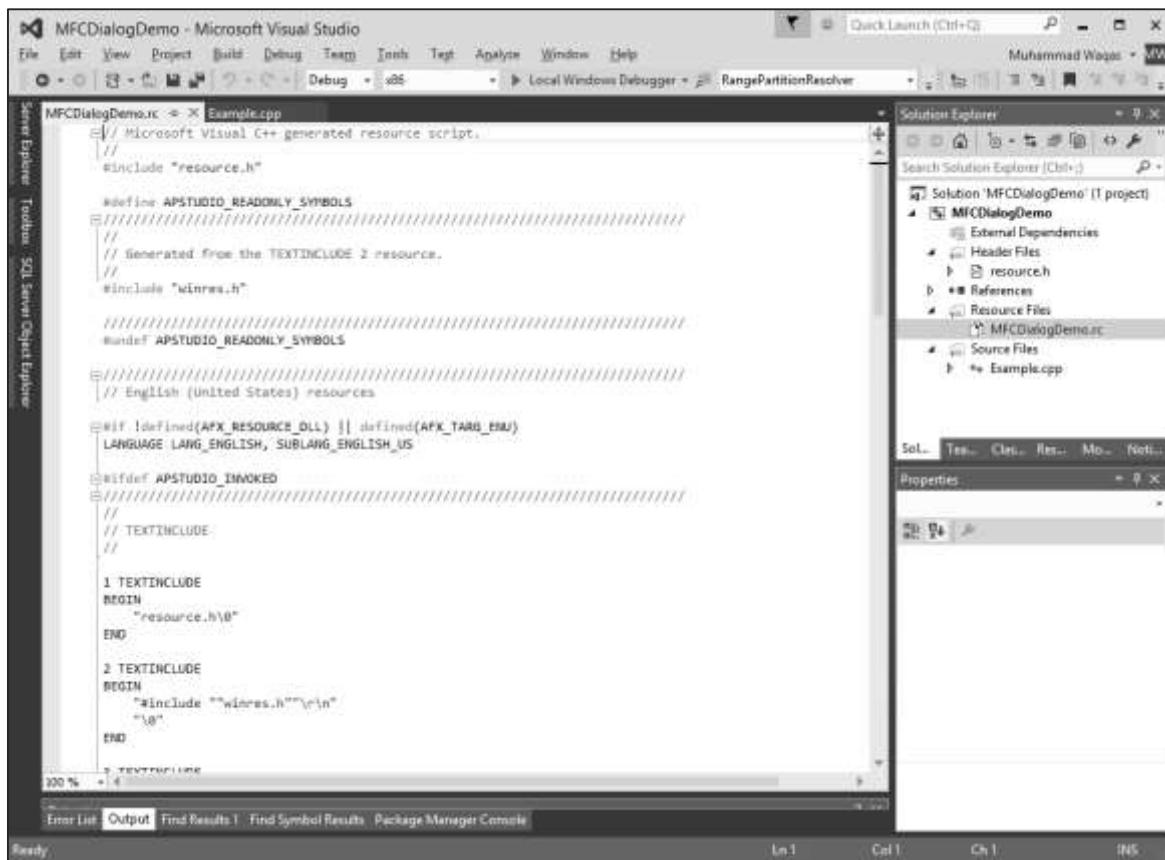
**Step 5:** You can now see the MFCDialogDemo.rc file created under Resource Files.



**Step 6:** The resource file is open in designer. The same can be opened as a text file. Right-click on the resource file and select Open With.

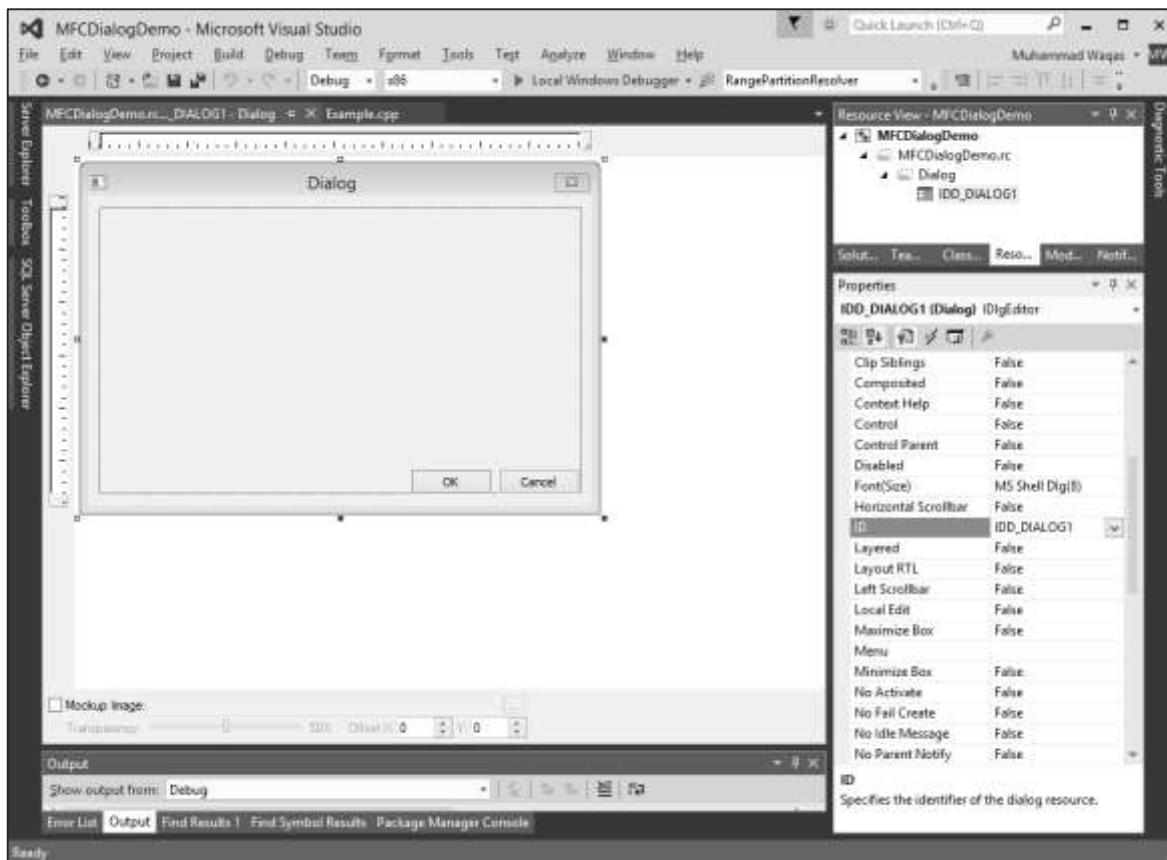


**Step 7:** Select the Source Code (Text) editor and click Add button.



The screenshot shows the Microsoft Visual Studio interface. The title bar reads "MFCDialogDemo - Microsoft Visual Studio". The left pane displays the code for "MFCDialogDemo.rc" (Resource Script File). The code includes defines like APSTUDIO\_READONLY\_SYMBOLS, #include "resource.h", and #include "winres.h". It also contains TEXTINCLUDE sections for resources. The right pane shows the "Solution Explorer" window with the project structure: "MFCDialogDemo" containing "Header Files", "resource.h", "References", "Resource Files" (with "MFCDialogDemo.rc"), and "Source Files" (with "Example.cpp"). Below the Solution Explorer is the "Properties" window. The bottom navigation bar includes tabs for "Error List", "Output", "Find Results", "Find Symbol Results", and "Package Manager Console".

**Step 8:** Go back to the designer and right-click on the dialog and select Properties.



**Step 9:** You need to choose out of the many options.

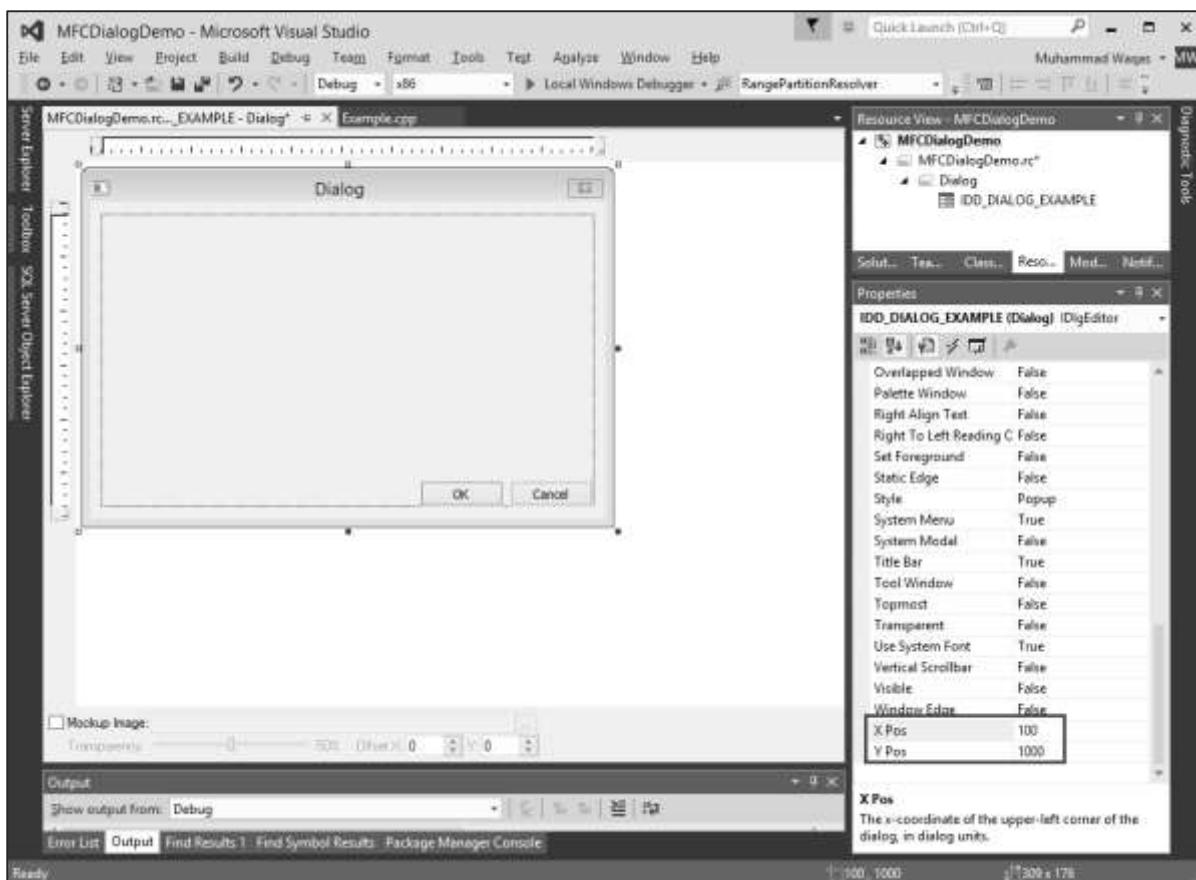
**Step 10:** Like most other controls, a dialog box must be identified. The identifier (ID) of a dialog box usually starts with IDD\_, Let us change the ID to IDD\_EXAMPLE\_DLG.

## Dialog Location

---

A dialog box must be “physically” located on an application. Because a dialog box is usually created as a parent to other controls, its location depends on its relationship to its parent window or to the desktop.

If you look and the Properties window, you see two fields, X Pos and Y Pos.



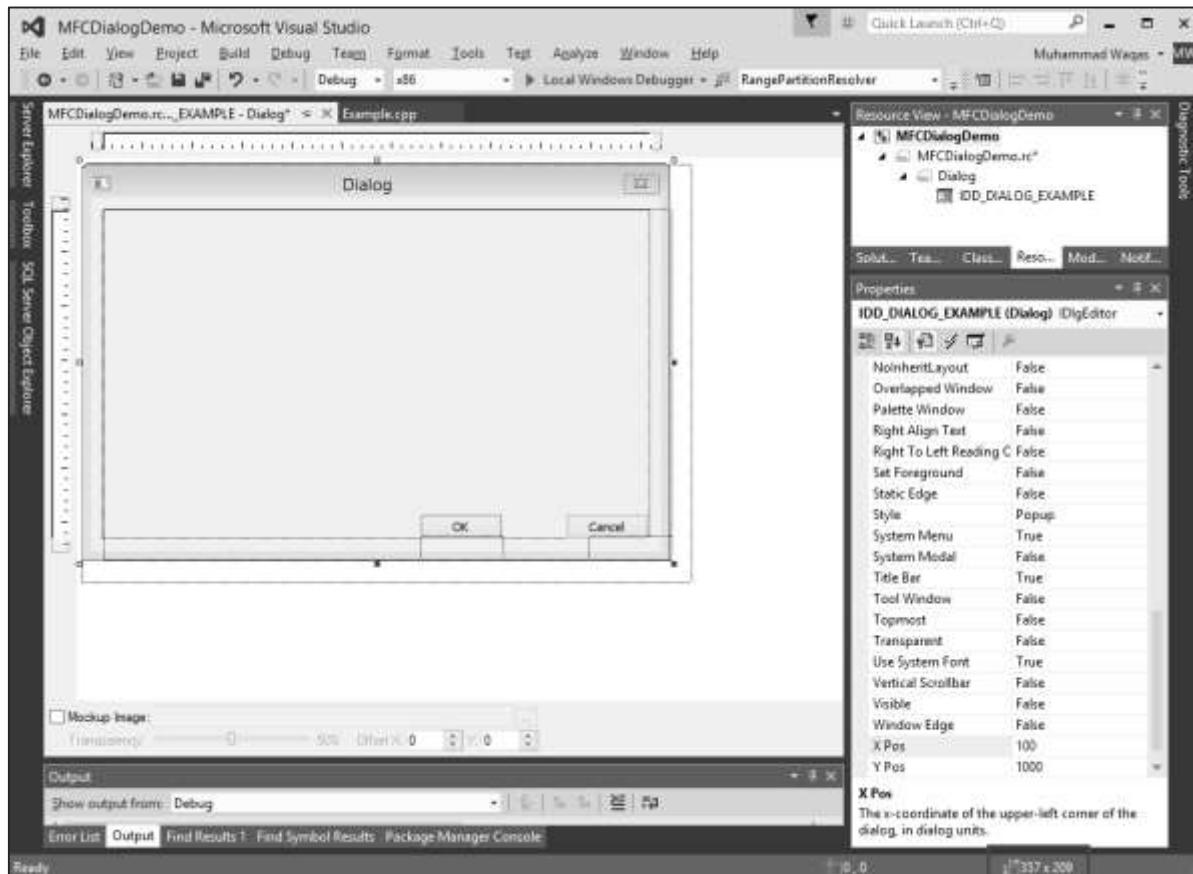
- X is the distance from the left border of the monitor to the left border of the dialog box.
- Y is the distance from the top border of the monitor to the top border of the dialog box.

By default, these fields are set to zero. You can also change as shown above.

If you specify these two dimensions as 0, the left and top borders of the dialog box would be set so the object appears in the center-middle of the screen.

## Dialog Box Dimensions

The dimensions of a dialog box refer to its width and its height. You can resize the width and height with the help of mouse in designer window.



You can see the changes in width and height on the Status Bar.

## Dialog Box Methods

The base class used for displaying dialog boxes on the screen is `CDialog` class. To create a dialog box, we need to derive a class from `CDialog`. The `CDialog` class itself provides three constructors which are as follows:

```
CDialog();
CDialog(UINT nIDTemplate, CWnd* pParentWnd = NULL);
CDialog(LPCTSTR lpszTemplateName, CWnd* pParentWnd = NULL);
```

Let us create another class CExampleDlg and derive it from CDialog. We will implement its default constructor/destructor as shown in the following code.

```
class CExampleDlg : public CDialog
{
public:
    enum { IDD = IDD_EXAMPLE_DLG };

    CExampleDlg();
    ~CExampleDlg();
};

CExampleDlg::CExampleDlg():CDialog(CExampleDlg::IDD)
{
}

CExampleDlg::~CExampleDlg()
{
}
```

We need to instantiate this dialog on CExample::InitInstance() method as shown in the following code.

```
BOOL CExample::InitInstance()
{
    CExampleDlg myDlg;
    m_pMainWnd = &myDlg;

    return TRUE;
}
```

## Modal Dialog Boxes

There are two types of dialog boxes — **modeless** and **modal**. Modal and modeless dialog boxes differ by the process used to create and display them.

## Modeless Dialog Box

- For a modeless dialog box, you must provide your own public constructor in your dialog class.
- To create a modeless dialog box, call your public constructor and then call the dialog object's Create member function to load the dialog resource.
- You can call Create either during or after the constructor call. If the dialog resource has the property WS\_VISIBLE, the dialog box appears immediately.
- If not, you must call its ShowWindow member function.

## Modal Dialog

- To create a modal dialog box, call either of the two public constructors declared in CDialog.
- Next, call the dialog object's **DoModal** member function to display the dialog box and manage interaction with it until the user chooses OK or Cancel.
- This management by DoModal is what makes the dialog box modal. For modal dialog boxes, DoModal loads the dialog resource.

**Step 1:** To display the dialog box as modal, in the CExample::InitInstance() event call the DoModal() method using your dialog variable:

```
BOOL CExample::InitInstance()
{
    CExampleDlg myDlg;
    m_pMainWnd = &myDlg;
    myDlg.DoModal();
    return TRUE;
}
```

**Step 2:** Here is the complete implementation of Example.cpp file.

```
#include <afxwin.h>
#include "resource.h"

class CExample : public CWinApp
{
public:
    BOOL InitInstance();
```

```
};

class CExampleDlg : public CDialog
{
public:
    enum { IDD = IDD_EXAMPLE_DLG };

    CExampleDlg();
    ~CExampleDlg();
};

CExampleDlg::CExampleDlg():CDialog(CExampleDlg::IDD)
{
}

CExampleDlg::~CExampleDlg()
{
}

BOOL CExample::InitInstance()
{
    CExampleDlg myDlg;
    m_pMainWnd = &myDlg;
    myDlg.DoModal();
    return TRUE;
}
CExample MyApp;
```

**Step 3:** When the above code is compiled and executed, you will see the following dialog box.

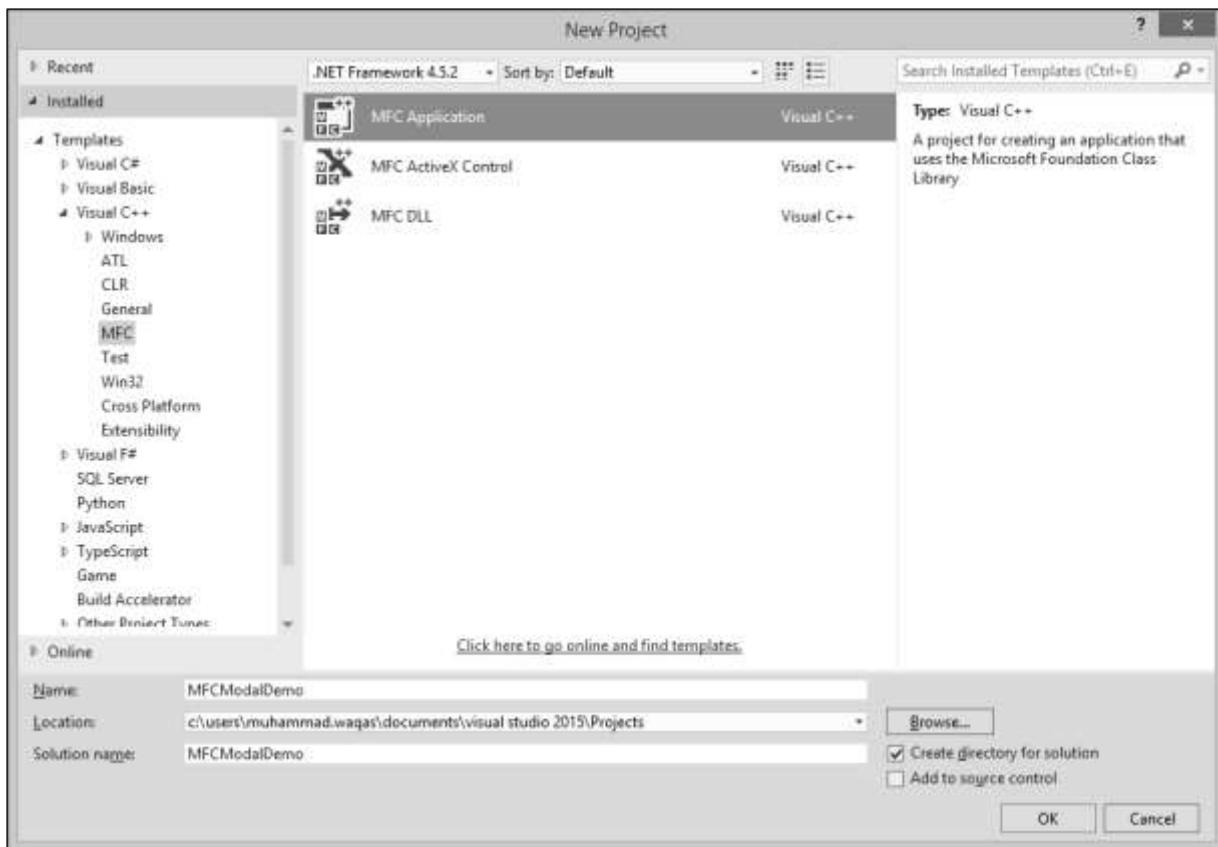


## Dialog-Based Applications

---

Microsoft Visual Studio provides an easier way to create an application that is mainly based on a dialog box. Here are the steps to create a dialog base project using project templates available in Visual Studio:

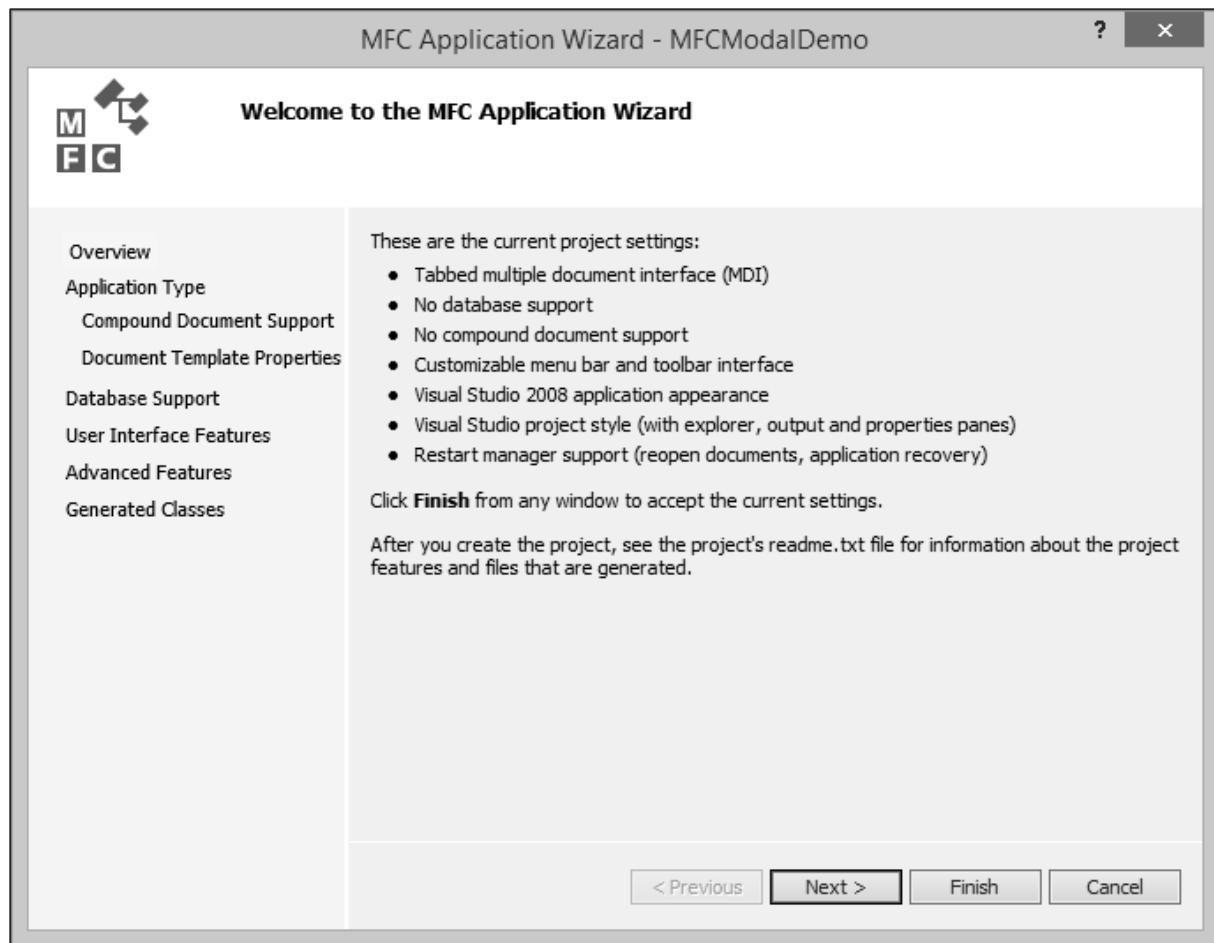
**Step 1:** Open the Visual studio and click on the File -> New -> Project menu option. You can see the New Project dialog box.



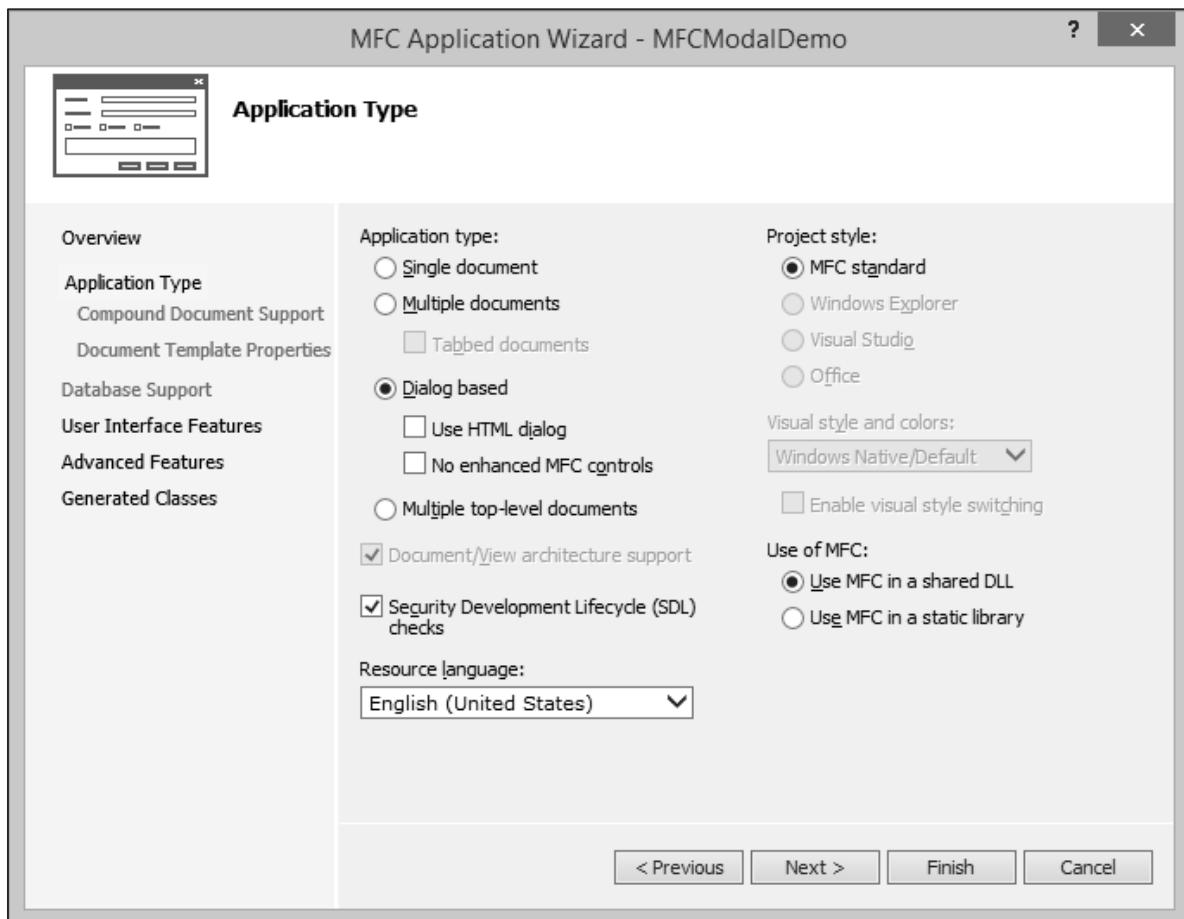
**Step 2:** From the left pane, select Templates -> Visual C++ -> MFC.

**Step 3:** In the middle pane, select MFC Application.

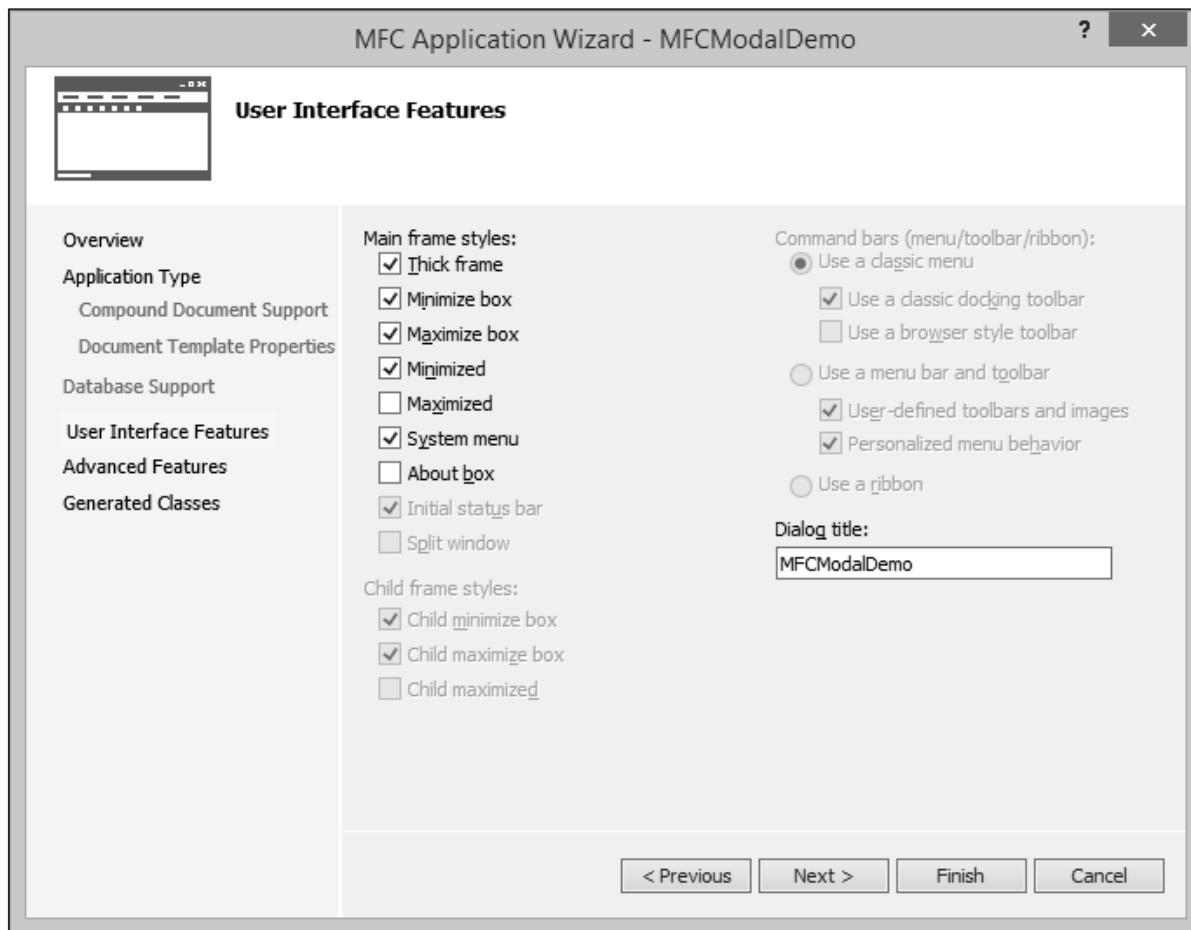
**Step 4:** Enter project name 'MFCModalDemo' in the Name field and click OK to continue. You will see the following dialog box.



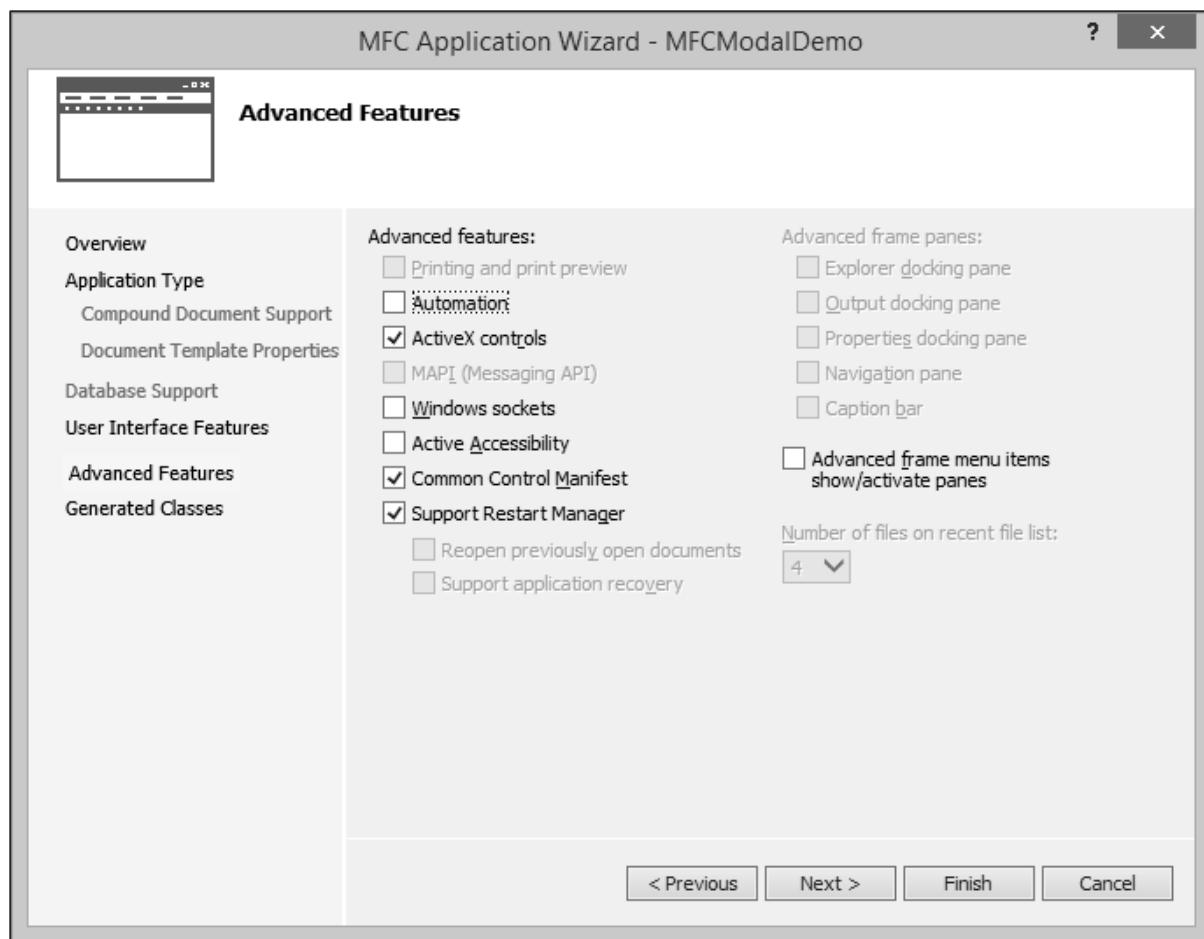
**Step 6:** Click Next.



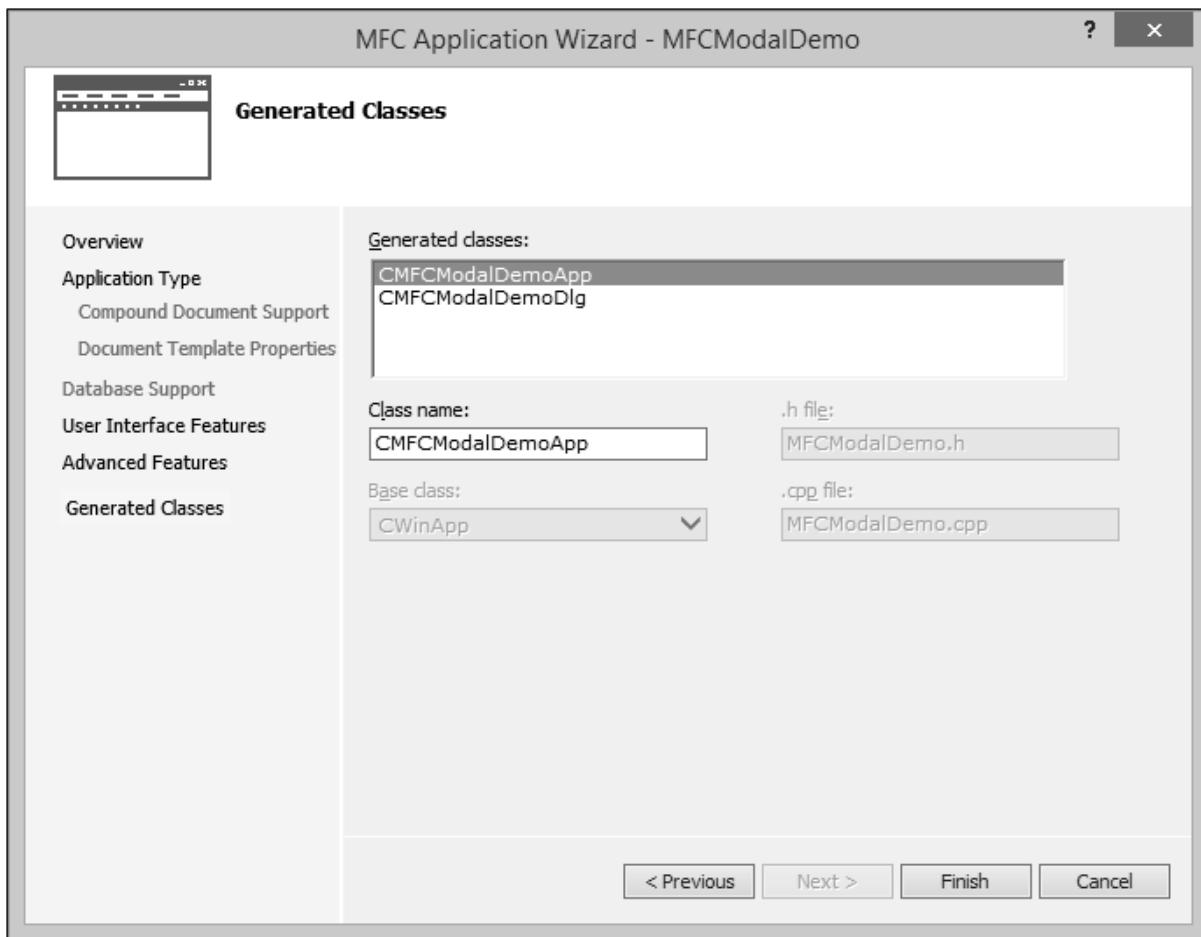
**Step 7:** Select the options shown in the above dialog box and click Next.



**Step 8:** Check all the options that you choose to have on your dialog box like Maximize and Minimize Boxes and click Next.

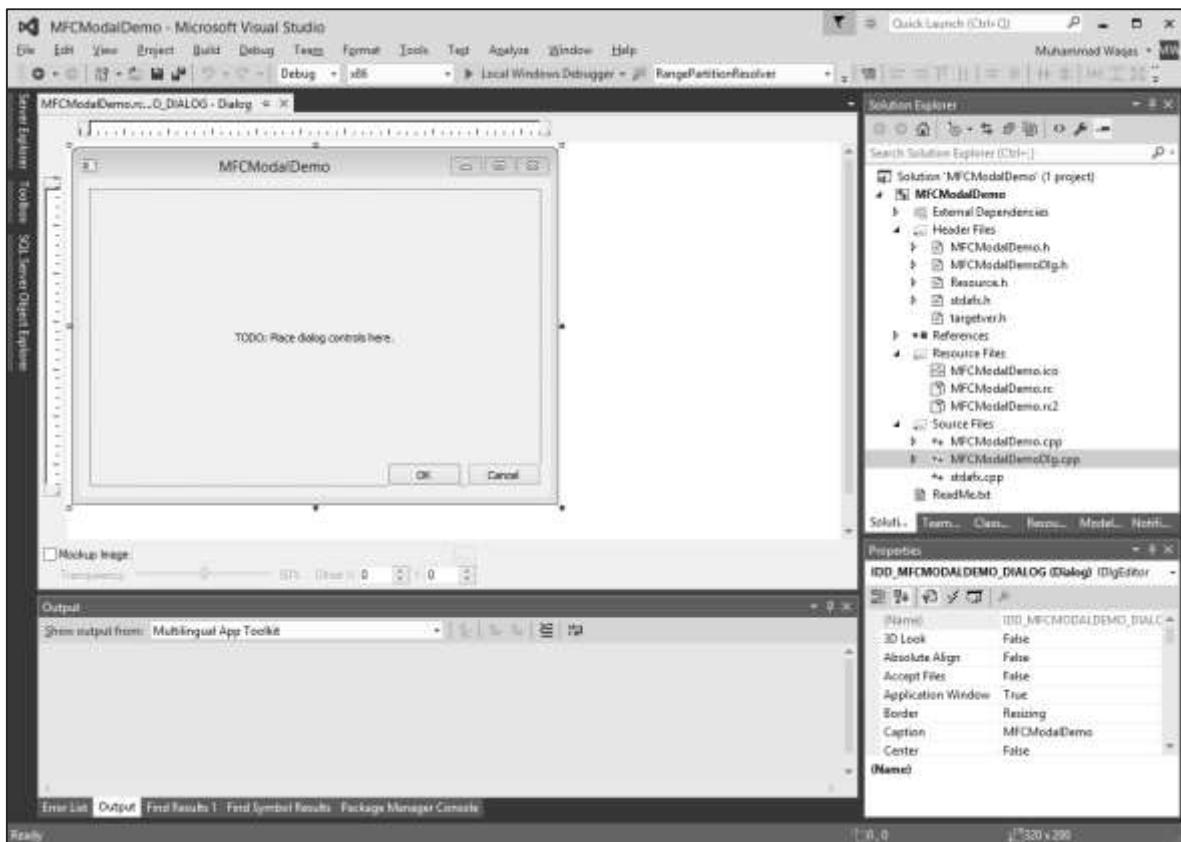


**Step 9:** Click Next.

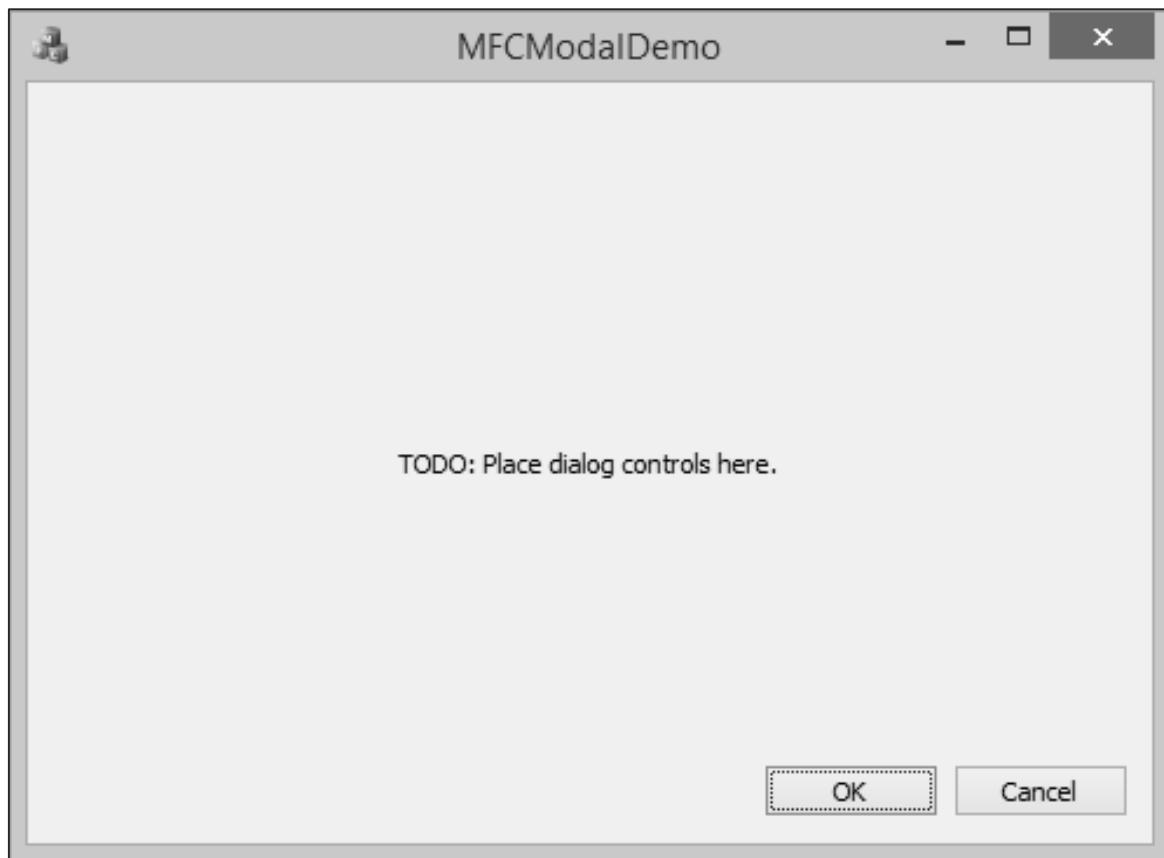


**Step 10:** It will generate these two classes. You can change the name of the classes and click Finish.

**Step 11:** You can now see that the MFC wizard creates this Dialog Box and the project files by default.



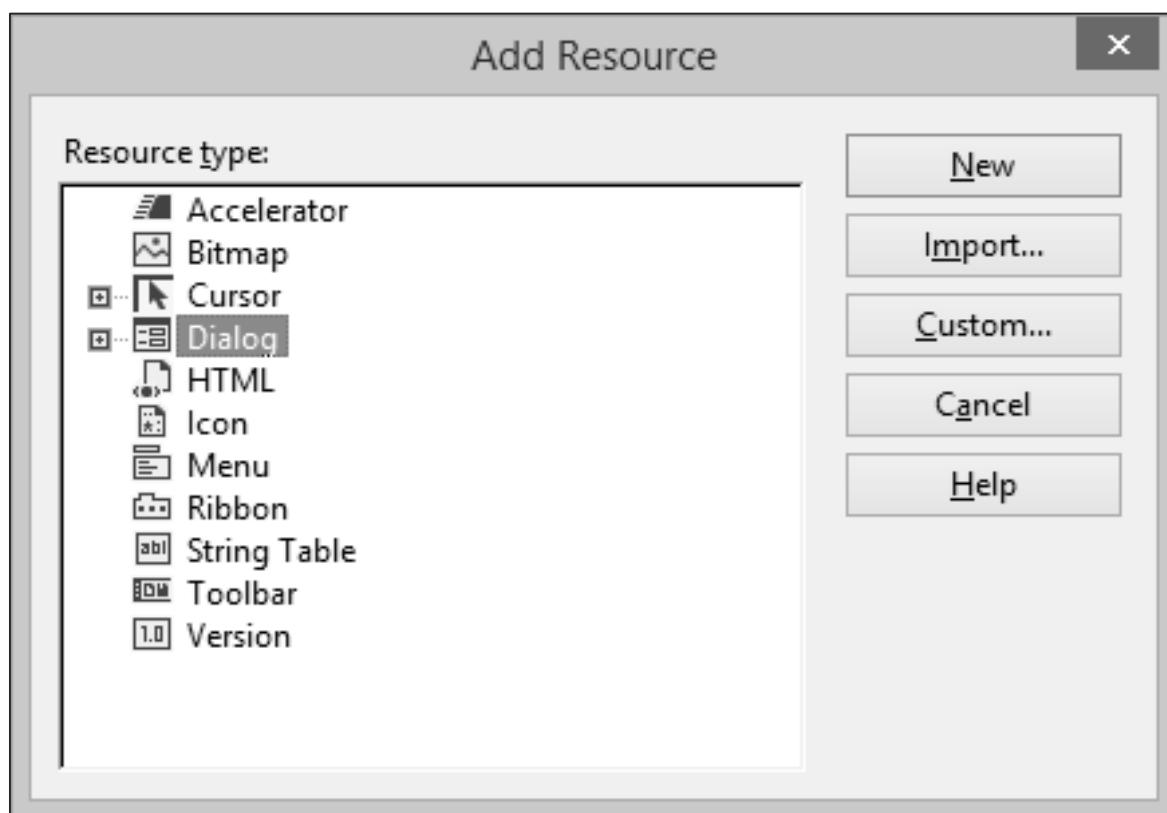
**Step 12:** When you run this application, you will see the following output.



## 7. MFC - Windows Resources

A **resource** is a text file that allows the compiler to manage objects such as pictures, sounds, mouse cursors, dialog boxes, etc. Microsoft Visual Studio makes creating a resource file particularly easy by providing the necessary tools in the same environment used to program. This means, you usually do not have to use an external application to create or configure a resource file. Following are some important features related to resources.

- Resources are interface elements that provide information to the user.
- Bitmaps, icons, toolbars, and cursors are all resources.
- Some resources can be manipulated to perform an action such as selecting from a menu or entering data in dialog box.
- An application can use various resources that behave independently of each other, these resources are grouped into a text file that has the \*.rc extension.
- Most resources are created by selecting the desired one from the Add Resource dialog box.



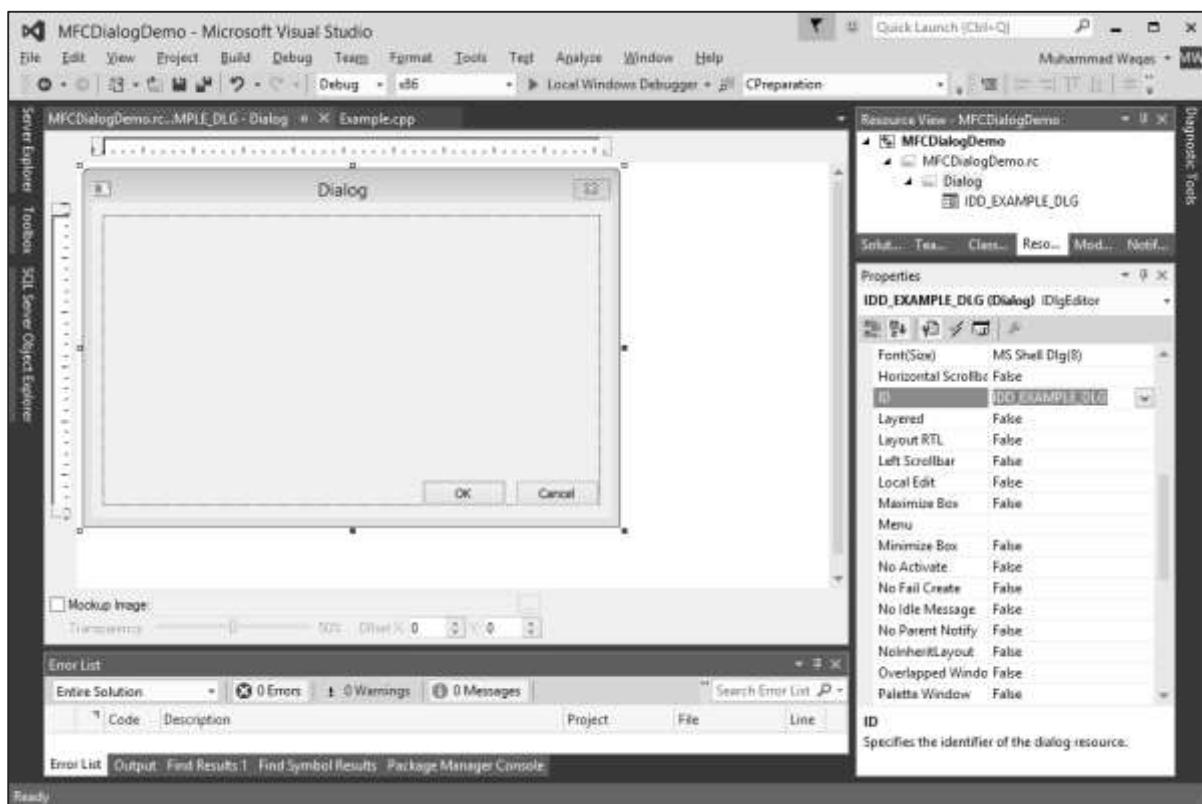
- The Add Resource dialog box provides an extensive list of resources which can be used as per requirements, but if you need something which is not available then you can add it manually to the \*.rc file before executing the program.

## Identifiers

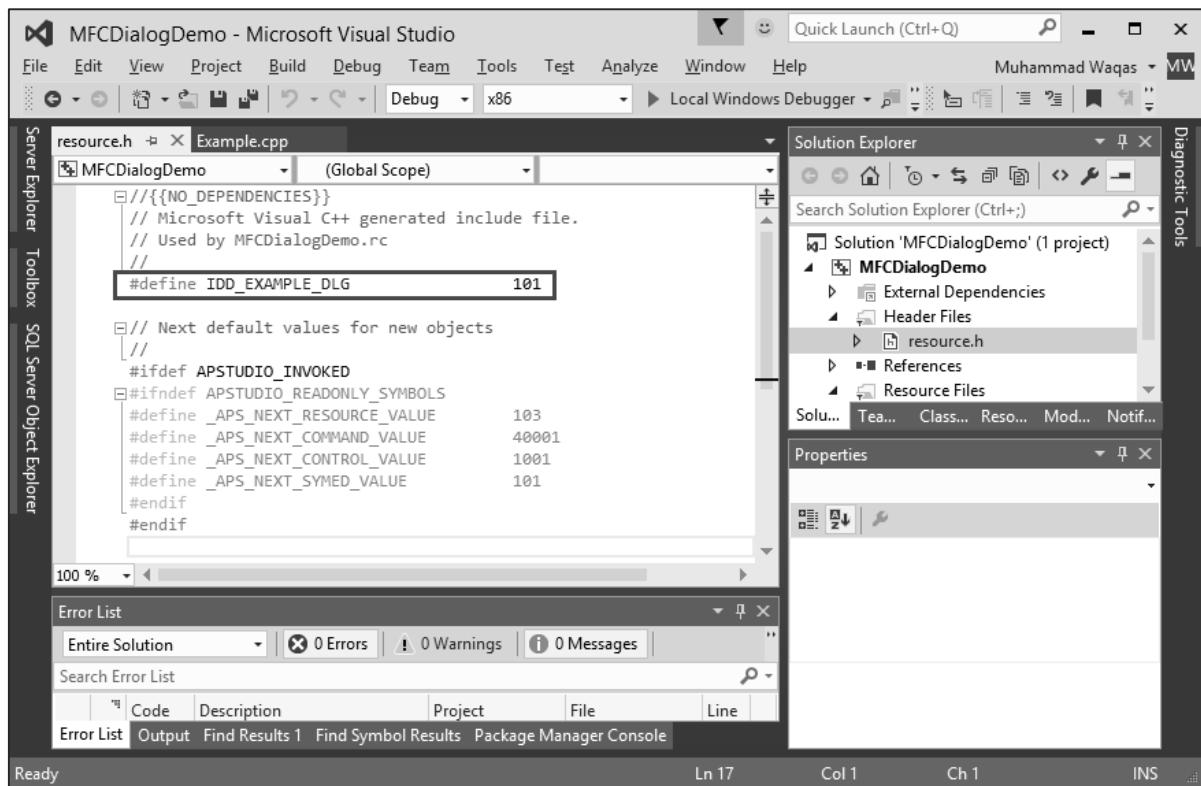
An **identifier** is a symbol which is a constant integer whose name usually starts with ID. It consists of two parts — a text string (symbol name) mapped to an integer value (symbol value).

- Symbols provide a descriptive way of referring to resources and user-interface objects, both in your source code and while you're working with them in the resource editors.
- When you create a new resource or resource object, the **resource editors** provide a default name for the resource, for example, IDC\_DIALOG1, and assign a value to it.
- The name-plus-value definition is stored in the Resource.h file.

**Step 1:** Let us look into our **CMFCDialogDemo** example from the last chapter in which we have created a dialog box and its ID is **IDD\_EXAMPLE\_DLG**.



**Step 2:** Go to the Solution Explorer, you will see the resource.h file under Header Files. Continue by opening this file in editor and you will see the dialog box identifier and its integer value as well.



## Icons

An **icon** is a small picture used on a window which represents an application. It is used in two main scenarios.

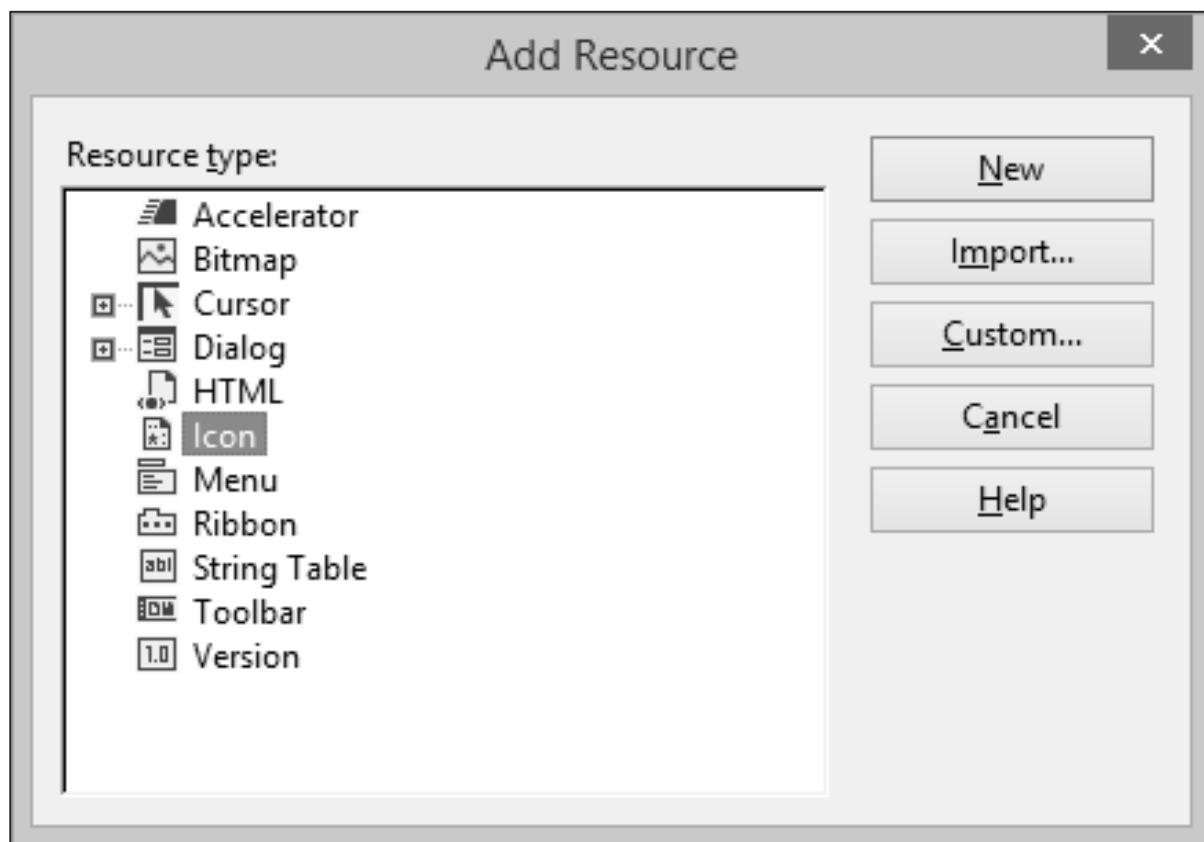
- On a Window's frame, it is displayed on the left side of the Window name on the title bar.
- In Windows Explorer, on the Desktop, in My Computer, or in the Control Panel window.

If you look at our MFCModalDemo example, you will see that Visual studio was using a default icon for the title bar as shown in the following snapshot.

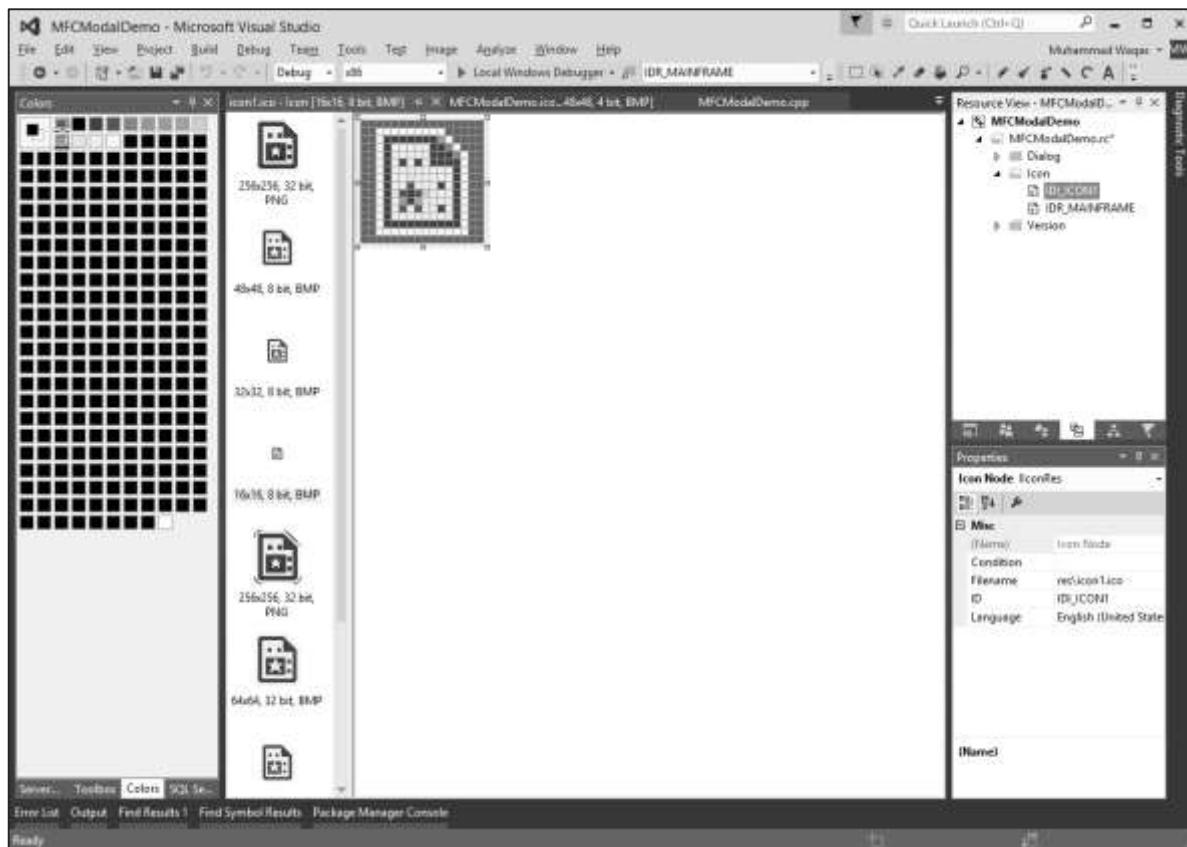


You can create your own icon by following the steps given below:

**Step 1:** Right-click on your project and select Add -> Resources, you will see the Add Resources dialog box.



**Step 2:** Select Icon and click New button and you will see the following icon.



**Step 3:** In Solution Explorer, go to Resource View and expand MFCModalDemo > Icon. You will see two icons. The IDR\_MAINFRAME is the default one and IDI\_ICON1 is the newly created icon.

**Step 4:** Right-click on the newly Created icon and select Properties.

**Step 5:** IDI\_ICON1 is the ID of this icon, now Let us change this ID to IDR\_MYICON.

**Step 6:** You can now change this icon in the designer as per your requirements. We will use the same icon.

**Step 7:** Save this icon.

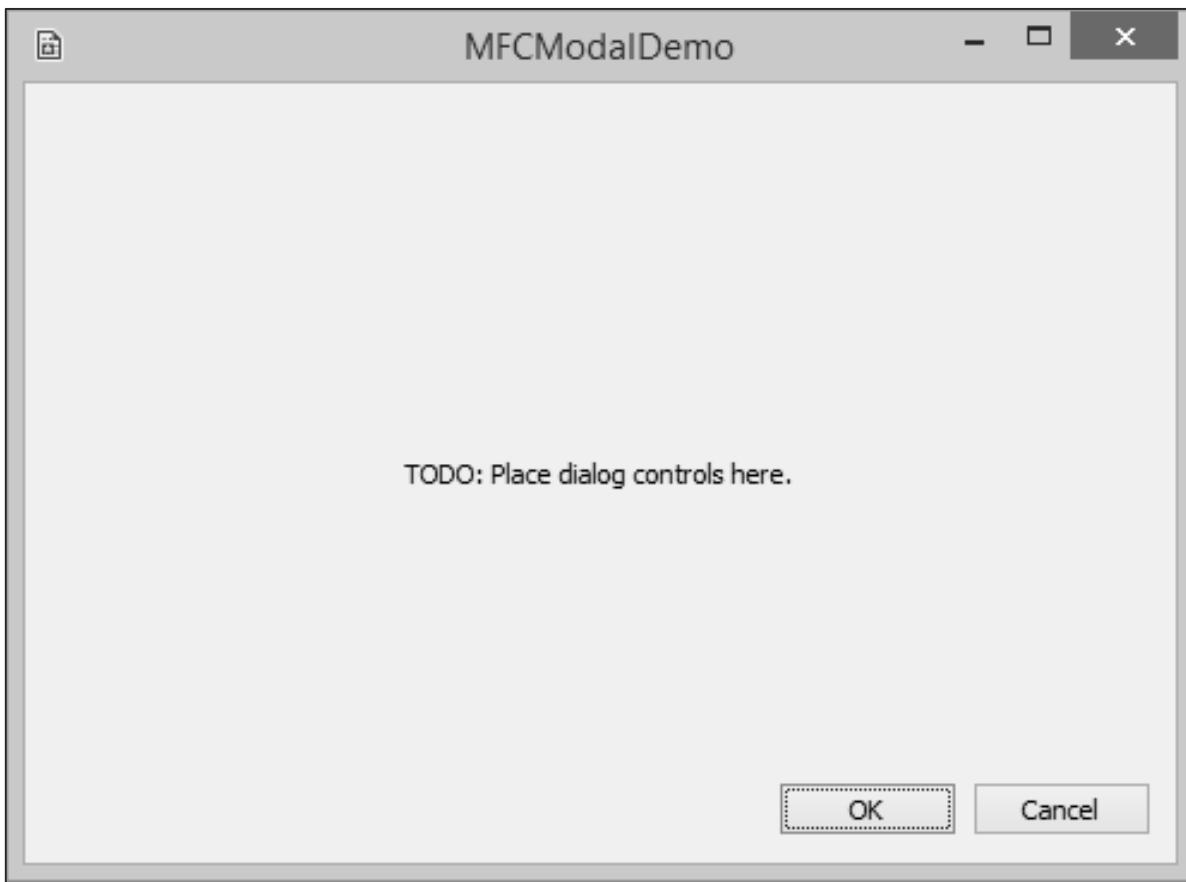
**Step 8:** Go to the CMFCModalDemoDlg constructor in CMFCModalDemoDlg.cpp file which will look like the following code.

```
CMFCModalDemoDlg::CMFCModalDemoDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(IDD_MFCMODALDEMO_DIALOG, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

**Step 9:** You can now see that the default icon is loaded in the constructor. Let us change it to IDR\_ MYICON as shown in the following code.

```
CMFCModalDemoDlg::CMFCModalDemoDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(IDD_MFCMODALDEMO_DIALOG, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_ MYICON);
}
```

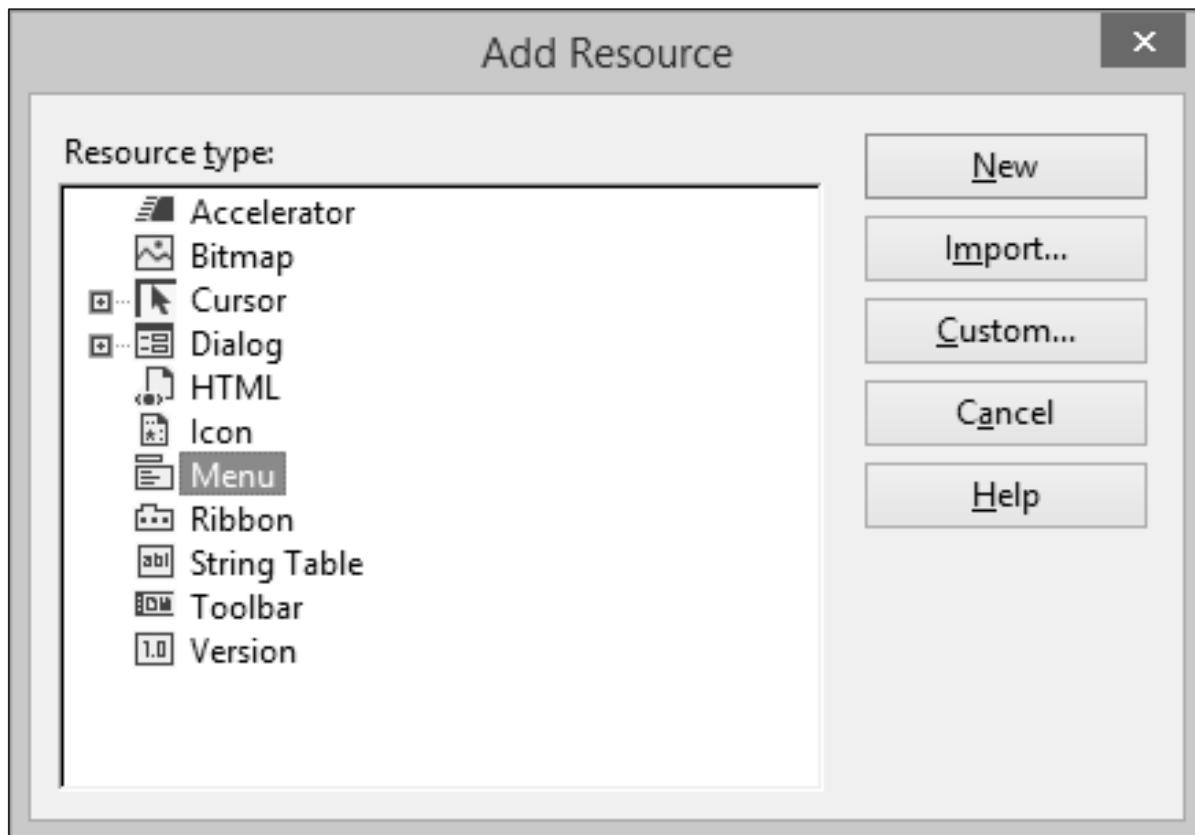
**Step 10:** When the above code is compiled and executed, you will see the new icon is displayed on the dialog box.



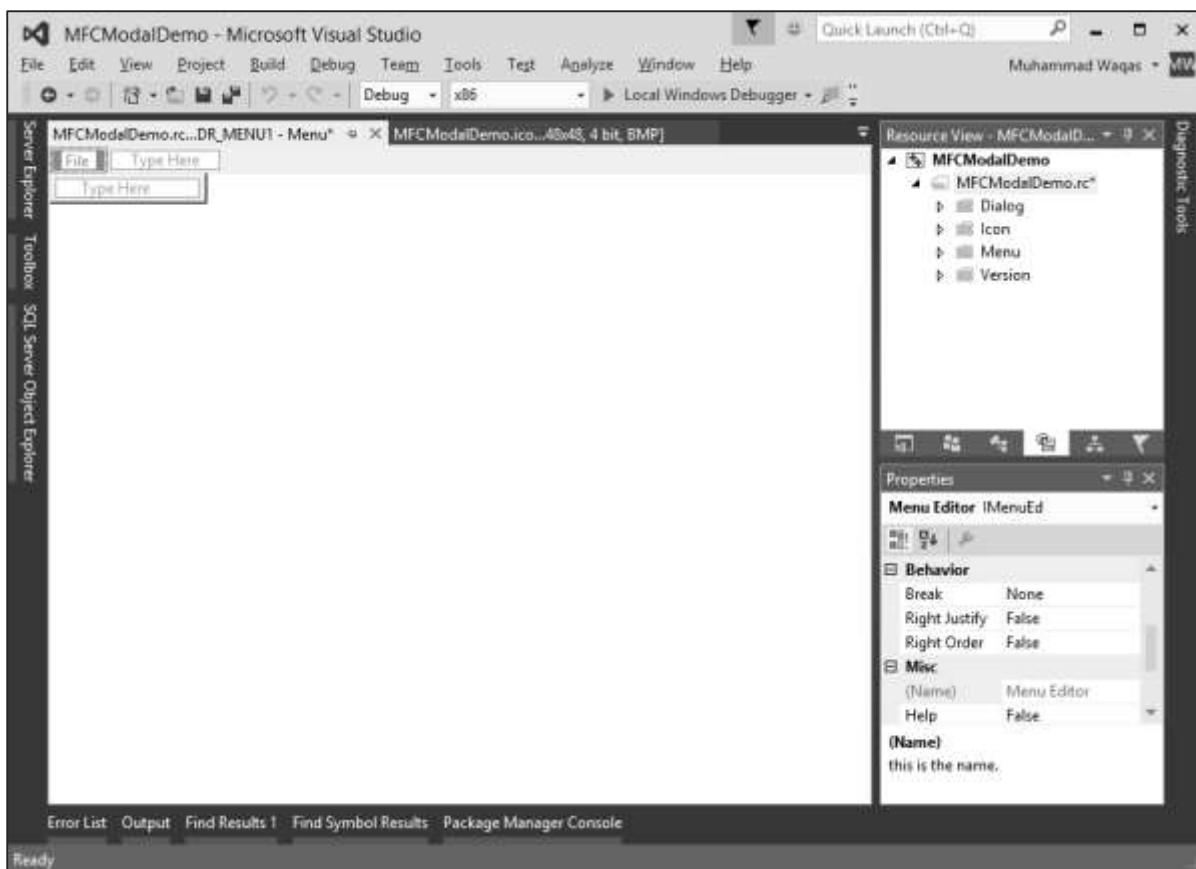
## Menus

**Menus** allow you to arrange commands in a logical and easy-to-find fashion. With the Menu editor, you can create and edit menus by working directly with a menu bar that closely resembles the one in your finished application. To create a menu, follow the steps given below:

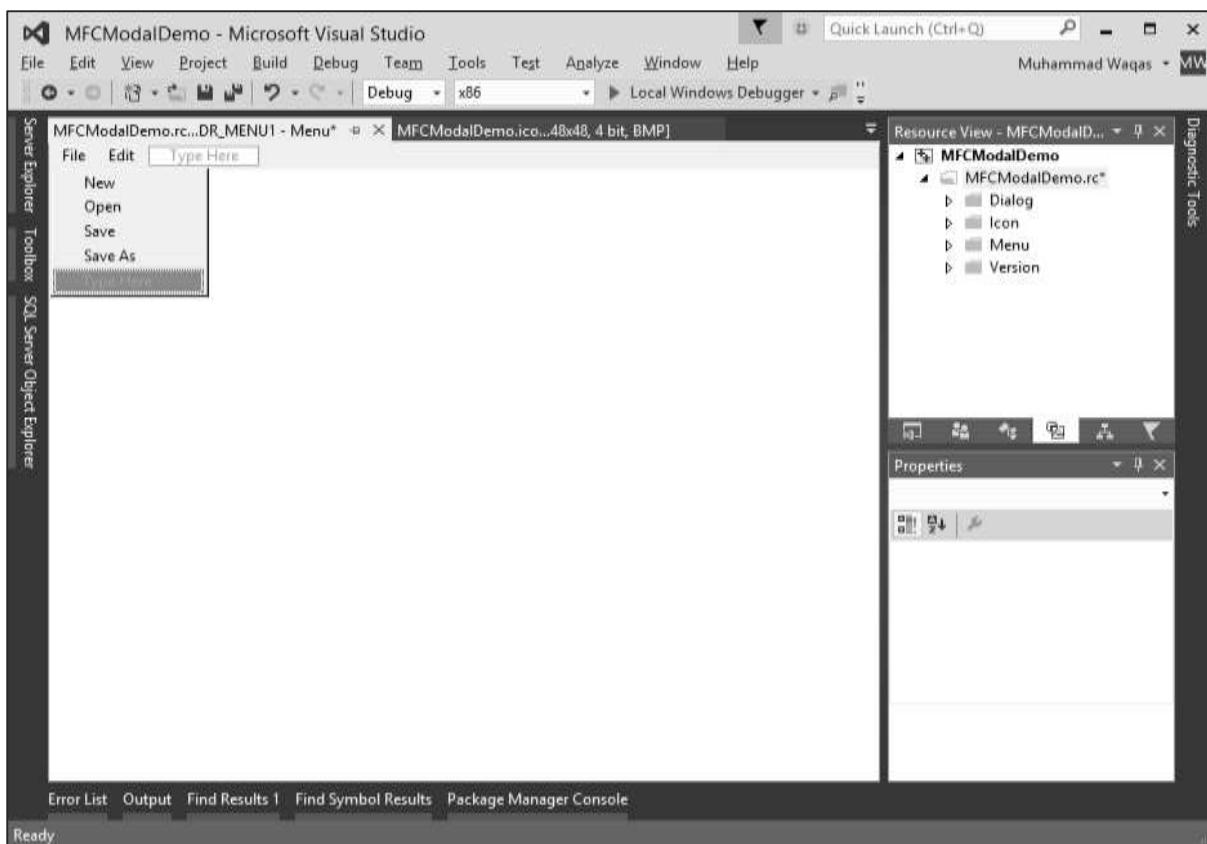
**Step 1:** Right-click on your project and select Add -> Resources. You will see the Add Resources dialog box.



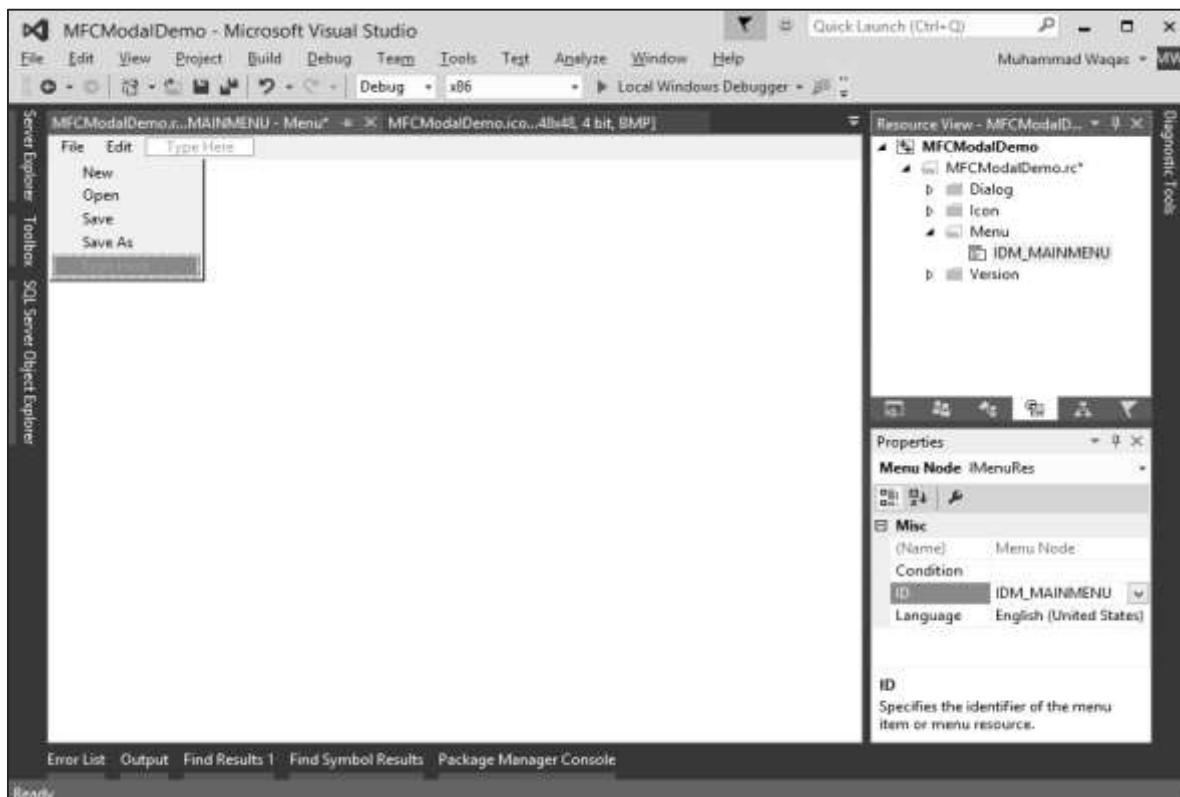
**Step 2:** Select Menu and click New. You will see the rectangle that contains "Type Here" on the menu bar.



**Step 3:** Write some menu options like File, Edit, etc. as shown in the following snapshot.

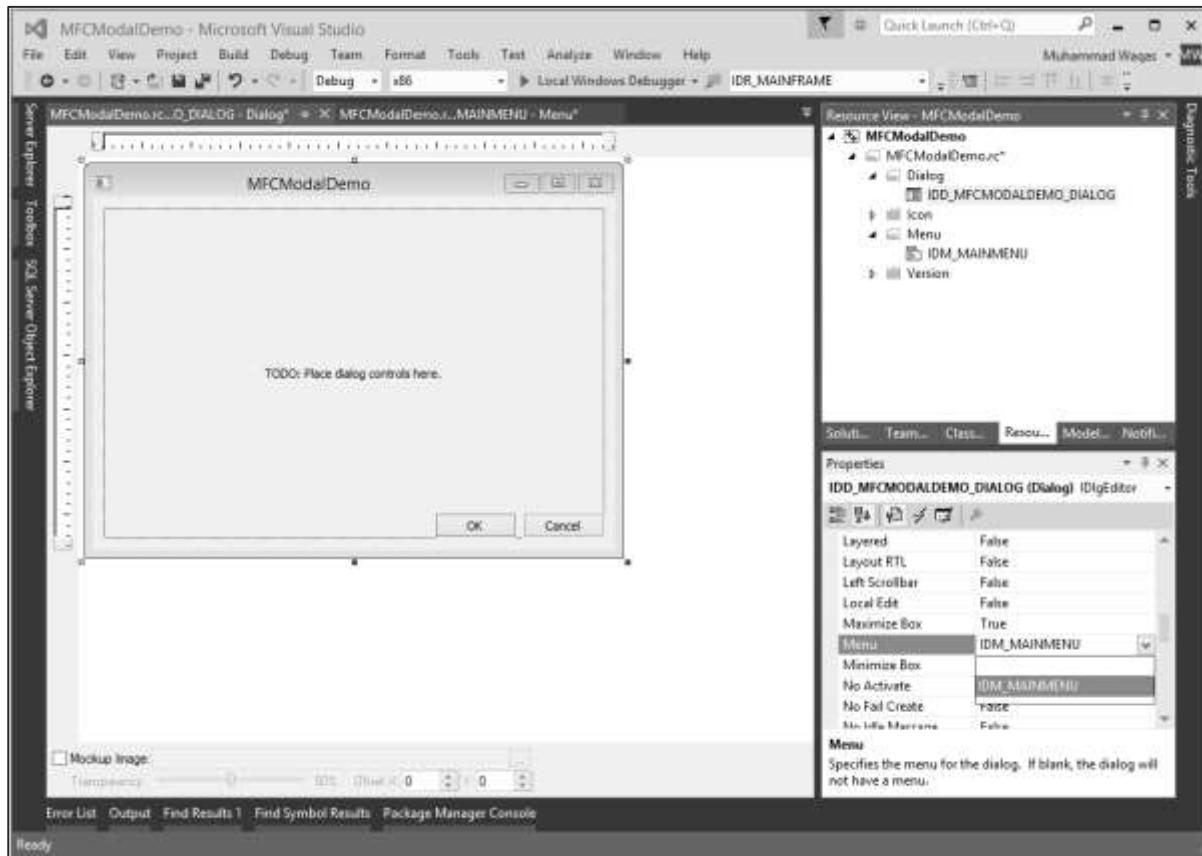


**Step 4:** If you expand the Menu folder in Resource View, you will see the Menu identifier IDR\_MENU1. Right-click on this identifier and change it to IDM\_MAINMENU.



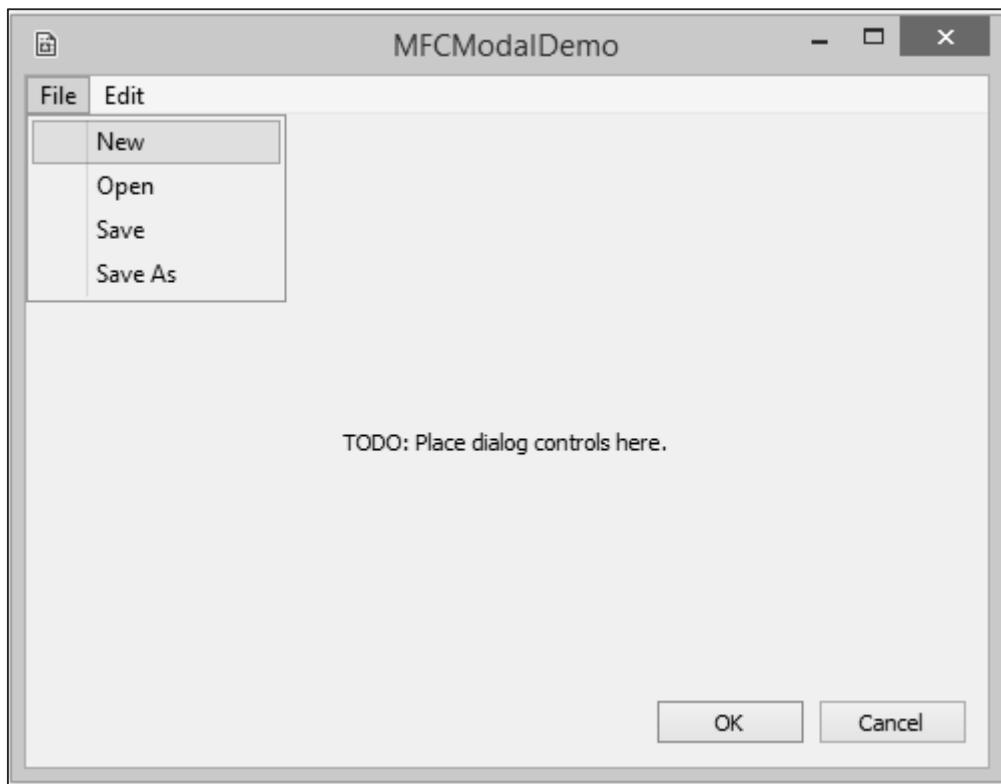
**Step 5:** Save all the changes.

**Step 6:** We need to attach this menu to our dialog box. Expand your Dialog folder in Solution Explorer and double click on the dialog box identifier.



**Step 7:** You will see the menu field in the Properties. Select the Menu identifier from the dropdown as shown above.

**Step 8:** Run this application and you will see the following dialog box which also contains menu options.



## Toolbars

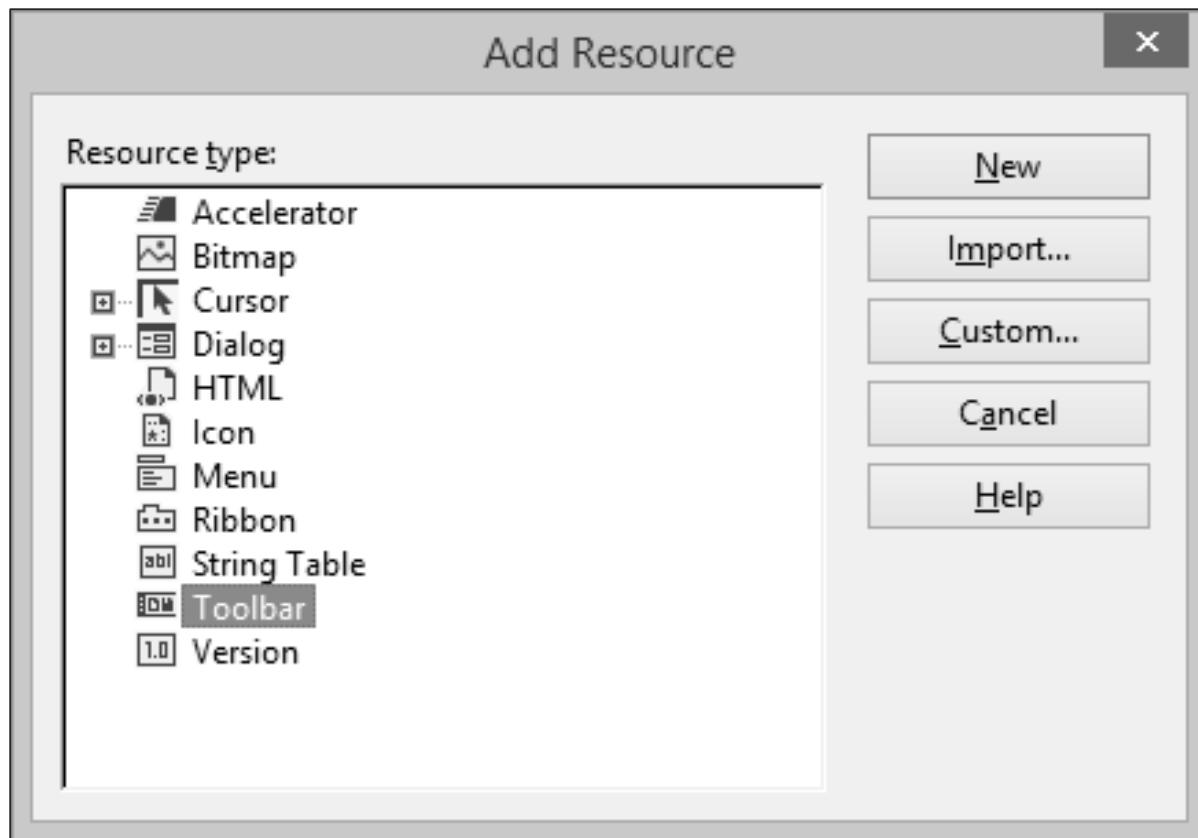
---

A **toolbar** is a Windows control that allows the user to perform some actions on a form by clicking a button instead of using a menu.

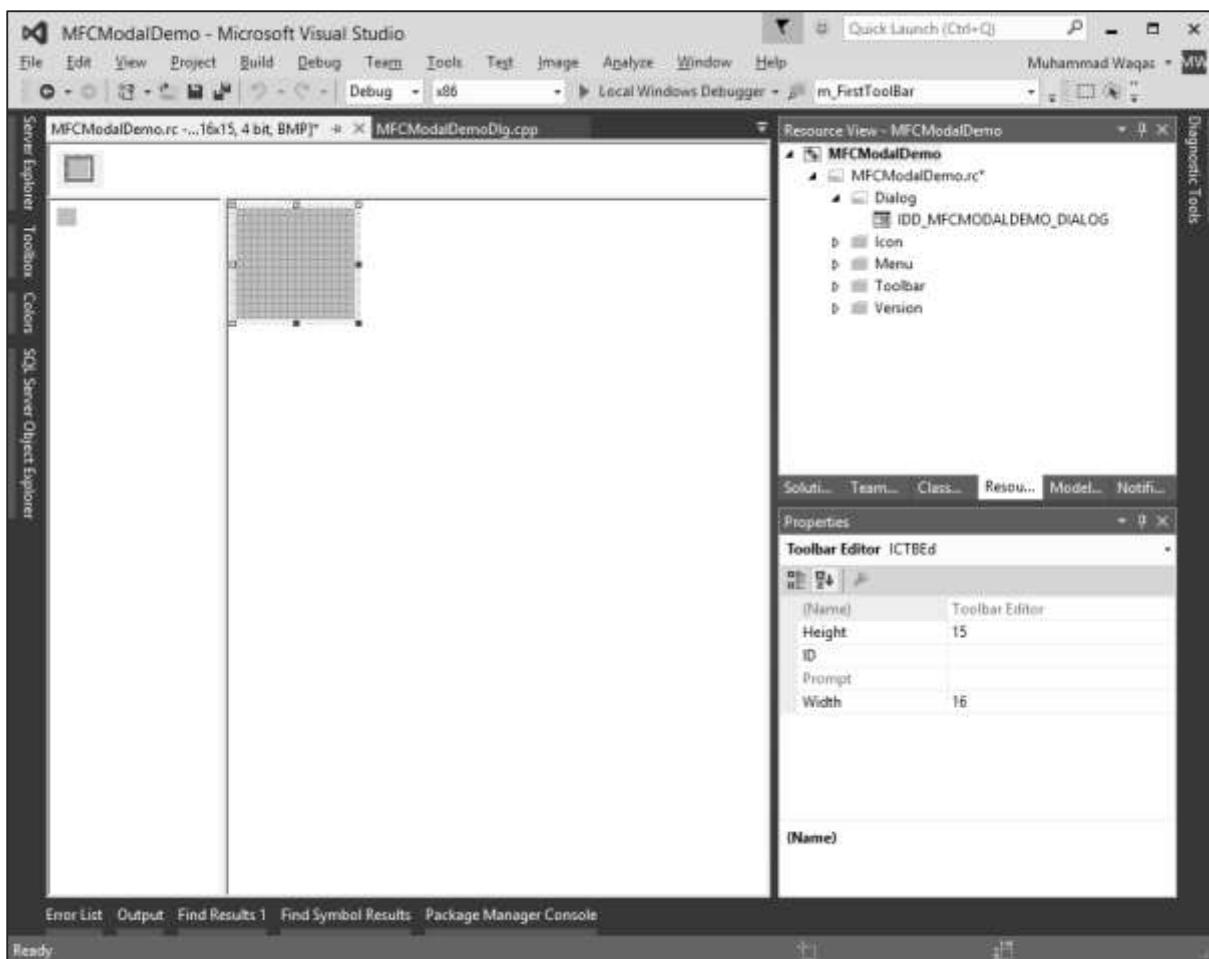
- A toolbar provides a convenient group of buttons that simplifies the user's job by bringing the most accessible actions as buttons.
- A toolbar can bring such common actions closer to the user.
- Toolbars usually display under the main menu.
- They can be equipped with buttons but sometimes their buttons or some of their buttons have a caption.
- Toolbars can also be equipped with other types of controls.

To create a toolbar, following are the steps.

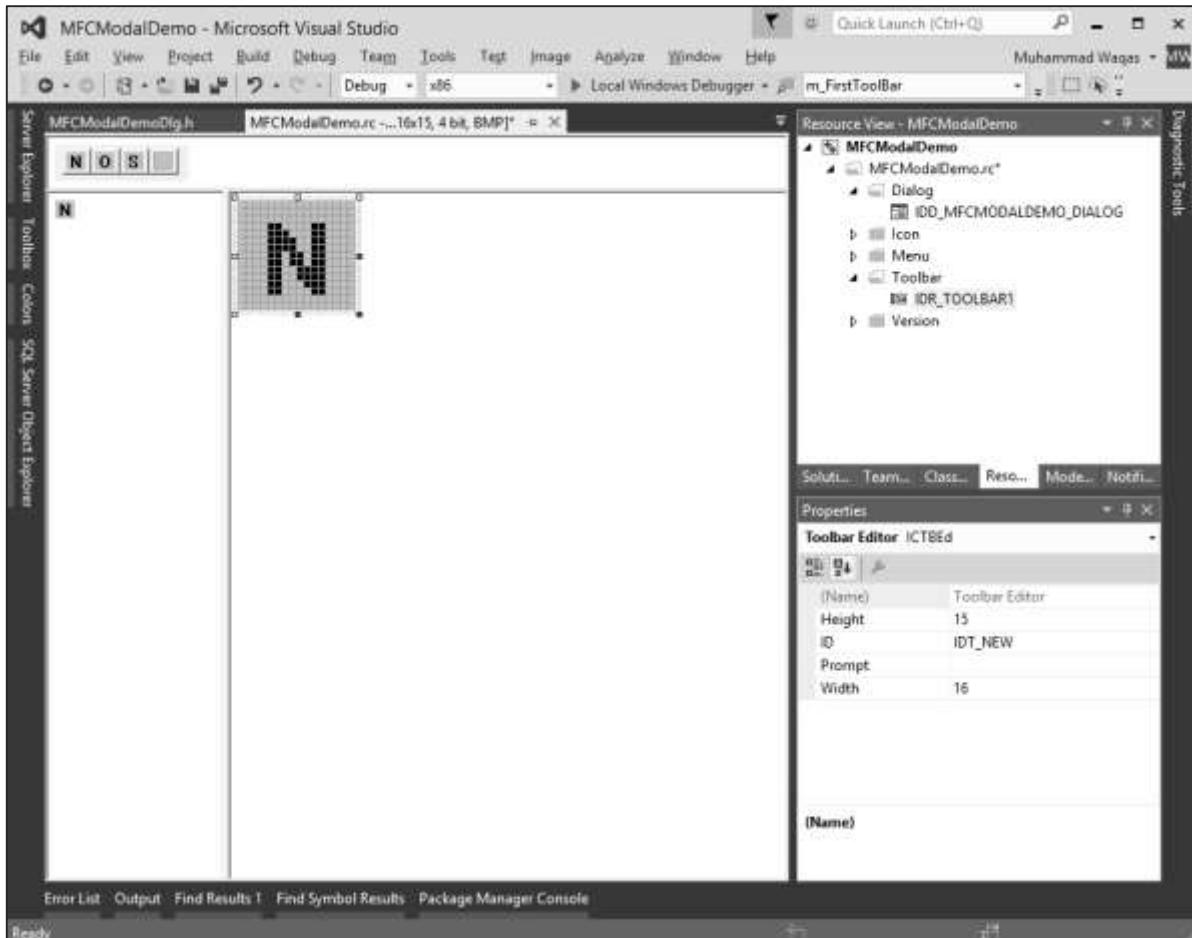
**Step 1:** Right-click on your project and select Add -> Resources. You will see the Add Resources dialog box.



**Step 2:** Select Toolbar and click New. You will see the following screen.



**Step 3:** Design your toolbar in the designer as shown in the following screenshot and specify the IDs as well.



**Step 4:** Add these two variables in CMFCModalDemoDlg class.

```
CToolBar m_wndToolBar;
BOOL butD;
```

**Step 5:** Following is the complete implementation of CMFCModalDemoDlg in CMFCModalDemoDlg.h file:

```
class CMFCModalDemoDlg : public CDialogEx
{
// Construction

public:
    CMFCModalDemoDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
```

71

```

#ifndef AFX DESIGN TIME
    enum { IDD = IDD_MFCMODALDEMO_DIALOG };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV support

// Implementation
protected:
    HICON m_hIcon;
    CToolBar m_wndToolBar;
    BOOL       butD;

// Generated message map functions
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
DECLARE_MESSAGE_MAP()

public:
    afx_msg void OnBnClickedOk();
};


```

**Step 6:** Update CMFCModalDemoDlg::OnInitDialog() as shown in the following code.

```

BOOL CMFCModalDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    if (!m_wndToolBar.Create(this)
        || !m_wndToolBar.LoadToolBar(IDR_TOOLBAR1))
        //if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |


```

```

//      WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |  

//      CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||  

//      !m_wndToolBar.LoadToolBar(IDR_TOOLBAR1))  

{  

    TRACE0("Failed to Create Dialog Toolbar\n");  

    EndDialog(IDCANCEL);  

}  

butD = TRUE;  

CRect rcClientOld; // Old Client Rect  

CRect rcClientNew; // New Client Rect with Toolbar Added  

// Retrive the Old Client WindowSize  

// Called to reposition and resize control bars in the client area  

of a window  

// The reposQuery FLAG does not really draw the Toolbar. It only  

does the calculations.  

// And puts the new ClientRect values in rcClientNew so we can do  

the rest of the Math.  

GetClientRect(rcClientOld);  

RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST,  

0,  

reposQuery, rcClientNew);  

// All of the Child Windows (Controls) now need to be moved so the  

Toolbar does not cover them up.  

// Offset to move all child controls after adding Toolbar  

CPoint ptOffset(rcClientNew.left - rcClientOld.left,  

rcClientNew.top - rcClientOld.top);  

CRect rcChild;  

CWnd* pwndChild = GetWindow(GW_CHILD); //Handle to the Dialog  

Controls  

while (pwndChild) // Cycle through all child controls  

{  

    pwndChild->GetWindowRect(rcChild); // Get the child control  

RECT  

    ScreenToClient(rcChild);  

// Changes the Child Rect by the values of the calculated  

offset

```

```
        rcChild.OffsetRect(ptOffset);
        pwndChild->MoveWindow(rcChild, FALSE); // Move the Child
Control
        pwndChild = pwndChild->GetNextWindow();

    }

    CRect rcWindow;
    // Get the RECT of the Dialog
    GetWindowRect(rcWindow);

    // Increase width to new Client Width
    rcWindow.right += rcClientOld.Width() - rcClientNew.Width();

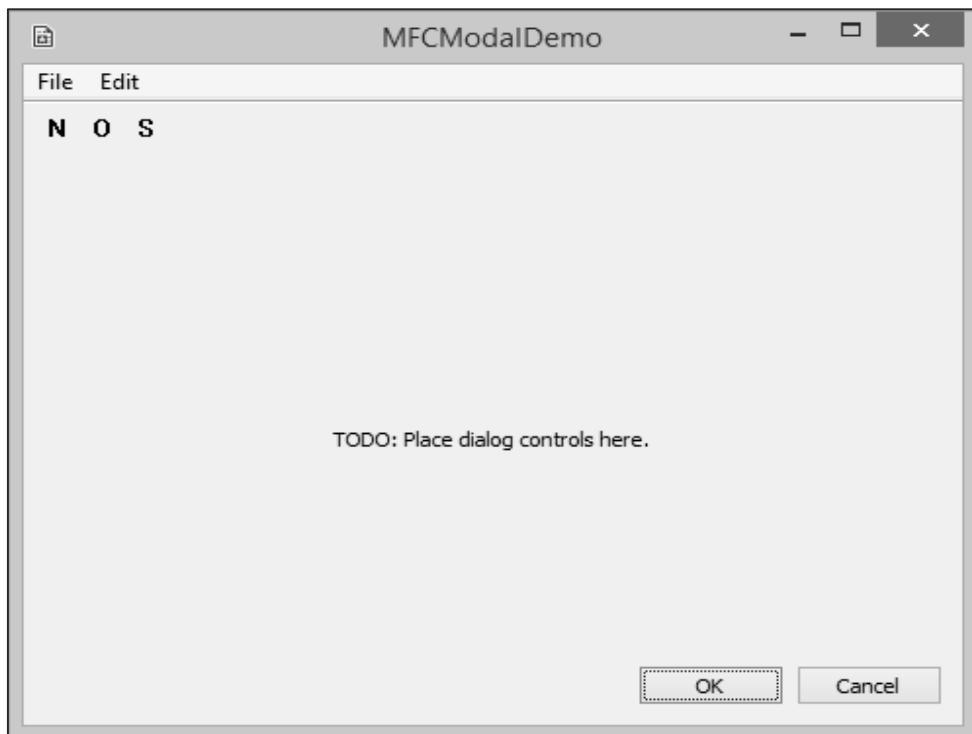
    // Increase height to new Client Height
    rcWindow.bottom += rcClientOld.Height() - rcClientNew.Height();
    // Redraw Window
    MoveWindow(rcWindow, FALSE);

    // Now we REALLY Redraw the Toolbar
    RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST,
0);

// TODO: Add extra initialization here

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 7:** Run this application. You will see the following dialog box which also contains the toolbar.

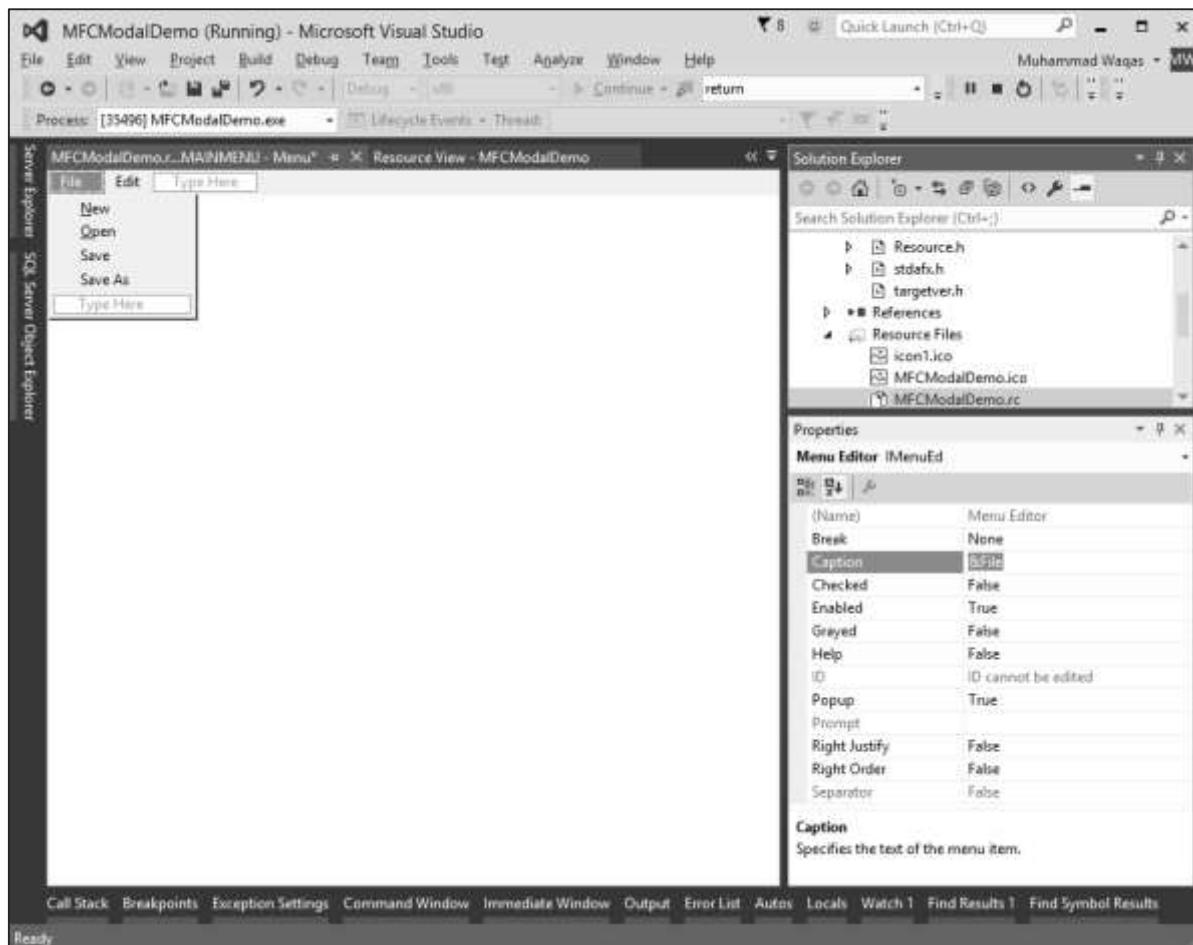


## Accelerators

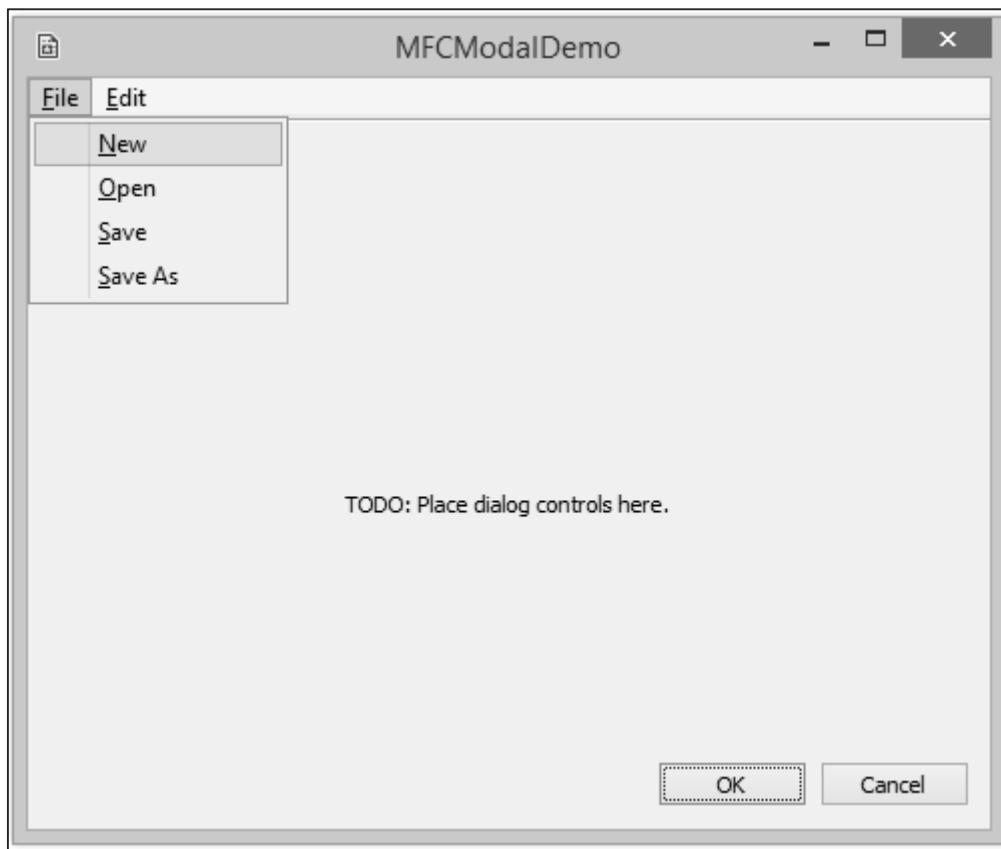
---

An **access key** is a letter that allows the user to perform a menu action faster by using the keyboard instead of the mouse. This is usually faster because the user would not need to position the mouse anywhere, which reduces the time it takes to perform the action.

**Step 1:** To create an access key, type an ampersand "&" on the left of the menu item.



**Step 2:** Repeat this step for all menu options. Run this application and press Alt. You will see that the first letter of all menu options are underlined.

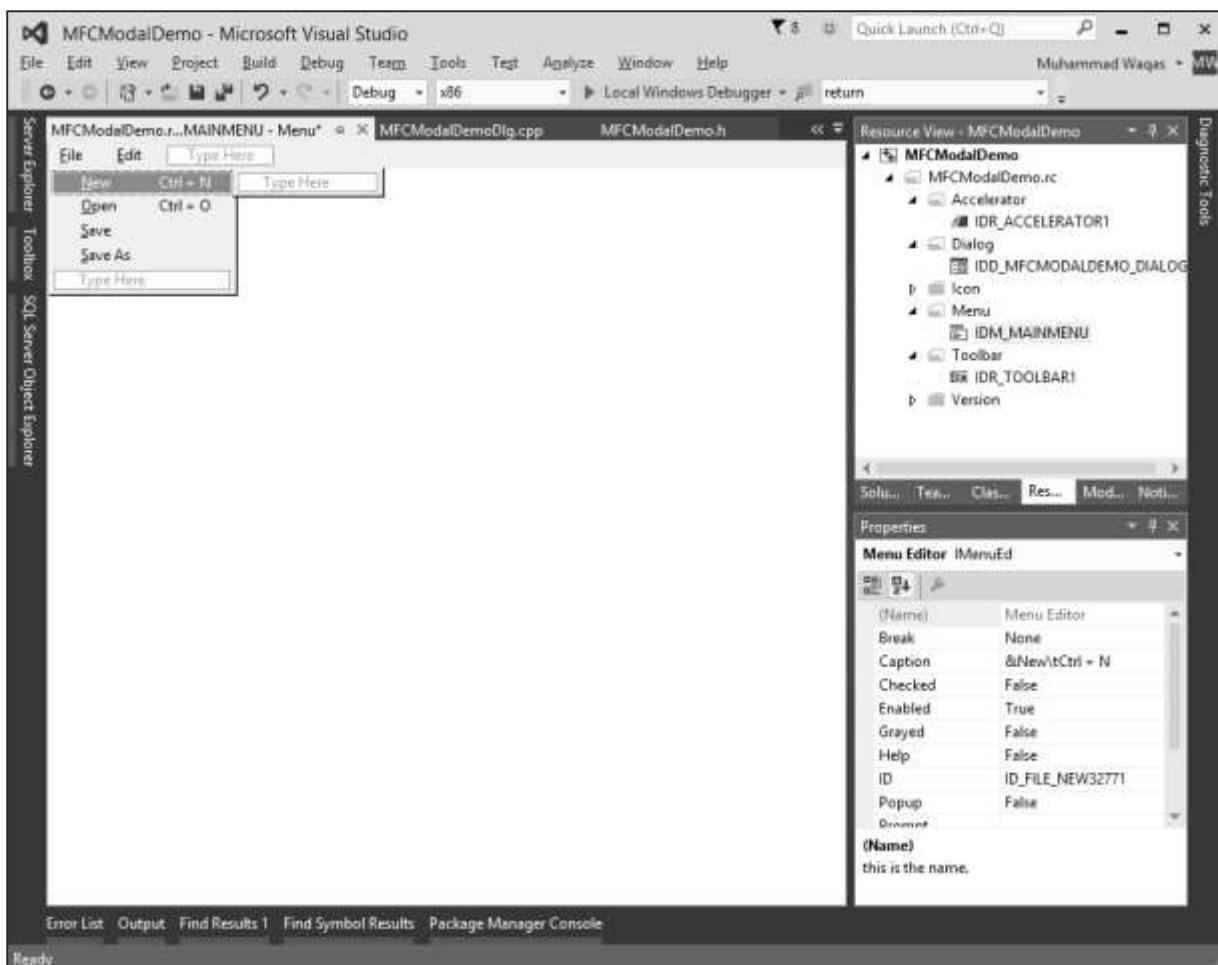


## Shortcut Key

A shortcut key is a key or a combination of keys used by advanced users to perform an action that would otherwise be done on a menu item. Most shortcuts are a combination of the Ctrl key simultaneously pressed with a letter key. For example, Ctrl + N, Ctrl + O, or Ctrl + D.

To create a shortcut, on the right side of the string that makes up a menu caption, right-click on the menu item and select properties.

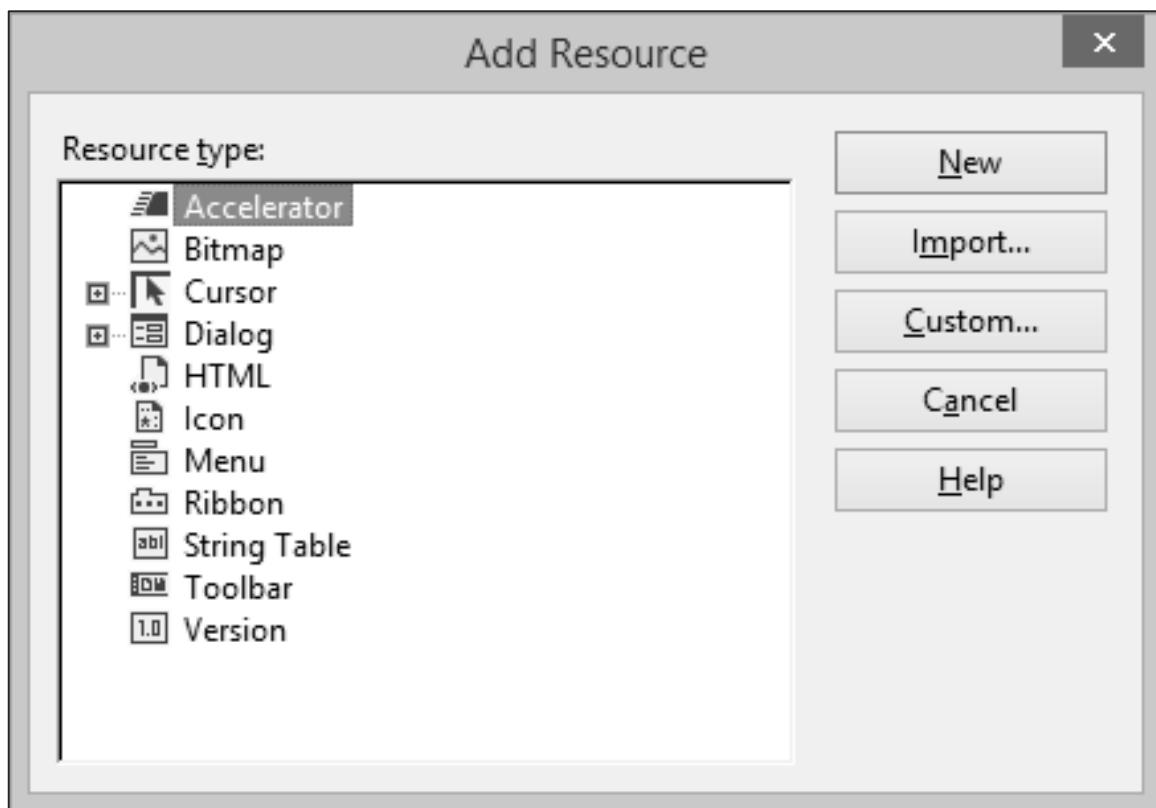
In the Caption field type \t followed by the desired combination as shown below for the New menu option. Repeat the step for all menu options.



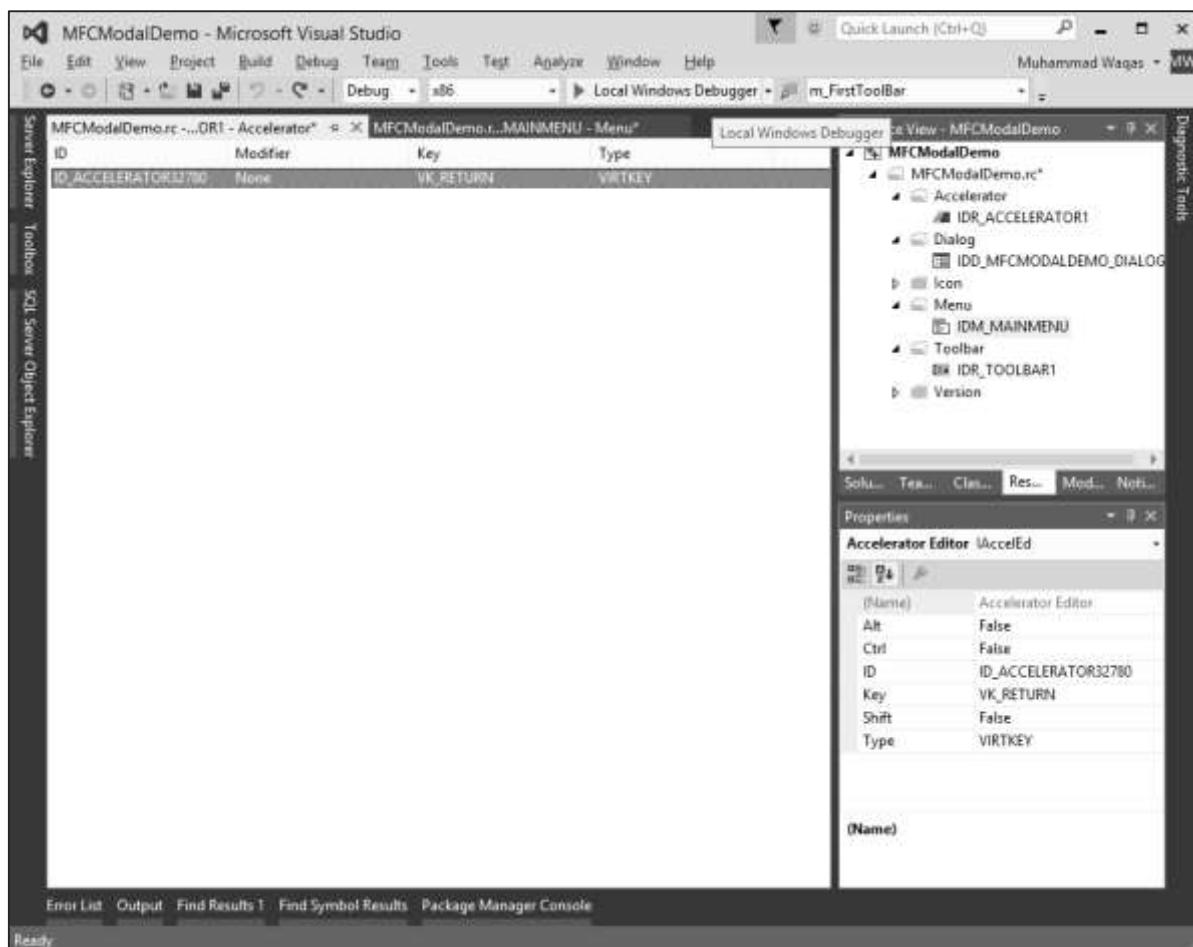
## Accelerator Table

An Accelerator Table is a list of items where each item of the table combines an identifier, a shortcut key, and a constant number that specifies the kind of accelerator key. Just like the other resources, an accelerator table can be created manually in a .rc file. Following are the steps to create an accelerator table.

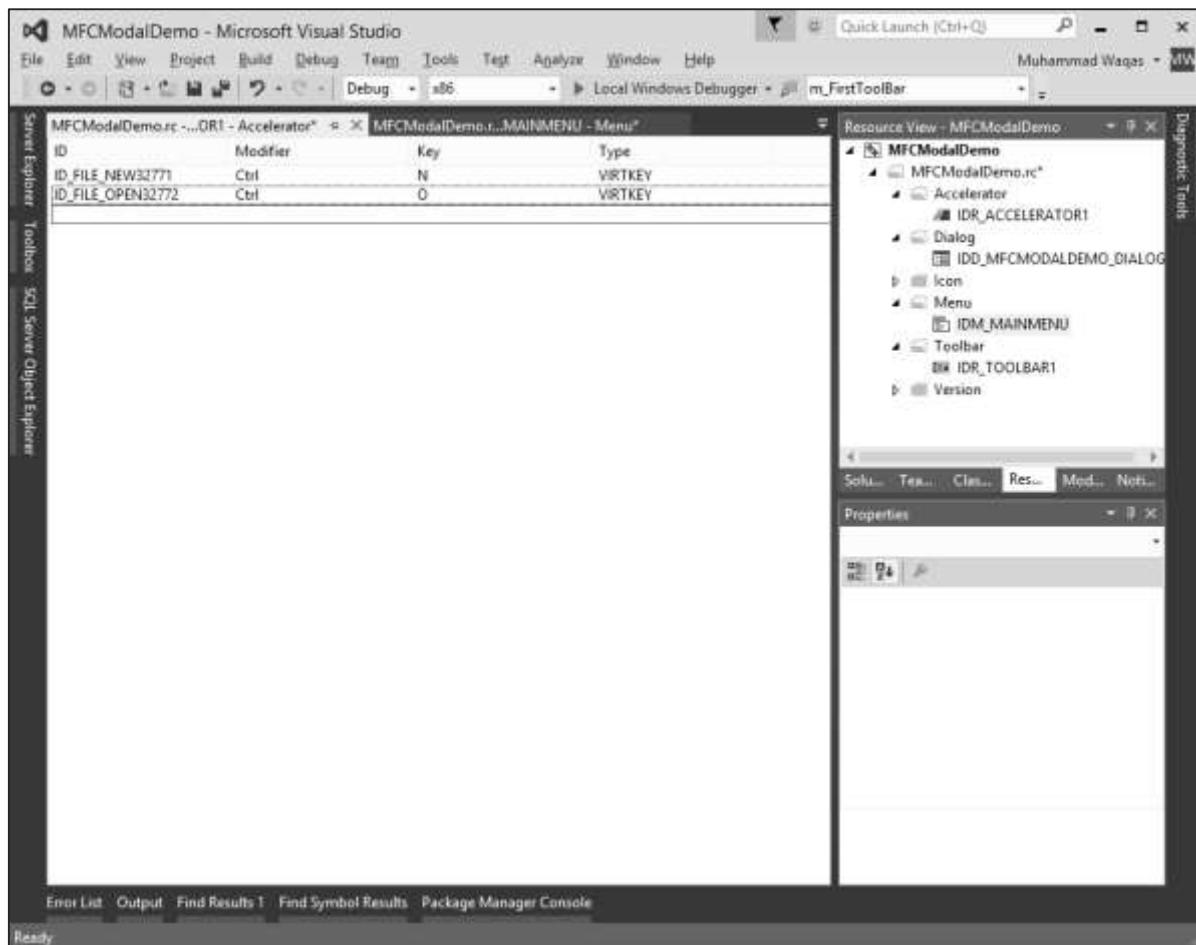
**Step 1:** To create an accelerator table, right-click on \*.rc file in the solution explorer.



**Step 2:** Select Accelerator and click New.



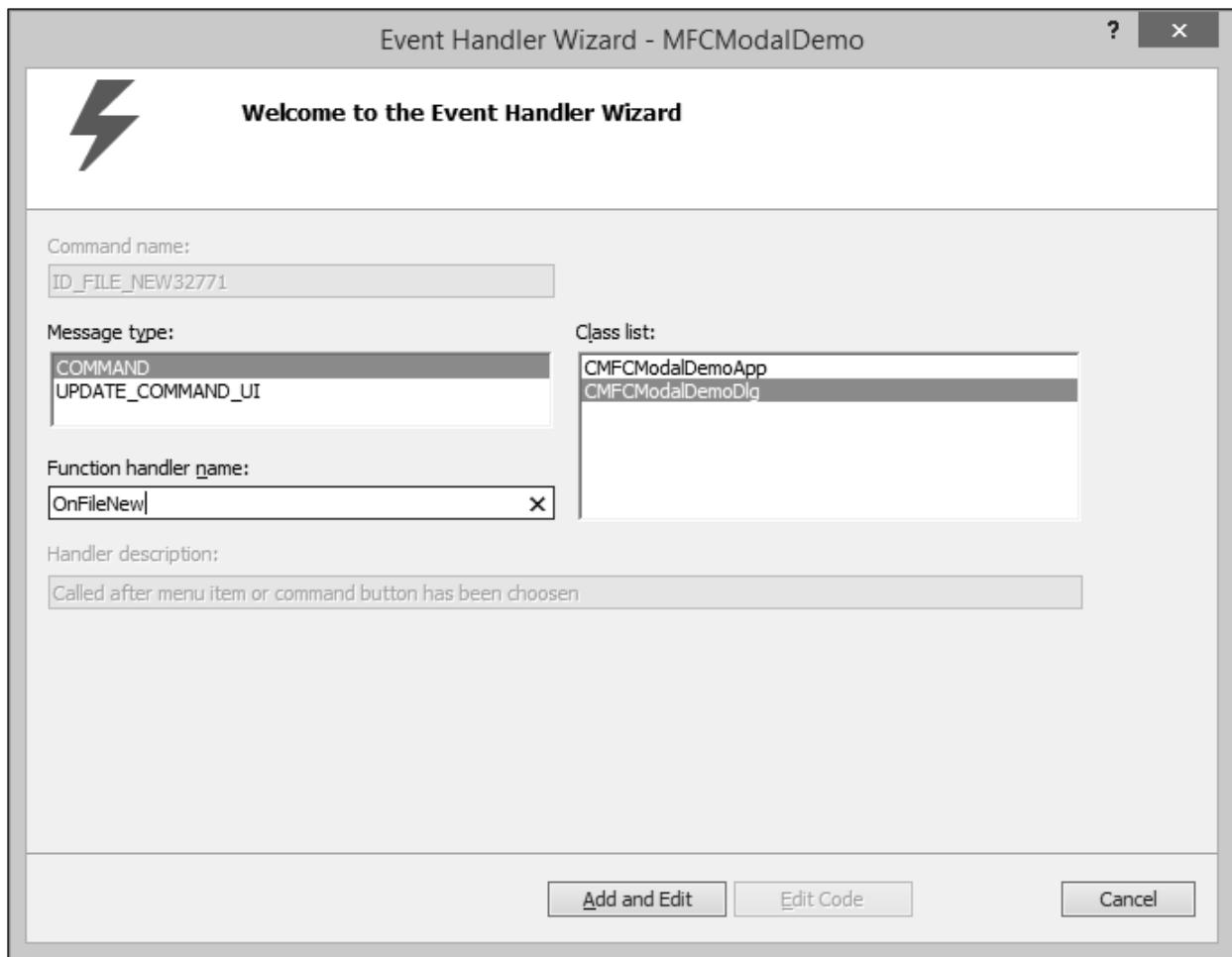
**Step 3:** Click the arrow of the ID combo box and select menu Items.



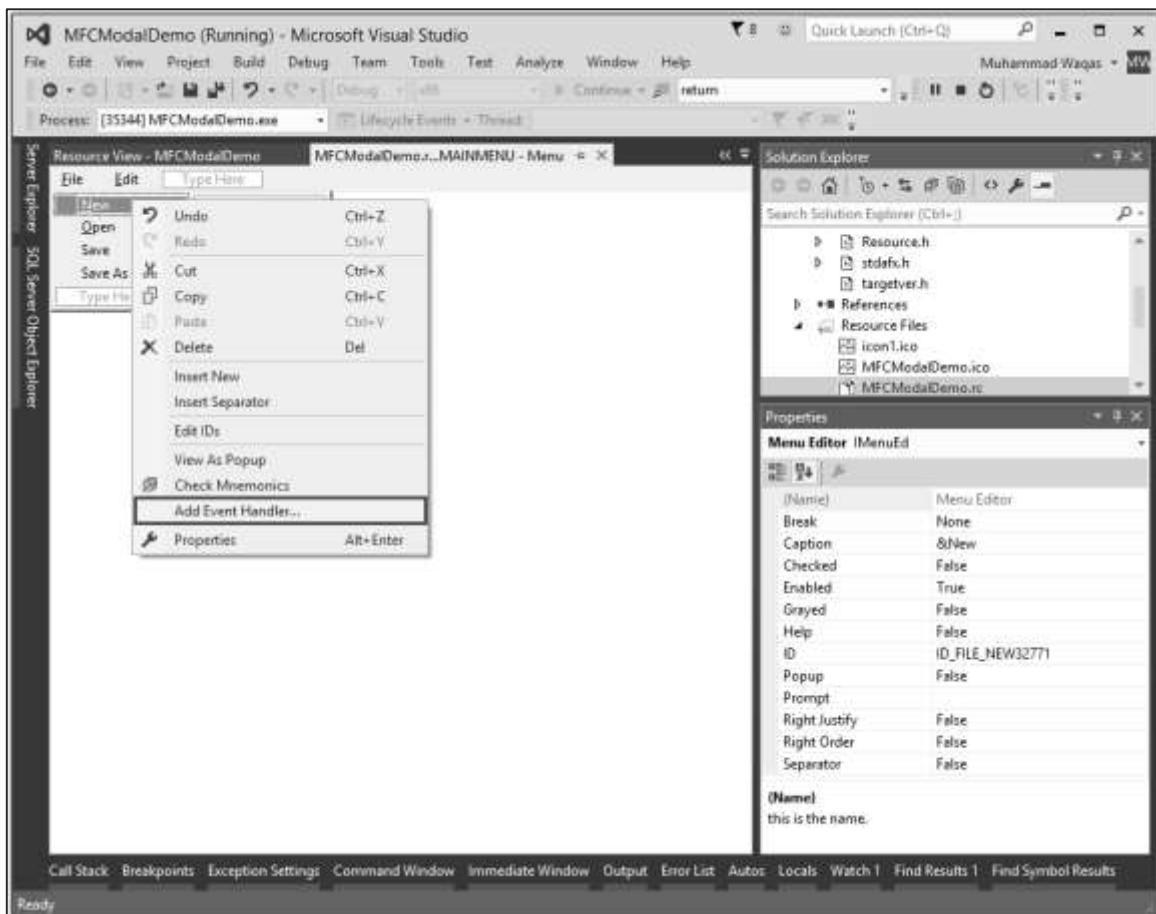
**Step 4:** Select Ctrl from the Modifier dropdown.

**Step 5:** Click the Key box and type the respective Keys for both menu options.

We will also add New menu item event handler to testing. Right-click on the New menu option.



**Step 6:** You can specify a class, message type and handler name. For now, let us leave it as it is and click Add and Edit button.



### Step 7: Select Add Event Handler

**Step 8:** You will now see the event added at the end of the CMFCModalDemoDlg.cpp file.

```
void CMFCModalDemoDlg::OnFileNew()
{
    // TODO: Add your command handler code here
    MessageBox(L"File > New menu option");
}
```

### Step 9: Now Let us add a message box that will display the simple menu option message.

To start accelerator table in working add the HACCEL variable and ProcessMessageFilter as shown in the following CMFCModalDemoApp.

```
class CMFCModalDemoApp : public CWinApp
{
public:
    CMFCModalDemoApp();
```

```

// Overrides

public:
    virtual BOOL InitInstance();
    HACCEL m_hAccelTable;

// Implementation

DECLARE_MESSAGE_MAP()
virtual BOOL ProcessMessageFilter(int code, LPMSG lpMsg);
};


```

**Step 10:** Load Accelerator and the following call in the CMFCModalDemoApp::InitInstance().

```

m_hAccelTable = LoadAccelerators(AfxGetInstanceHandle(),
MAKEINTRESOURCE(IDR_ACCELERATOR1));

```

**Step 11:** Here is the implementation of ProcessMessageFilter.

```

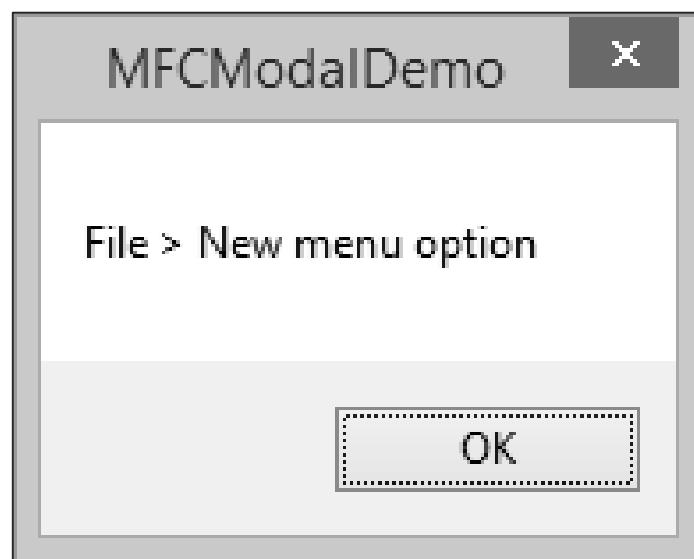
BOOL CMFCModalDemoApp::ProcessMessageFilter(int code, LPMSG lpMsg)
{
    if (code >= 0 && m_pMainWnd && m_hAccelTable)
    {
        if (::TranslateAccelerator(m_pMainWnd->m_hWnd, m_hAccelTable,
lpMsg))
            return TRUE;
    }
    return CWinApp::ProcessMessageFilter(code, lpMsg);
}

```

**Step 12:** When the above code is compiled and executed, you will see the following output.



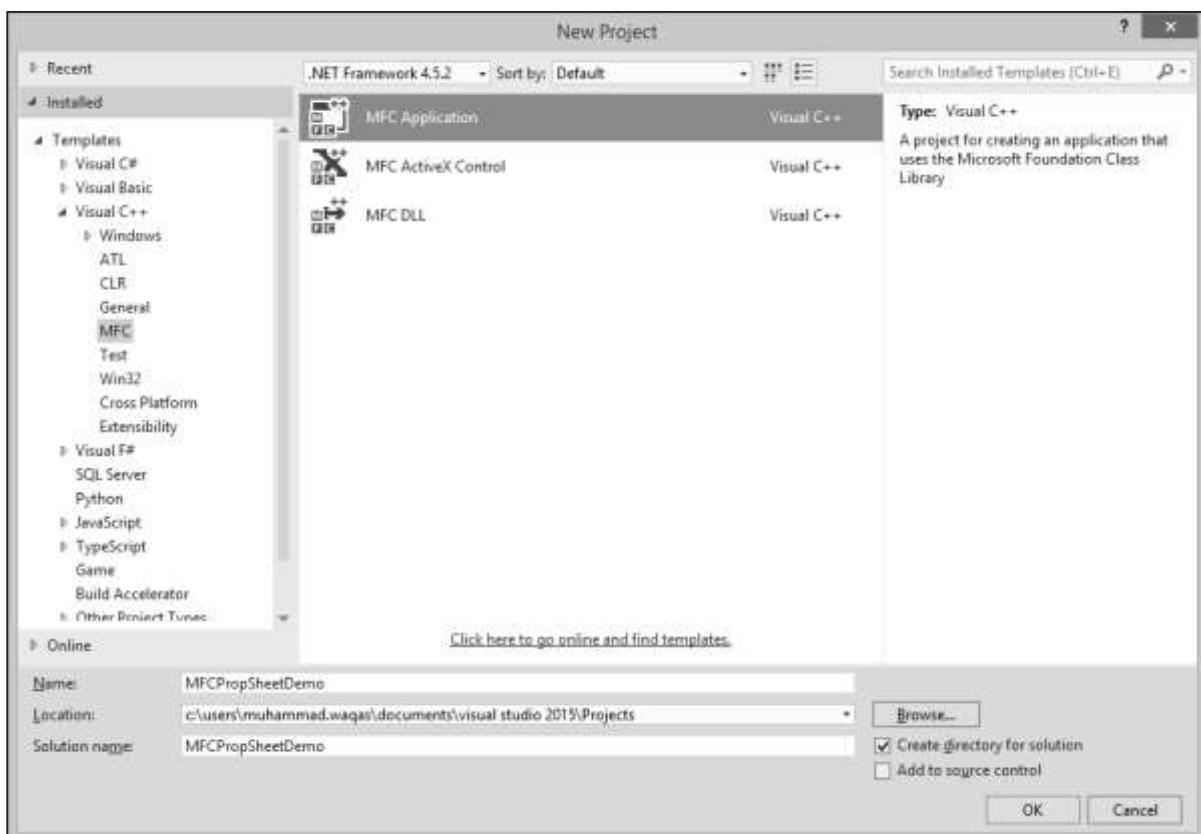
**Step 13:** Press Alt button followed by F key and then N key or Ctrl + N. You will see the following message.



## 8. MFC - Property Sheets

A **property sheet**, also known as a tab dialog box, is a dialog box that contains property pages. Each property page is based on a dialog template resource and contains controls. It is enclosed on a page with a tab on top. The tab names the page and indicates its purpose. Users click a tab in the property sheet to select a set of controls.

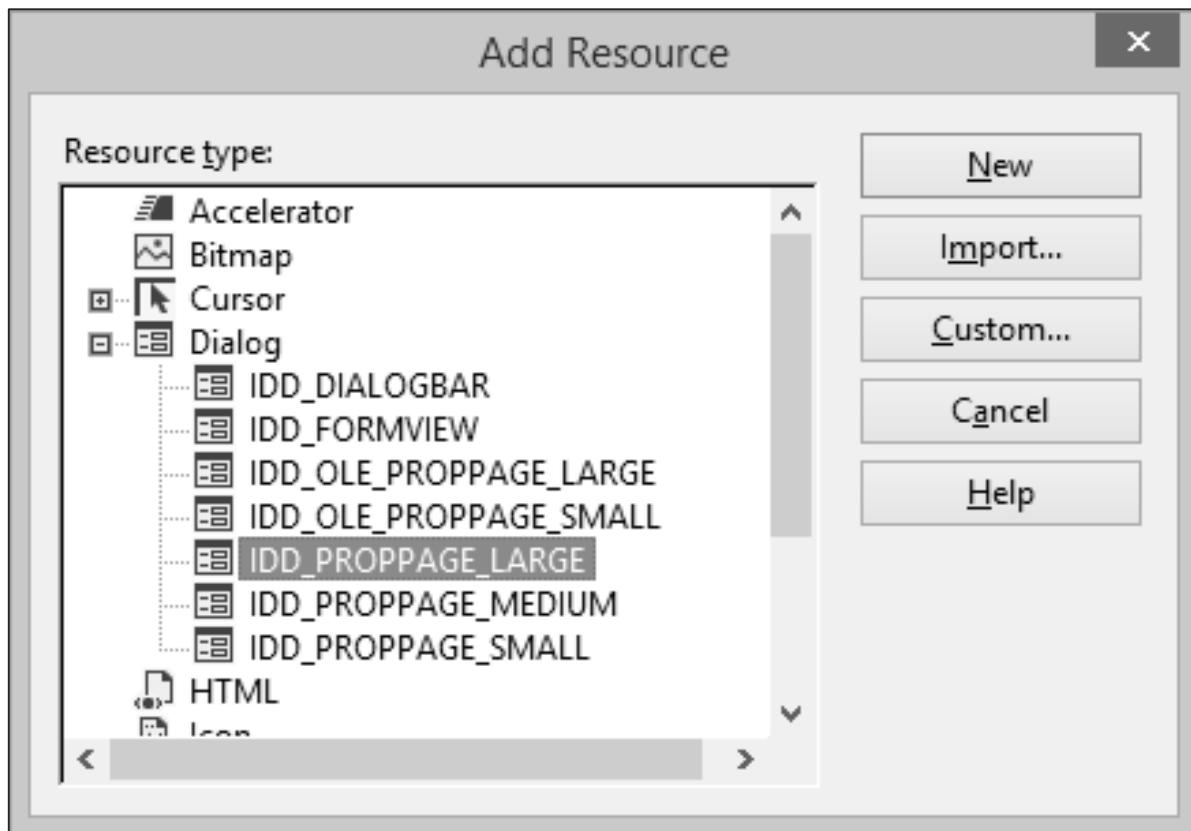
To create property pages, let us look into a simple example by creating a dialog based MFC project.



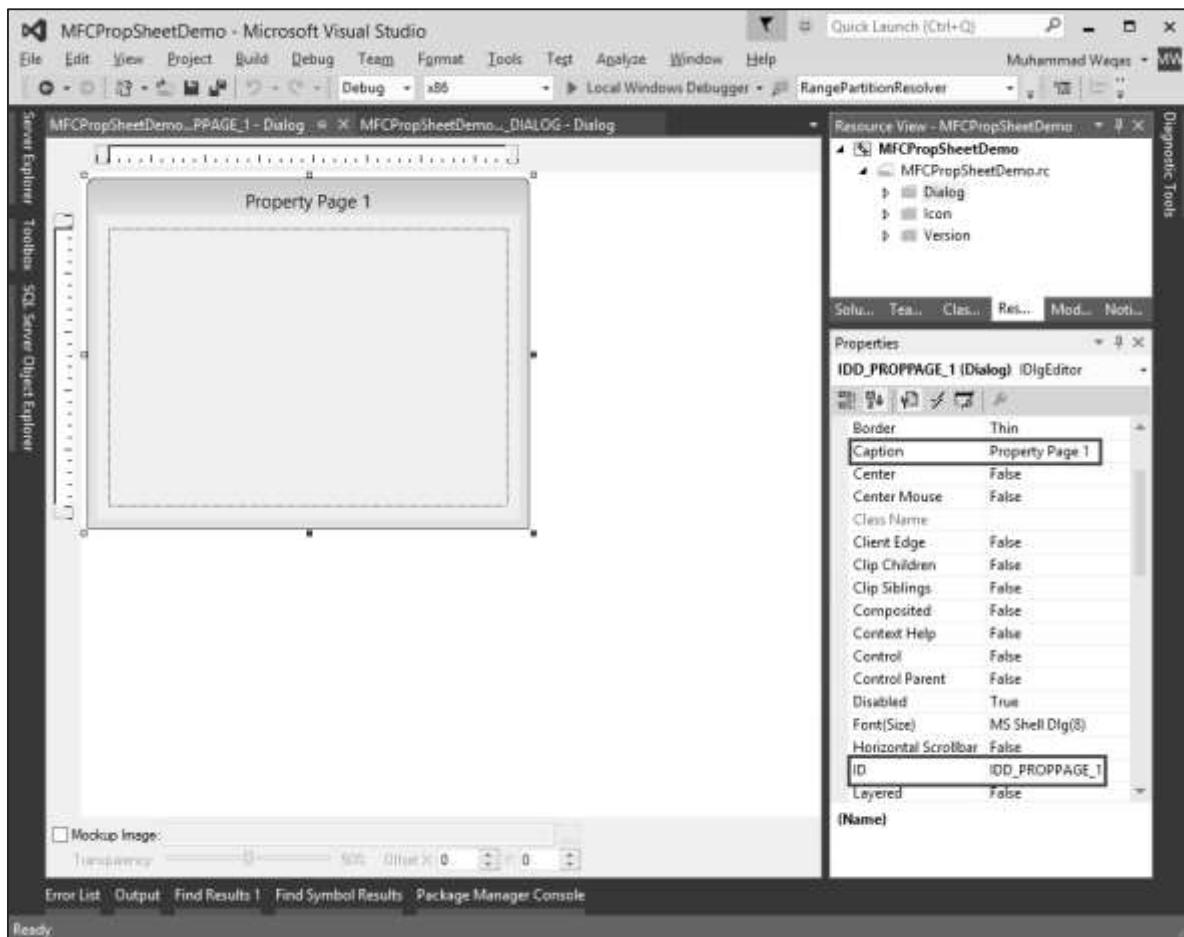
Once the project is created, we need to add some property pages.

Visual Studio makes it easy to create resources for property pages by displaying the Add Resource dialog box, expanding the Dialog node and selecting one of the IDD\_PROPSPAGE\_X items.

**Step 1:** Right-click on your project in solution explorer and select Add -> Resources.

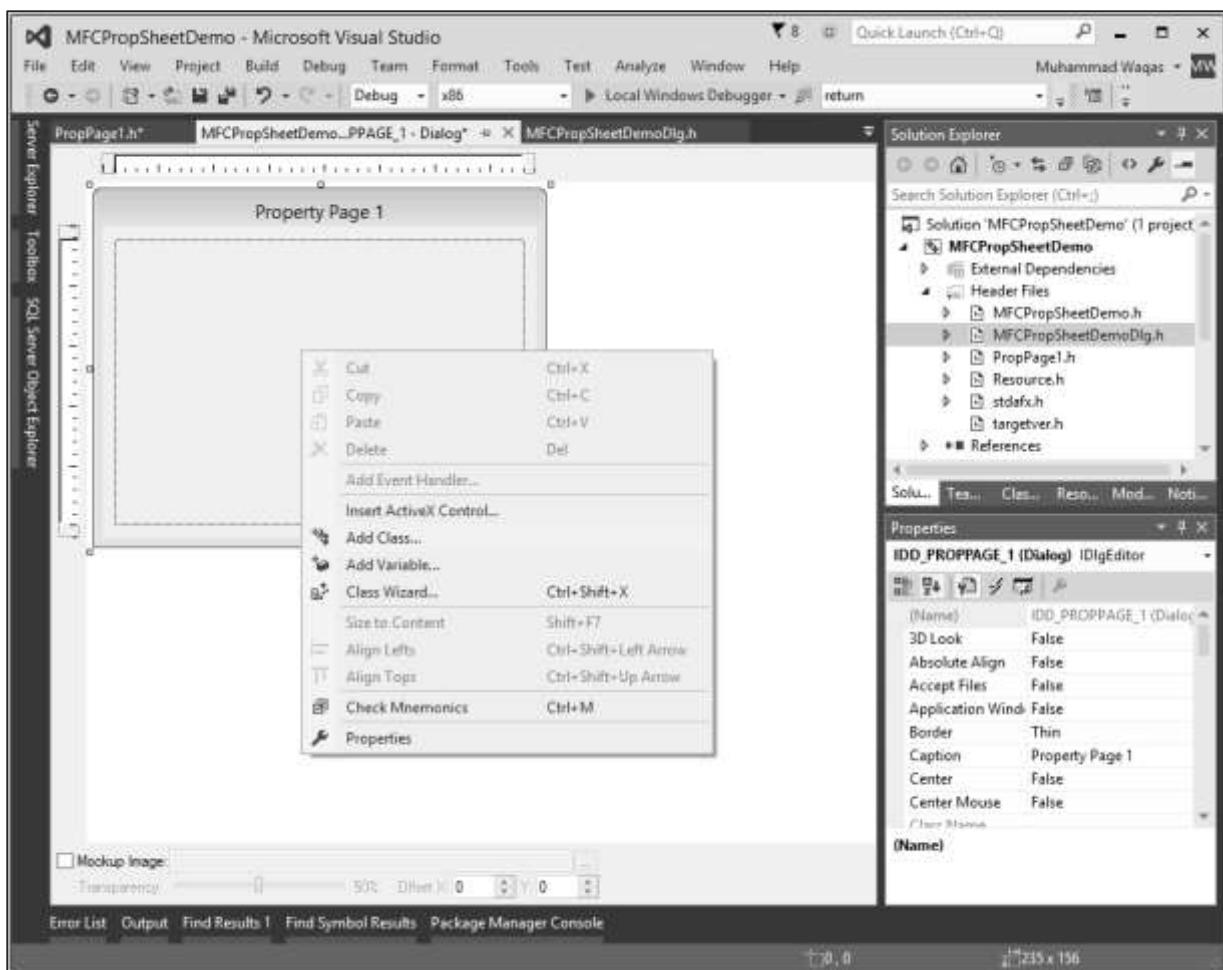


**Step 2:** Select the IDD\_PROPPAGE\_LARGE and click NEW.

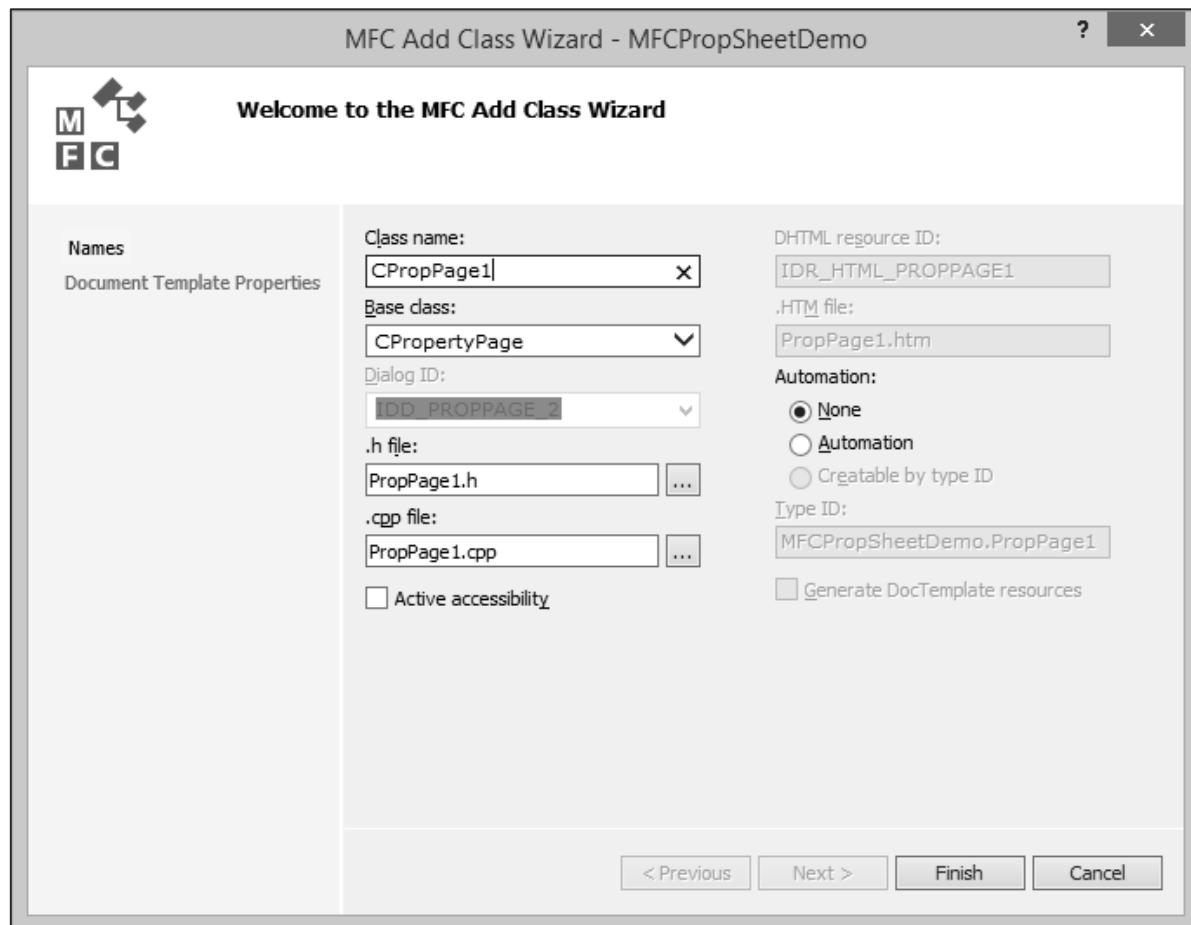


**Step 3:** Let us change ID and Caption of this property page to **IDD\_PROPPAGE\_1** and **Property Page 1** respectively as shown above.

**Step 4:** Right-click on the property page in designer window.



**Step 5:** Select the Add Class option.



**Step 6:** Enter the class name and select CPropertyPage from base class dropdown list.

**Step 7:** Click Finish to continue.

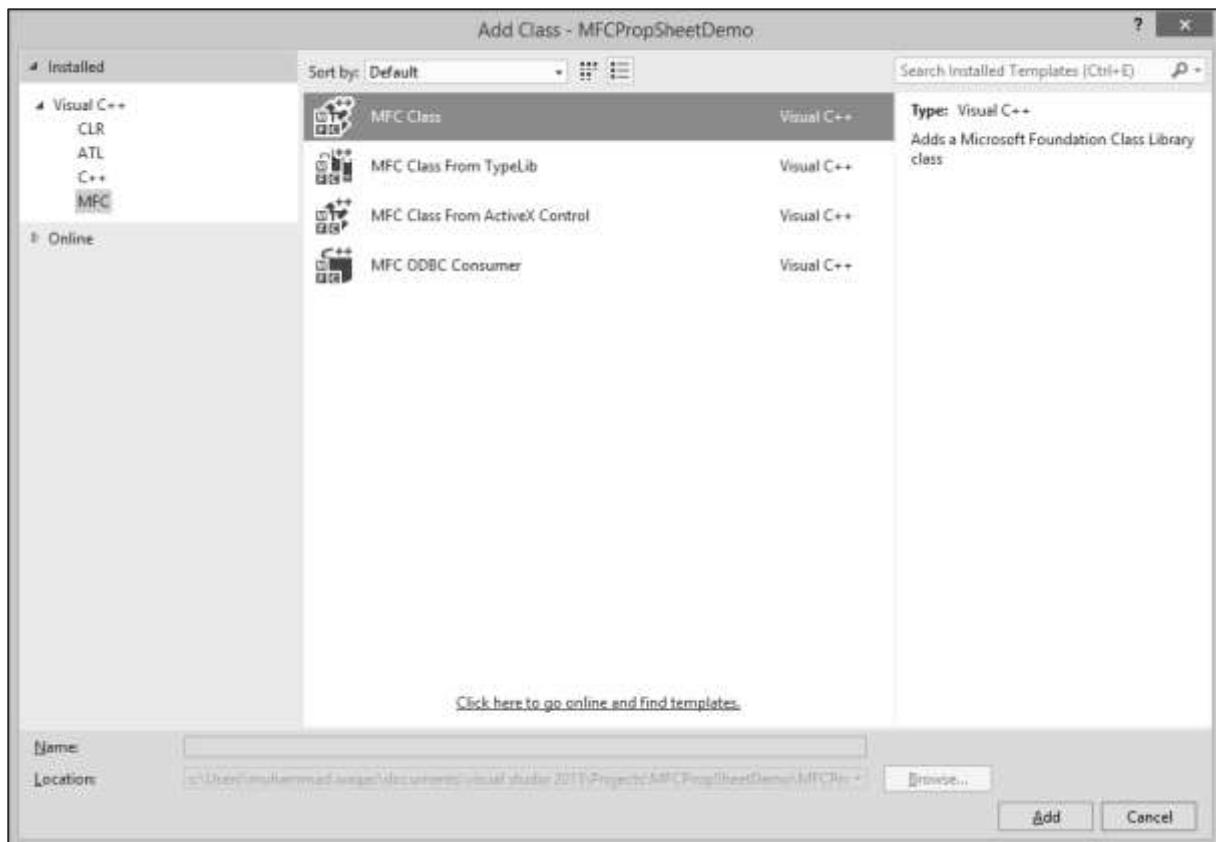
**Step 8:** Add one more property page with ID IDD\_PROPPAGE\_2 and Caption Property Page 2 by following the above mentioned steps.

**Step 9:** You can now see two property pages created. To implement its functionality, we need a property sheet.

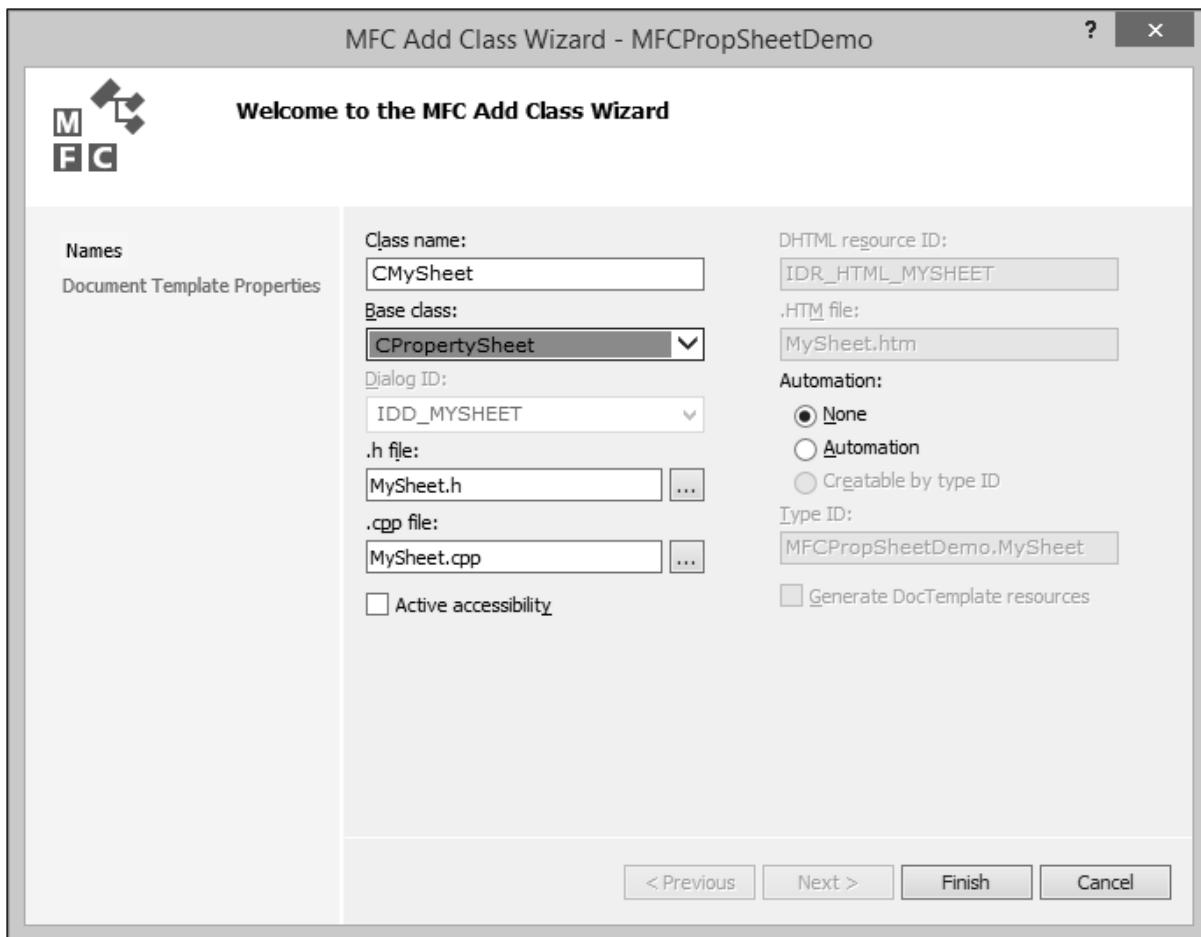
The Property Sheet groups the property pages together and keeps it as entity.

To create a property sheet, follow the steps given below:

**Step 1:** Right-click on your project and select Add > Class menu options



**Step 2:** Select Visual C++ -> MFC from the left pane and MFC Class in the template pane and click Add.



**Step 3:** Enter the class name and select CPropertySheet from base class dropdown list.

**Step 4:** Click finish to continue.

**Step 5:** To launch this property sheet, we need the following changes in our main project class.

**Step 6:** Add the following references in CMFCPropSheetDemo.cpp file.

```
#include "MySheet.h"
#include "PropPage1.h"
#include "PropPage2.h"
```

**Step 7:** Modify the CMFCPropSheetDemoApp::InitInstance() method as shown in the following code.

```
CMysheet mySheet(L"Property Sheet Demo");
CPropPage1 page1;
CPropPage2 page2;

mySheet.AddPage(&page1);
```

92

```

mySheet.AddPage(&page2);

m_pMainWnd = &mySheet;
INT_PTR nResponse = mySheet.DoModal();

```

**Step 8:** Here is the complete implementation of CMFCPropSheetDemo.cpp file.

```

// MFCPropSheetDemo.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "MFCPropSheetDemo.h"
#include "MFCPropSheetDemoDlg.h"
#include "MySheet.h"
#include "PropPage1.h"
#include "PropPage2.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif

// CMFCPropSheetDemoApp

BEGIN_MESSAGE_MAP(CMFCPropSheetDemoApp, CWinApp)
    ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()

// CMFCPropSheetDemoApp construction

CMFCPropSheetDemoApp::CMFCPropSheetDemoApp()
{
    // support Restart Manager
    m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_RESTART;
}

```

```
// TODO: add construction code here,
// Place all significant initialization in InitInstance
}

// The one and only CMFCPropSheetDemoApp object

CMFCPropSheetDemoApp theApp;

// CMFCPropSheetDemoApp initialization

BOOL CMFCPropSheetDemoApp::InitInstance()
{
    // InitCommonControlsEx() is required on Windows XP if an application
    // manifest specifies use of ComCtl32.dll version 6 or later to enable
    // visual styles. Otherwise, any window creation will fail.
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // Set this to include all the common control classes you want to use
    // in your application.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();

    AfxEnableControlContainer();

    // Create the shell manager, in case the dialog contains
    // any shell tree view or shell list view controls.
    CShellManager *pShellManager = new CShellManager;

    // Activate "Windows Native" visual manager for enabling themes in MFC
    // controls
    CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows));
}
```

```

// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need
// Change the registry key under which our settings are stored
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization
SetRegistryKey(_T("Local AppWizard-Generated Applications"));

CMySheet mySheet(L"Property Sheet Demo");
CPropPage1 page1;
CPropPage2 page2;

mySheet.AddPage(&page1);
mySheet.AddPage(&page2);

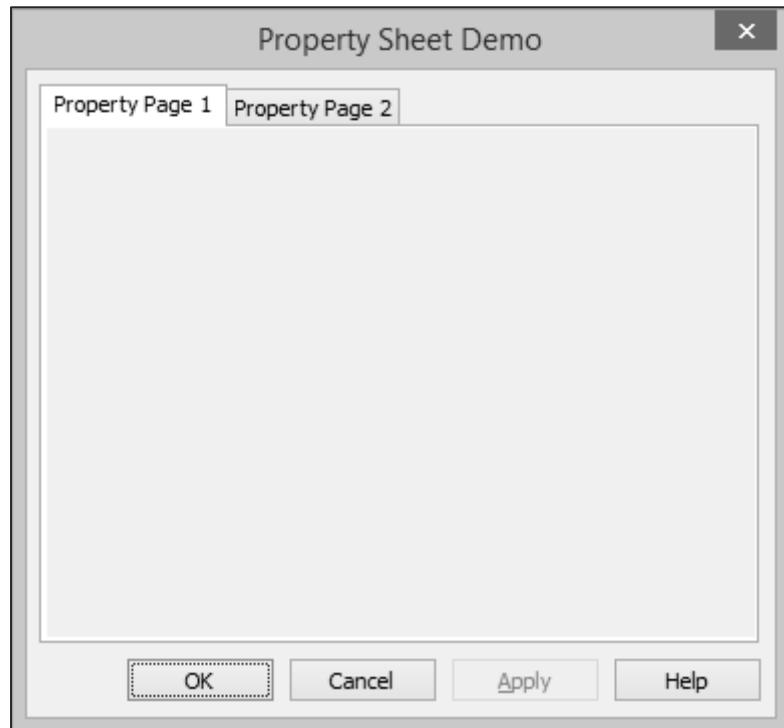
m_pMainWnd = &mySheet;
INT_PTR nResponse = mySheet.DoModal();
if (nResponse == IDOK)
{
    // TODO: Place code here to handle when the dialog is
    // dismissed with OK
}
else if (nResponse == IDCANCEL)
{
    // TODO: Place code here to handle when the dialog is
    // dismissed with Cancel
}
else if (nResponse == -1)
{
    TRACE(traceAppMsg, 0, "Warning: dialog creation failed, so
application is terminating unexpectedly.\n");
    TRACE(traceAppMsg, 0, "Warning: if you are using MFC controls on
the dialog, you cannot #define _AFX_NO_MFC_CONTROLS_IN_DIALOGS.\n");
}

// Delete the shell manager created above.
if (pShellManager != NULL)
{

```

```
    delete pShellManager;  
}  
  
// Since the dialog has been closed, return FALSE so that we exit the  
// application, rather than start the application's message pump.  
return FALSE;}
```

**Step 9:** When the above code is compiled and executed, you will see the following dialog box. This dialog box contains two property pages.



# 9. MFC - Windows Layout

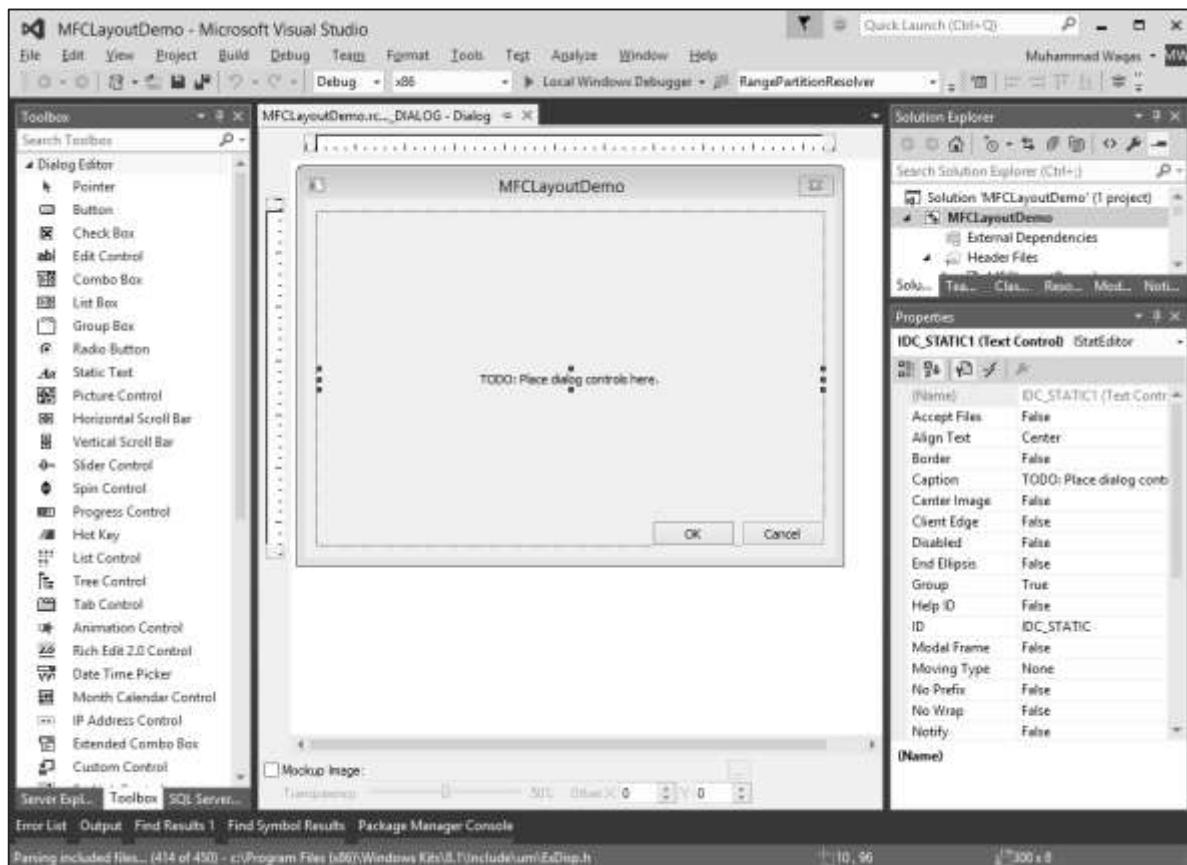
**Layout of controls** is very important and critical for application usability. It is used to arrange a group of GUI elements in your application. There are certain important things to consider while selecting layout:

- Positions of the child elements.
- Sizes of the child elements.

## Adding controls

Let us create new Dialog based MFC Project MFCLayoutDemo.

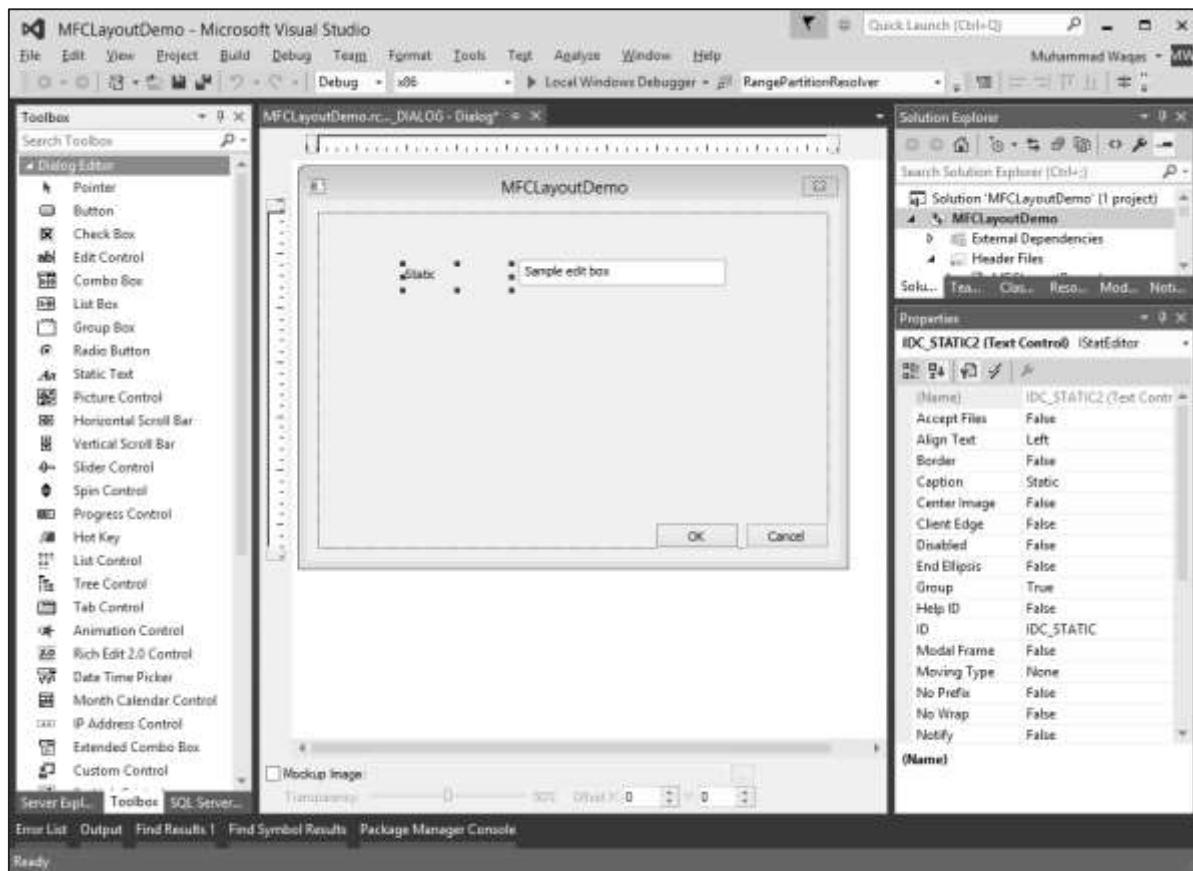
**Step 1:** Once the project is created, you will see the following screen.



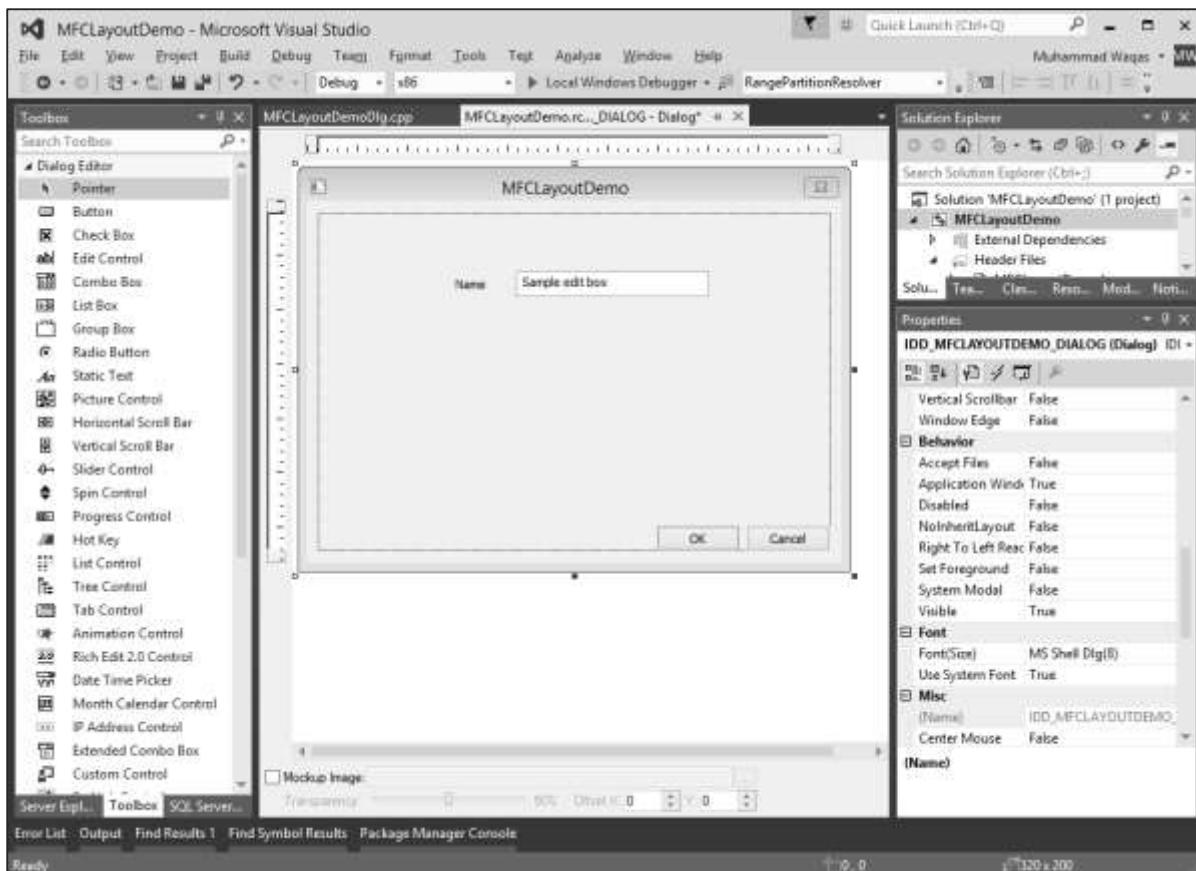
**Step 2:** Delete the TODO from the dialog box.

**Step 3:** Drag some controls from the Toolbox which you can see on the left side.

(We will drag one Static Text and one Edit Control as shown in the following snapshot).



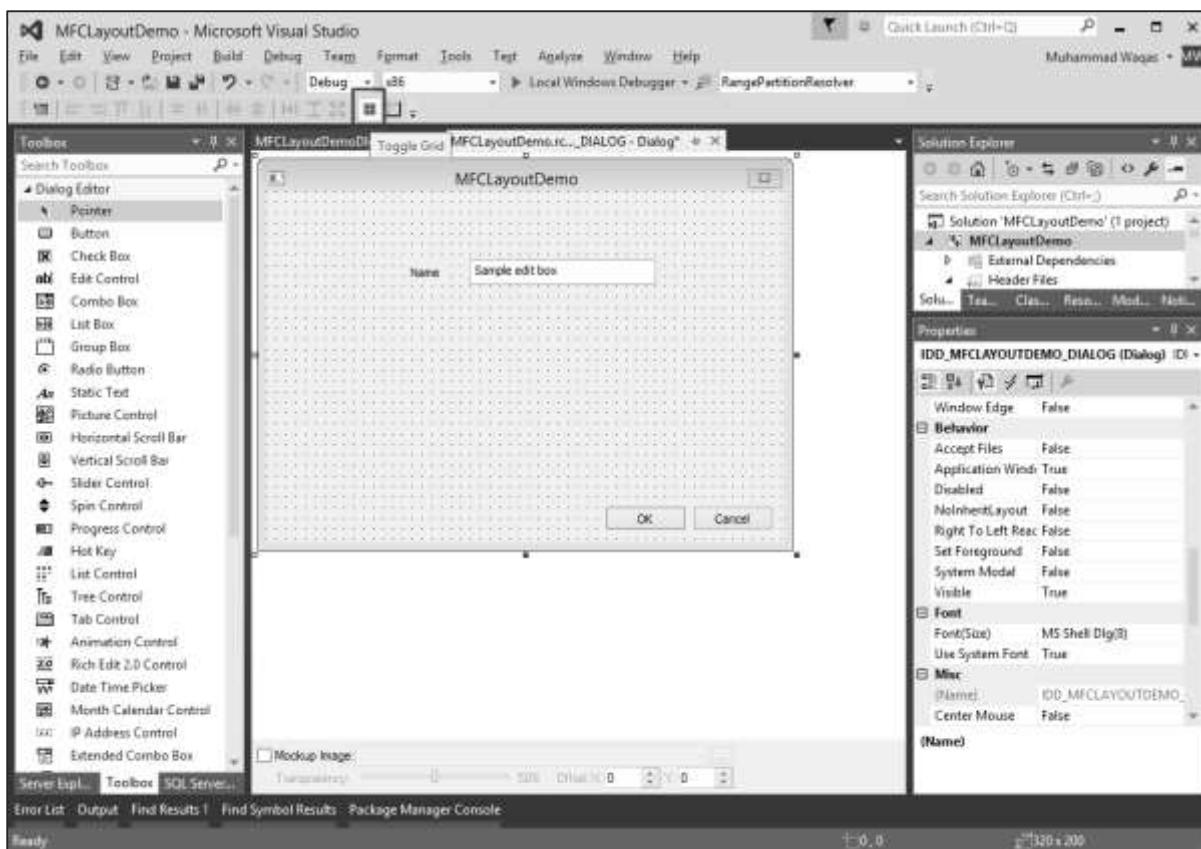
**Step 4:** Change the Caption of the Static Text to Name.



## Control Grid

Control grid is the guiding grid dots, which can help in positioning of the controls you are adding at the time of designing.

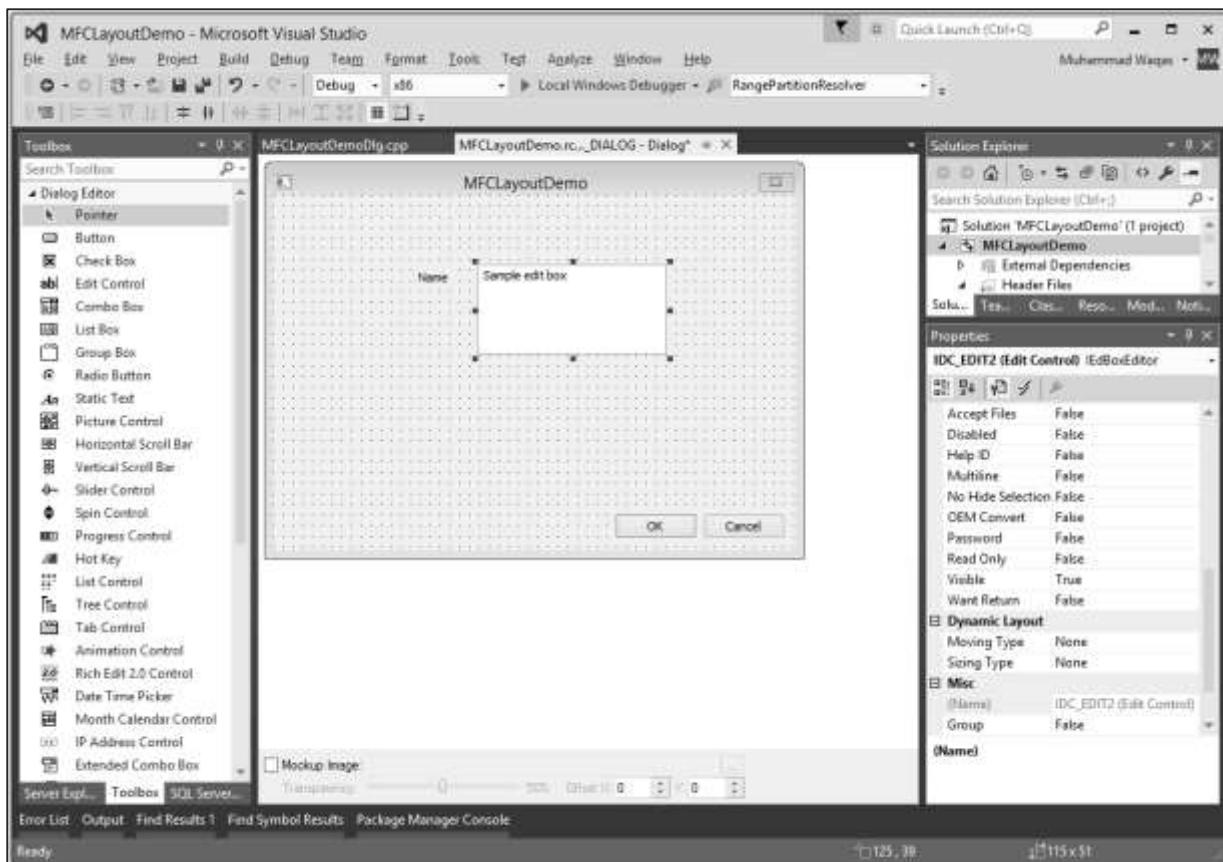
To enable the control grid, you need to click the Toggle Grid button in the toolbar as shown in the following snapshot.



## Controls Resizing

After you have added a control to a dialog box, it assumes either its default size or the size you drew it with. To help with the sizes of controls on the form or dialog box, Visual Studio provides a visual grid made of black points.

To resize a control, that is, to give it a particular width or height, position the mouse on one of the handles and drag it in the desired direction.

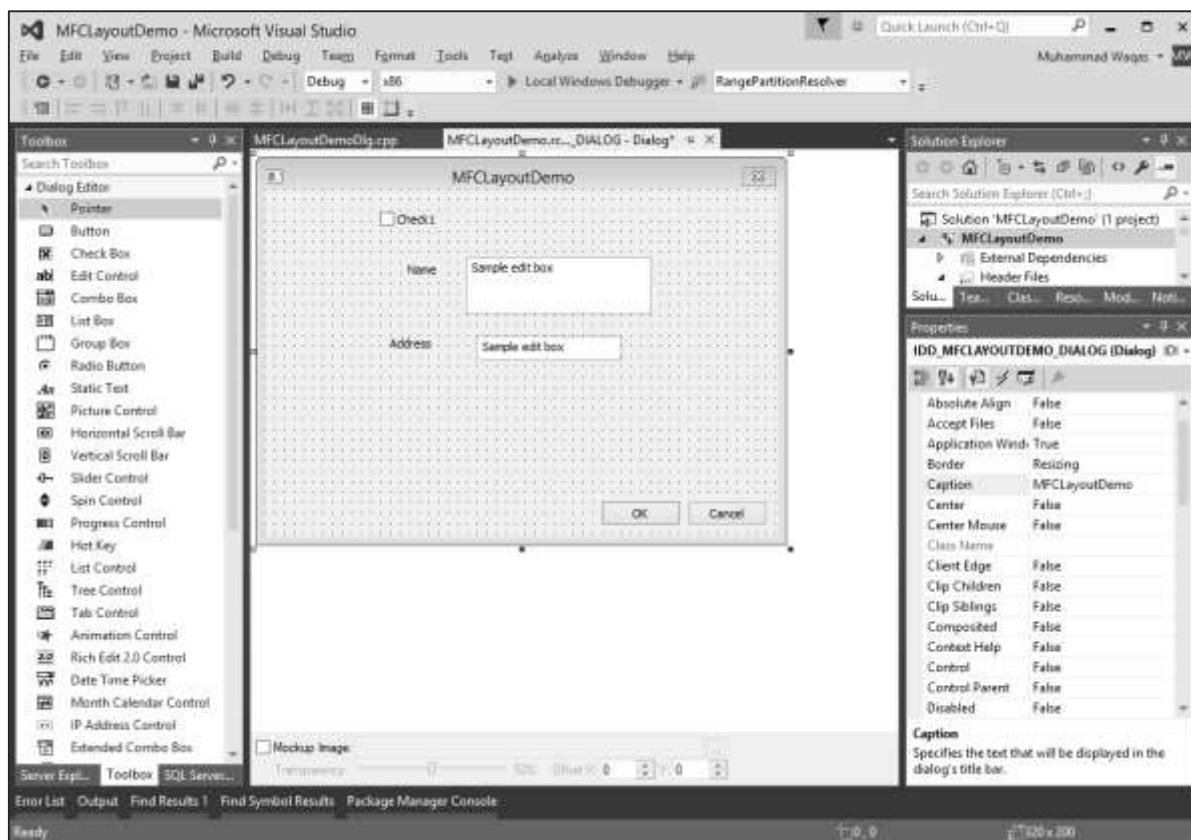


You can now resize the controls with the help of this dotted grid.

## Controls Positions

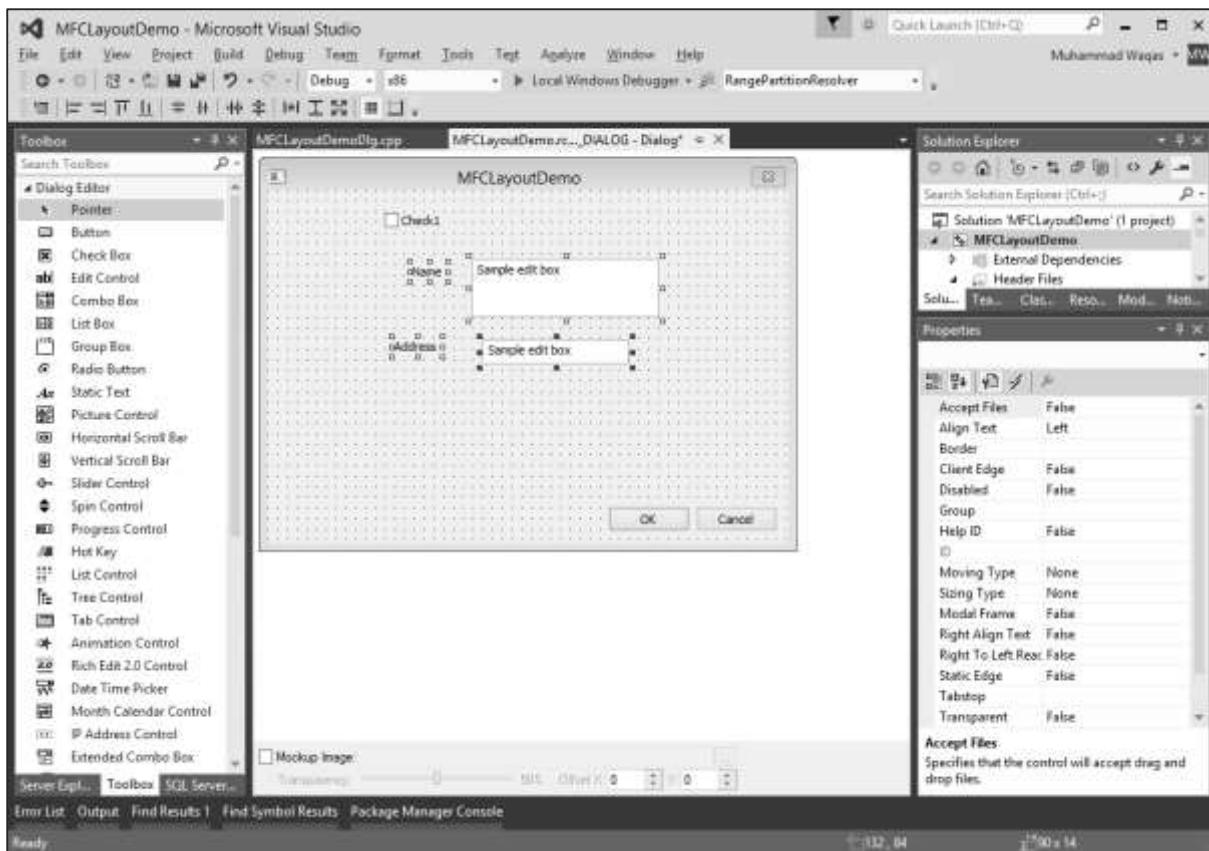
The controls you position on a dialog box or a form assume their given place. Most of the time, these positions are not practical. You can move them around to any position of your choice.

Let us add some more controls:

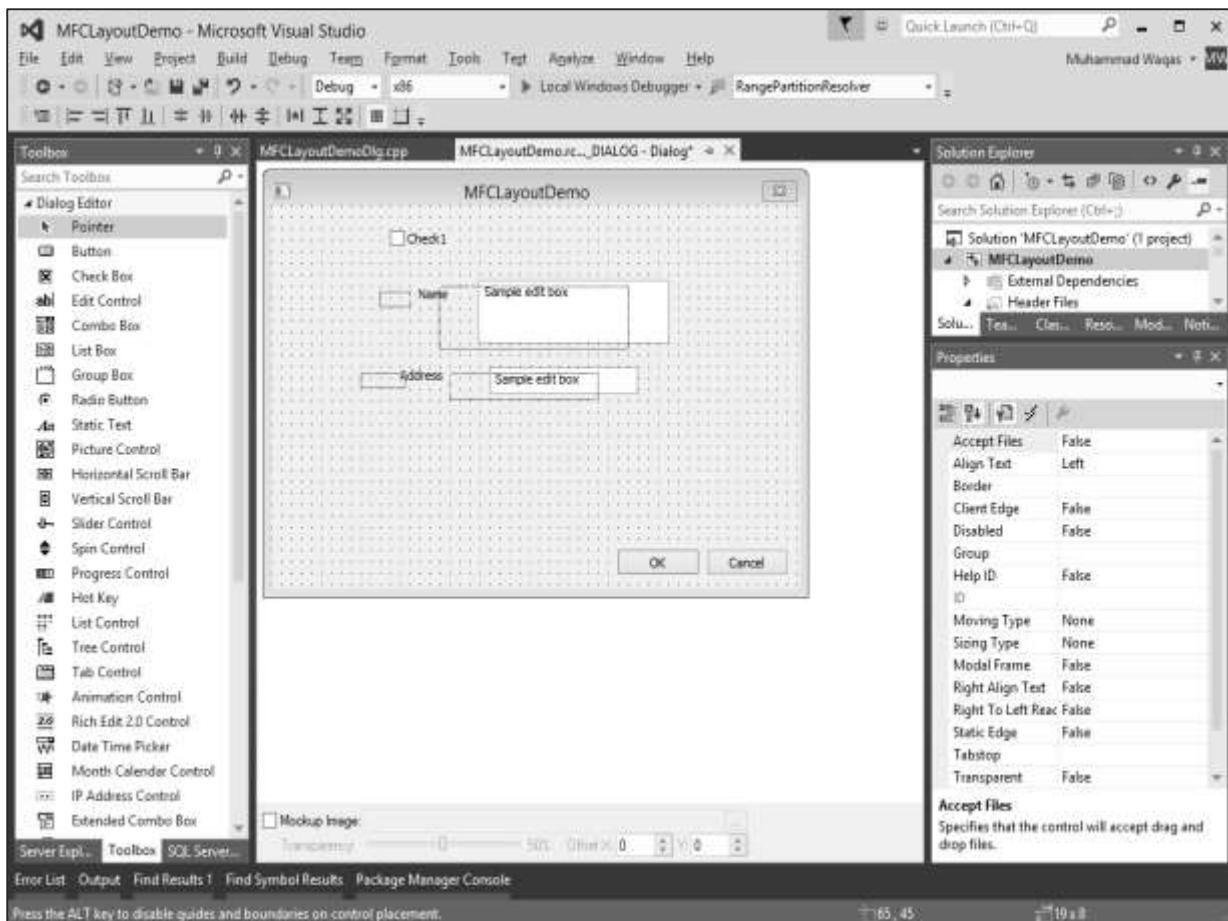


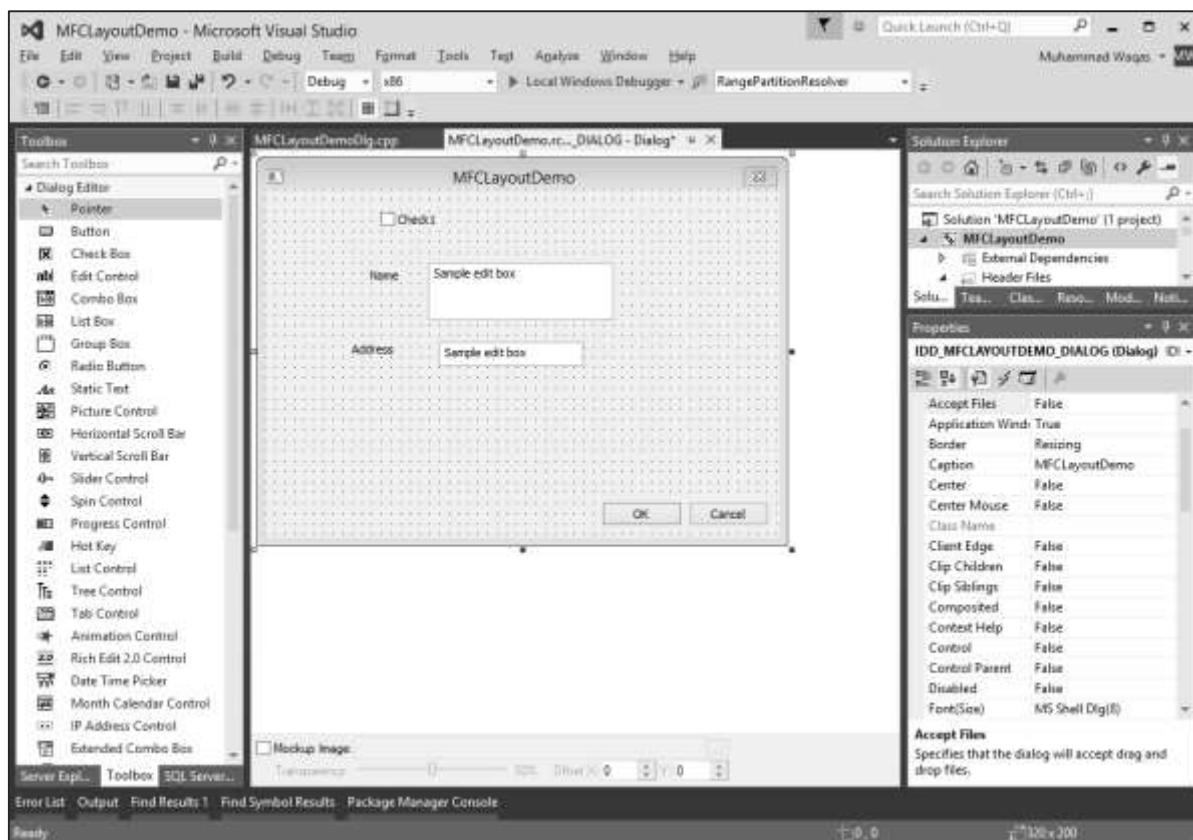
**Step 1:** To move a control, click and drag it in the desired direction until it reaches the intended position.

**Step 2:** To move a group of controls, first select them. Then drag the selection to the desired location. Let us select the Static Texts and Edit Controls.

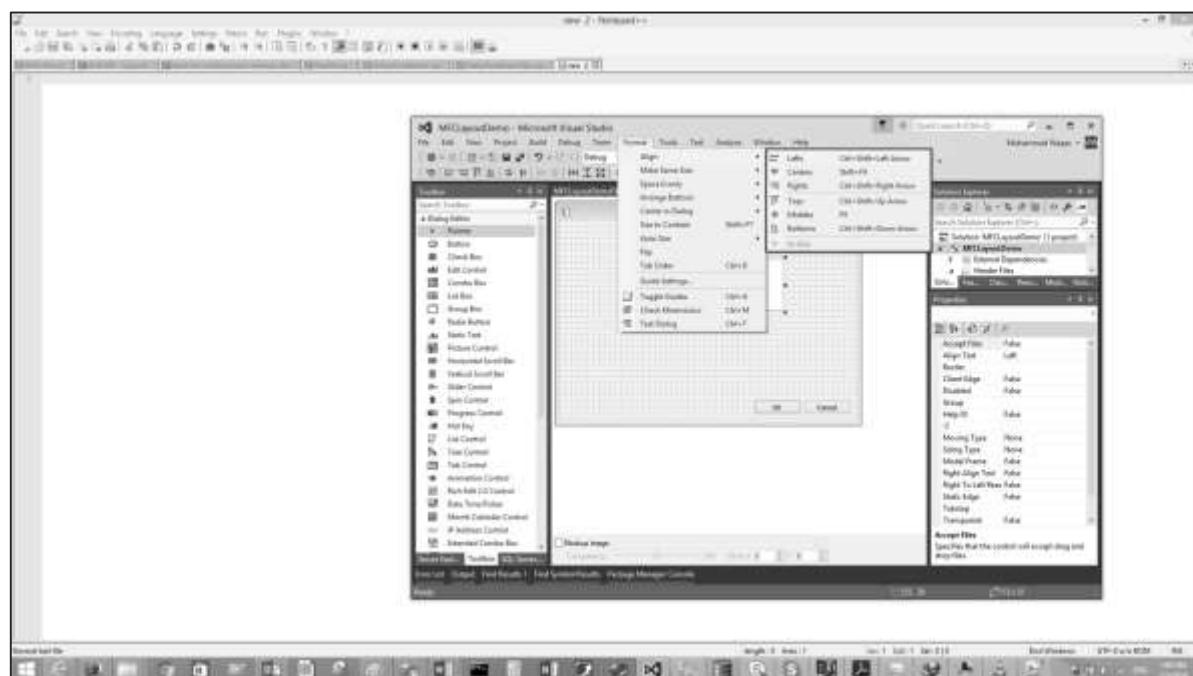


**Step 3:** Move these selected controls to the left side.

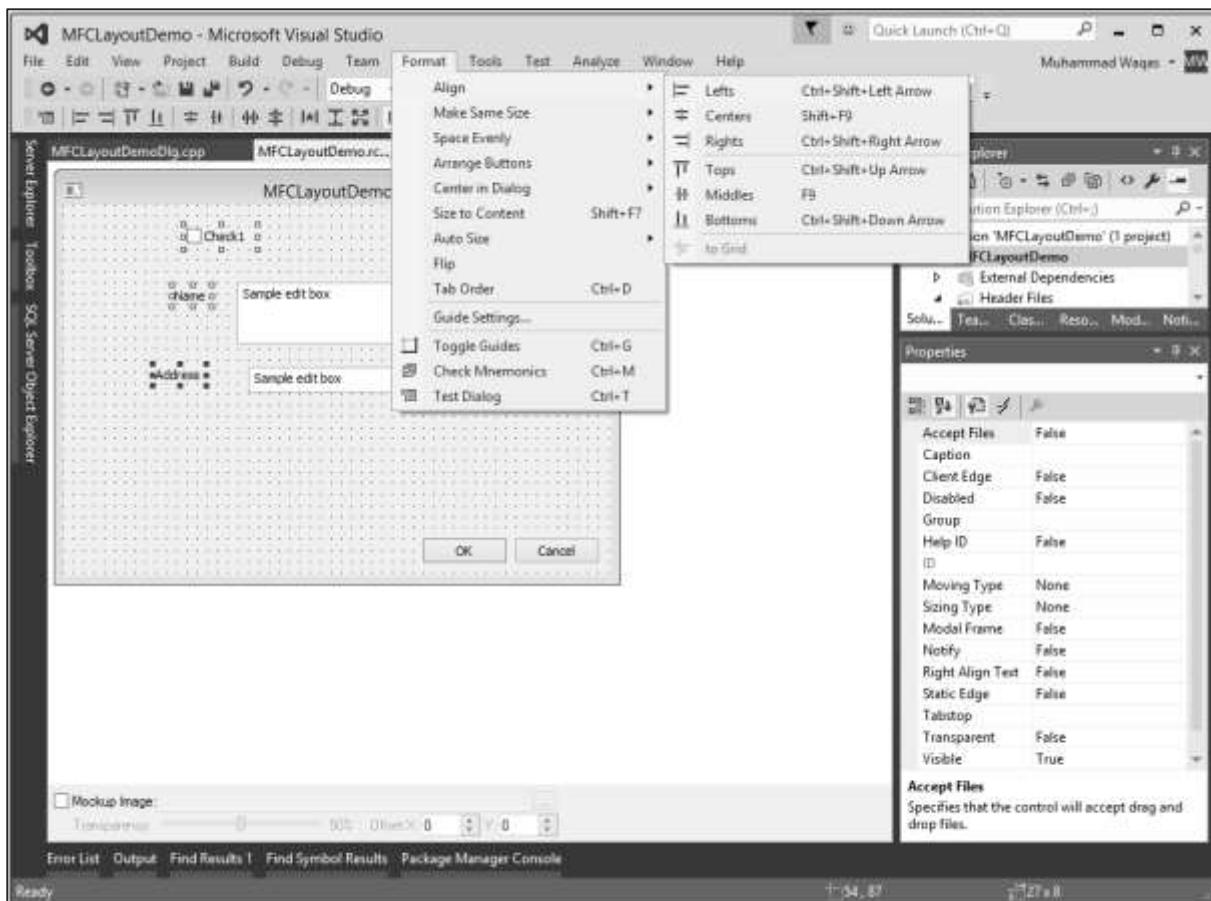




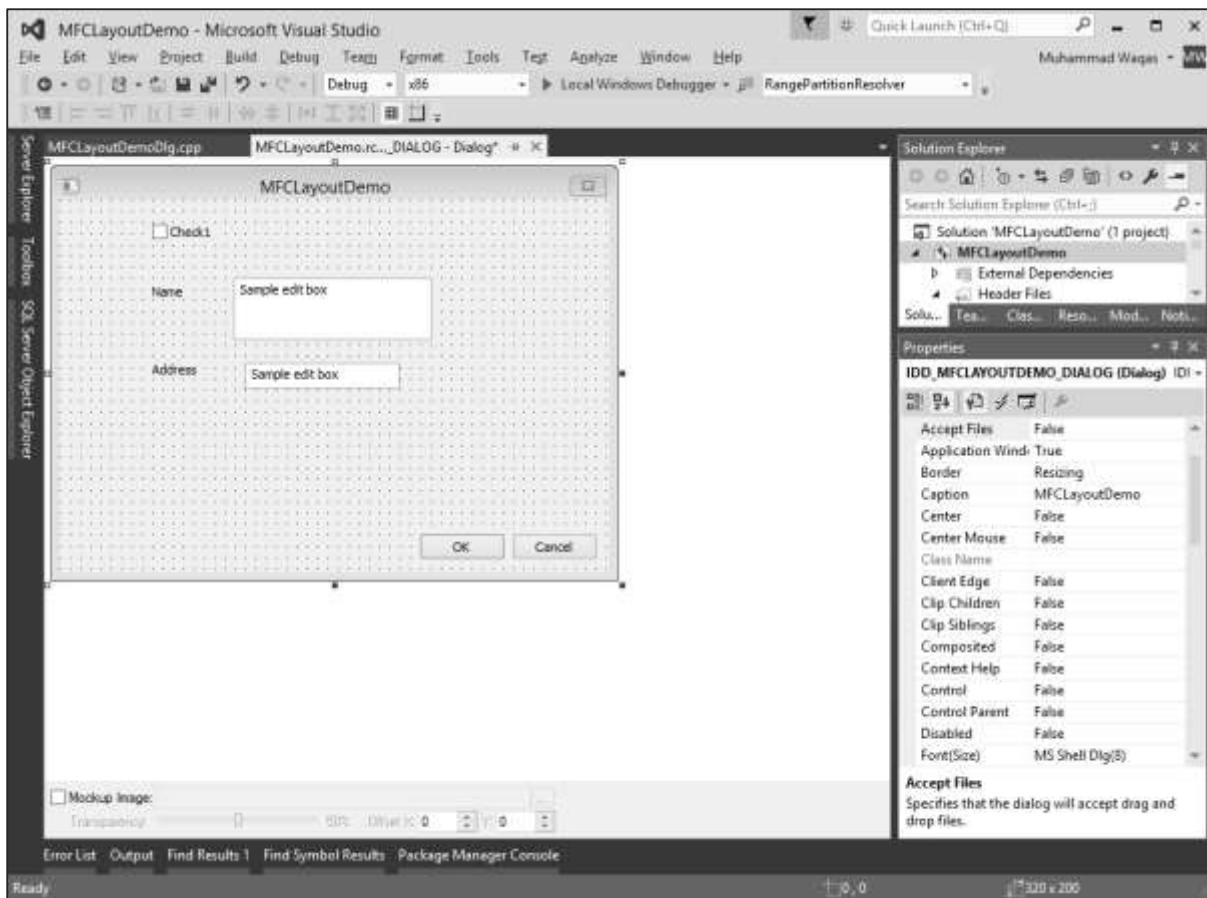
To help with positioning the controls, Visual Studio provides the Dialog toolbar with the following buttons.



**Step 1:** Let us align the Check box and Static Text controls to the left by selecting all these controls.



**Step 2:** Select the Format -> Align -> Lefts.

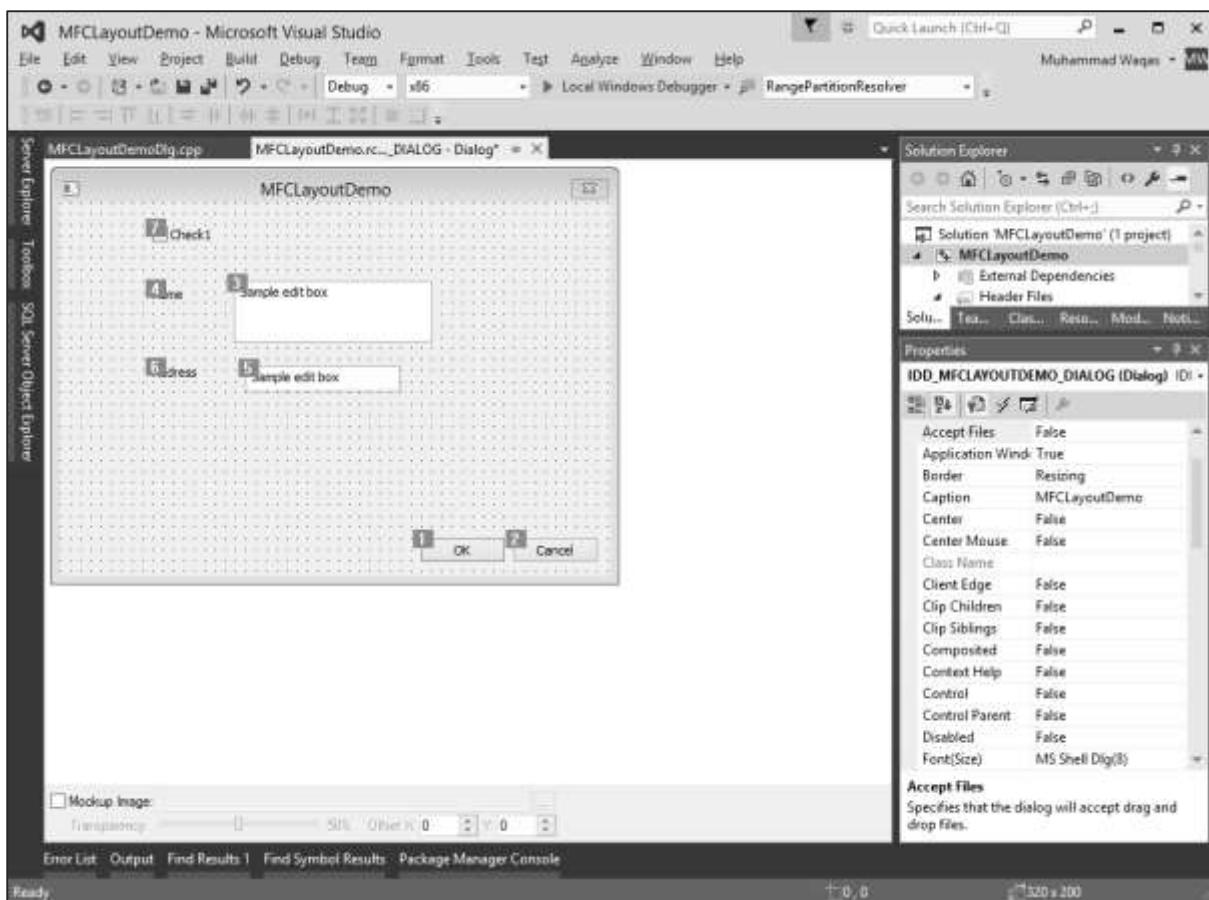


**Step 3:** You can now see all these controls are aligned to the left.

## Tab Ordering

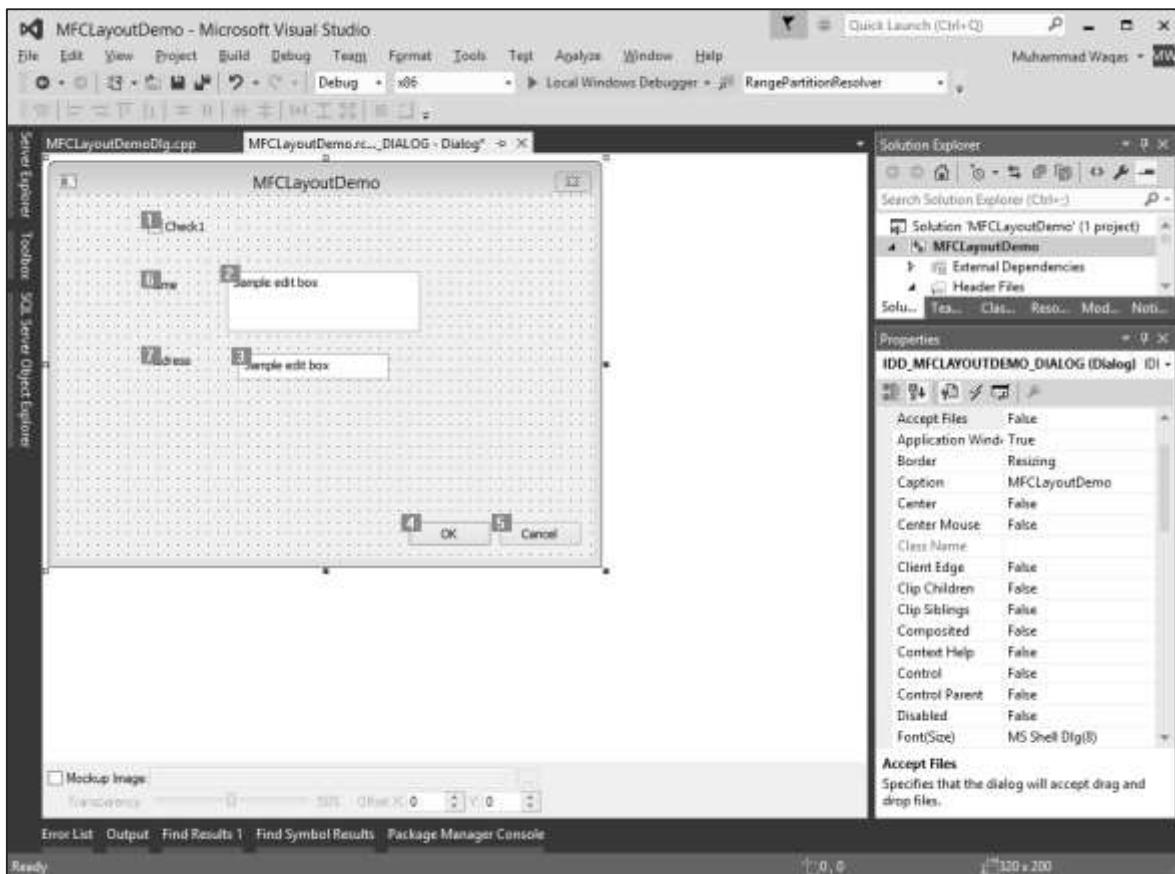
The controls you add to a form or a dialog box are positioned in a sequence that follows the order they were added. When you add control(s) regardless of the section or area you place the new control, it is sequentially positioned at the end of the existing controls. If you do not fix it, the user would have a hard time navigating the controls. The sequence of controls navigation is also known as the tab order.

To change the tab, you can either use the Format -> Tab Order menu option or you can also use the Ctrl + D shortcut. Let us press Ctrl + D.

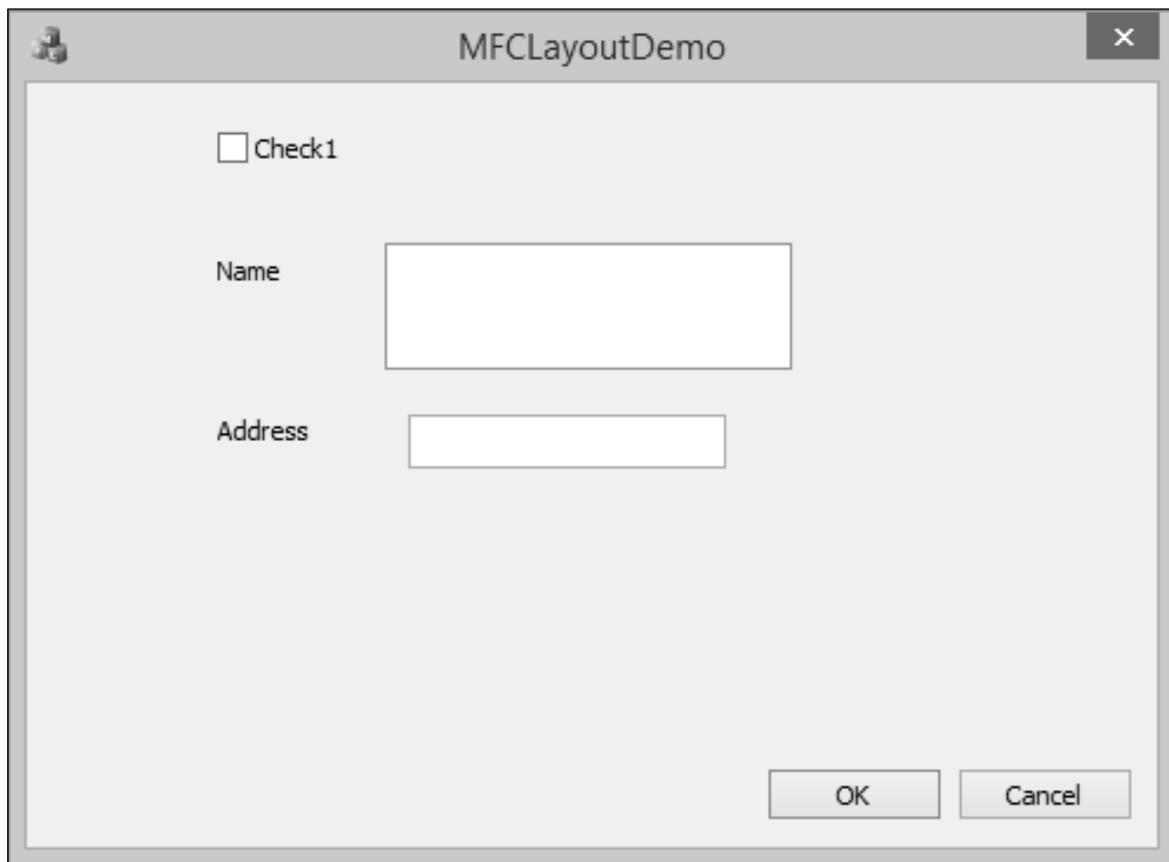


You can now see the order in which all these controls are added to this dialog box. To Change the order or sequence of controls, click on all the controls in sequence in which you want to navigate.

In this example, we will first click on the checkbox followed by Name and Address Edit controls. Then click OK and Cancel as shown in the following snapshot.



Let us run this application and you will see the following output.



# 10. MFC - Controls Management

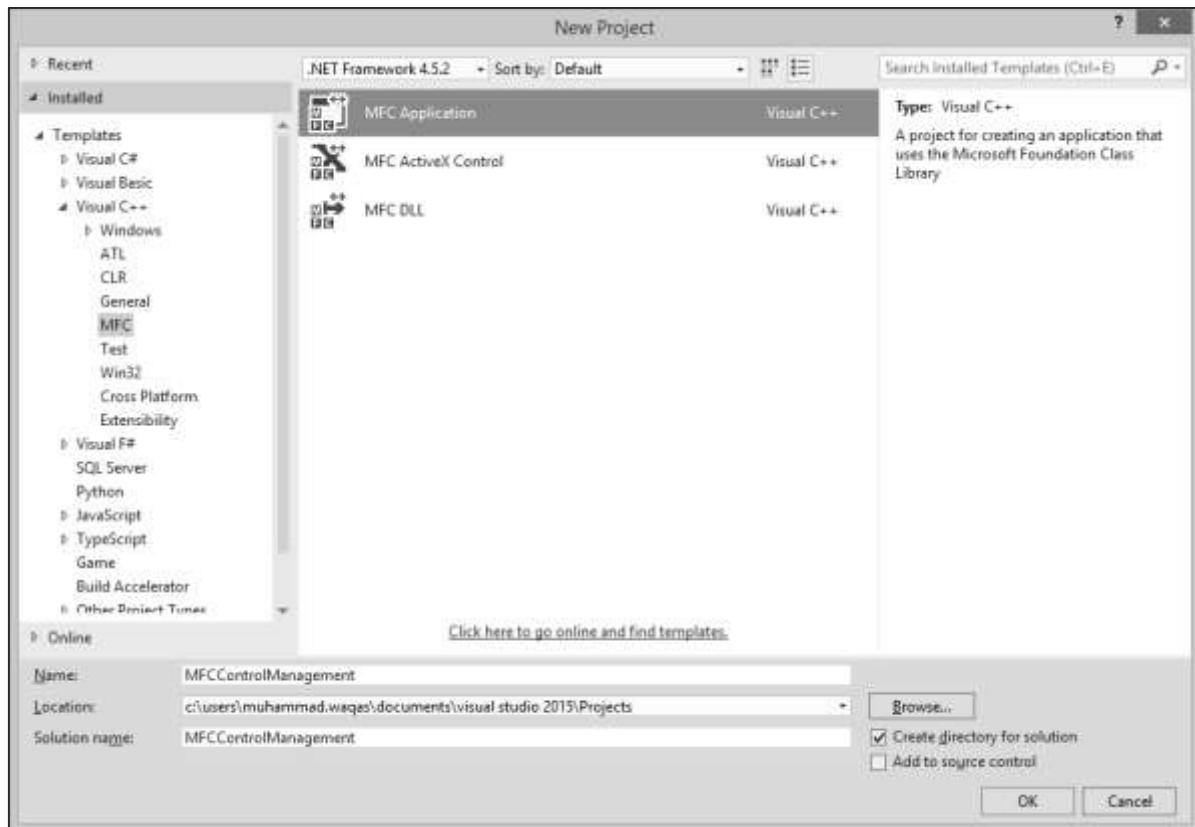
In MFC applications, after visually adding a control to your application, if you want to refer to it in your code, you can declare a variable based on, or associated with that control. The MFC library allows you to declare two types of variables for some of the controls used in an application a value or a control variable.

- One variable is used for the information stored in the control, which is also known as **Control Variable/Instance**.
- The other variable is known as **Control Value Variable**. A user can perform some sort of actions on that control with this variable.

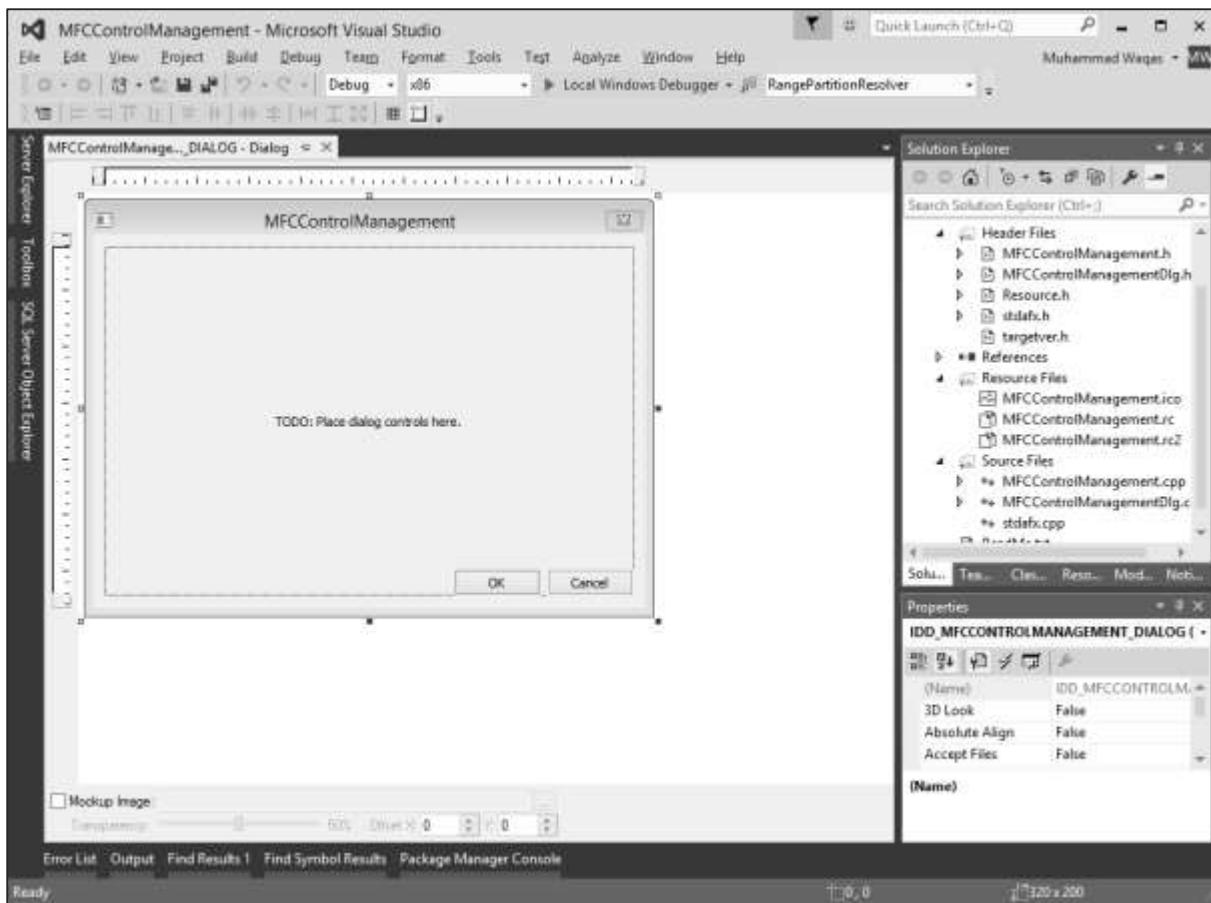
## Control Variable/Instance

A control variable is a variable based on the class that manages the control. For example, a button control is based on the CButton class.

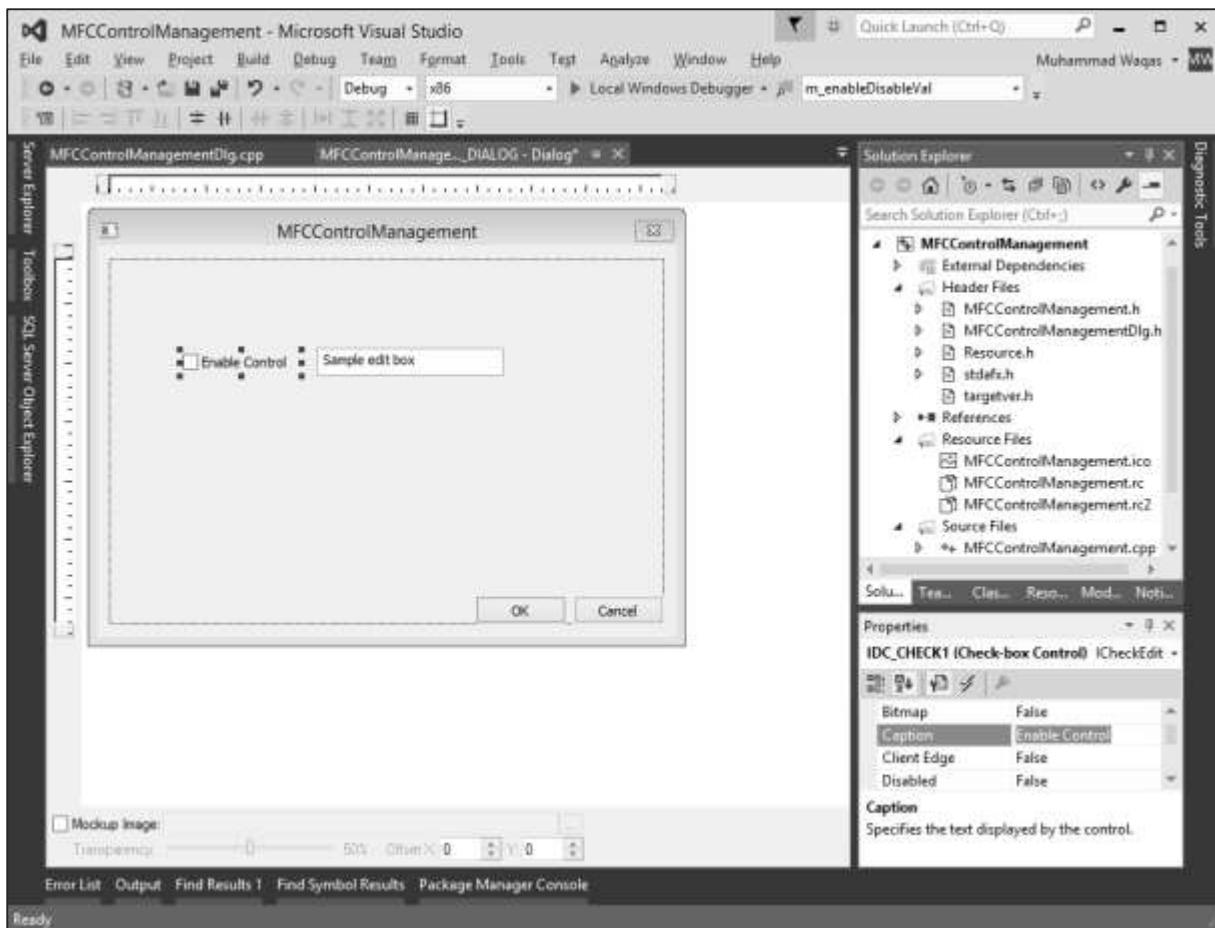
To see these concepts in real programming, let us create an MFC dialog based project MFCControlManagement.



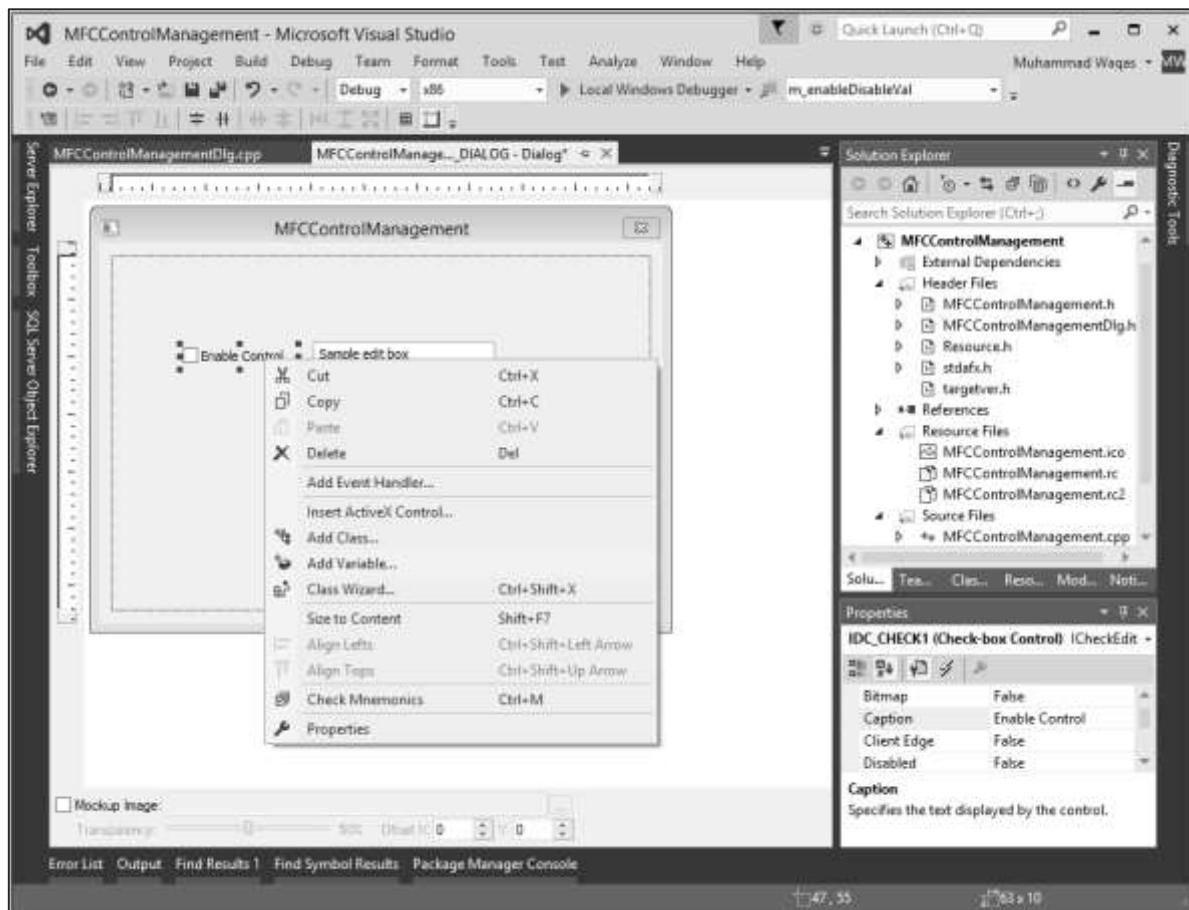
Once the project is created, you will see the following dialog box in designer window.



**Step 1:** Delete the TODO line and drag one checkbox and one Edit control as shown in the following snapshot. Change the caption of checkbox to Enable Control.

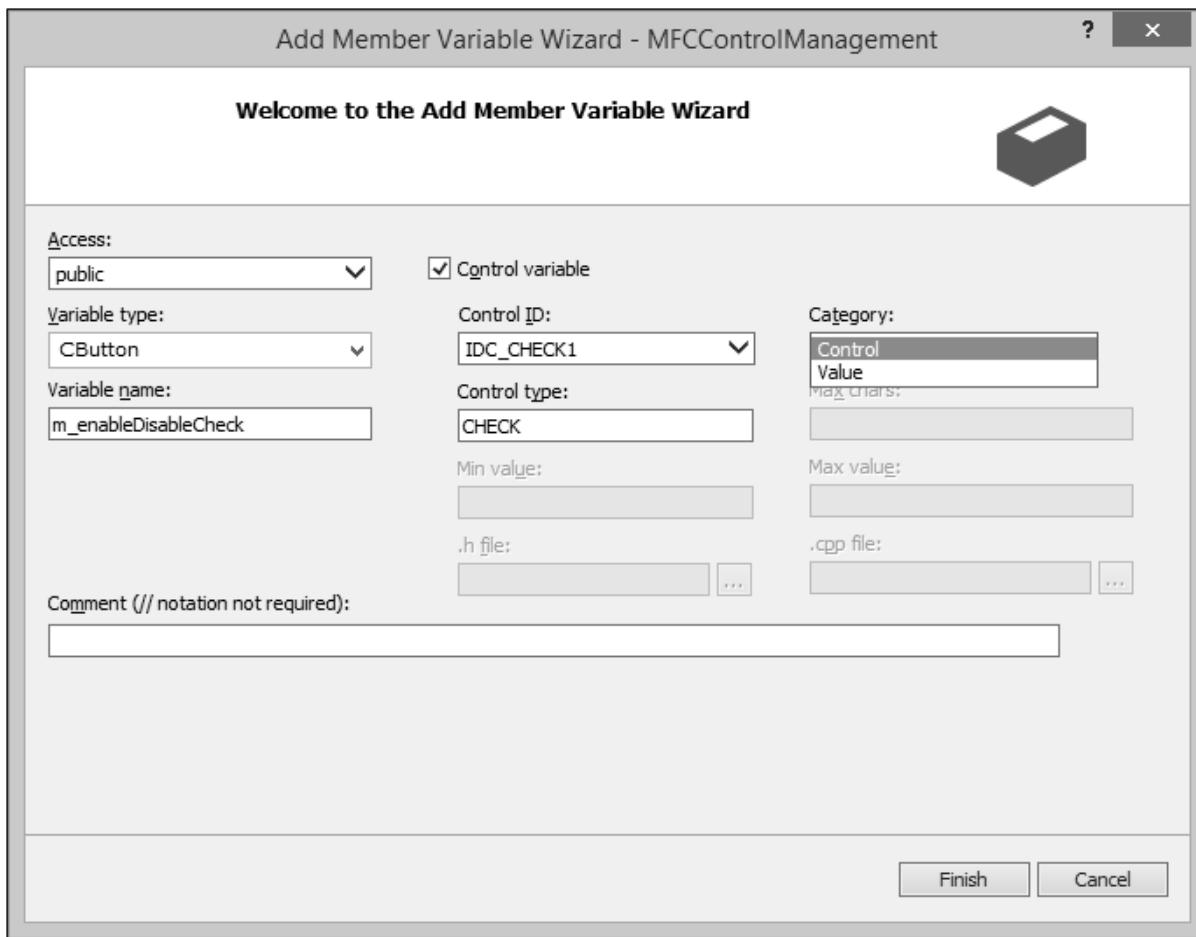


**Step 2:** Right-click on the checkbox.



**Step 3:** Select Add Variable.

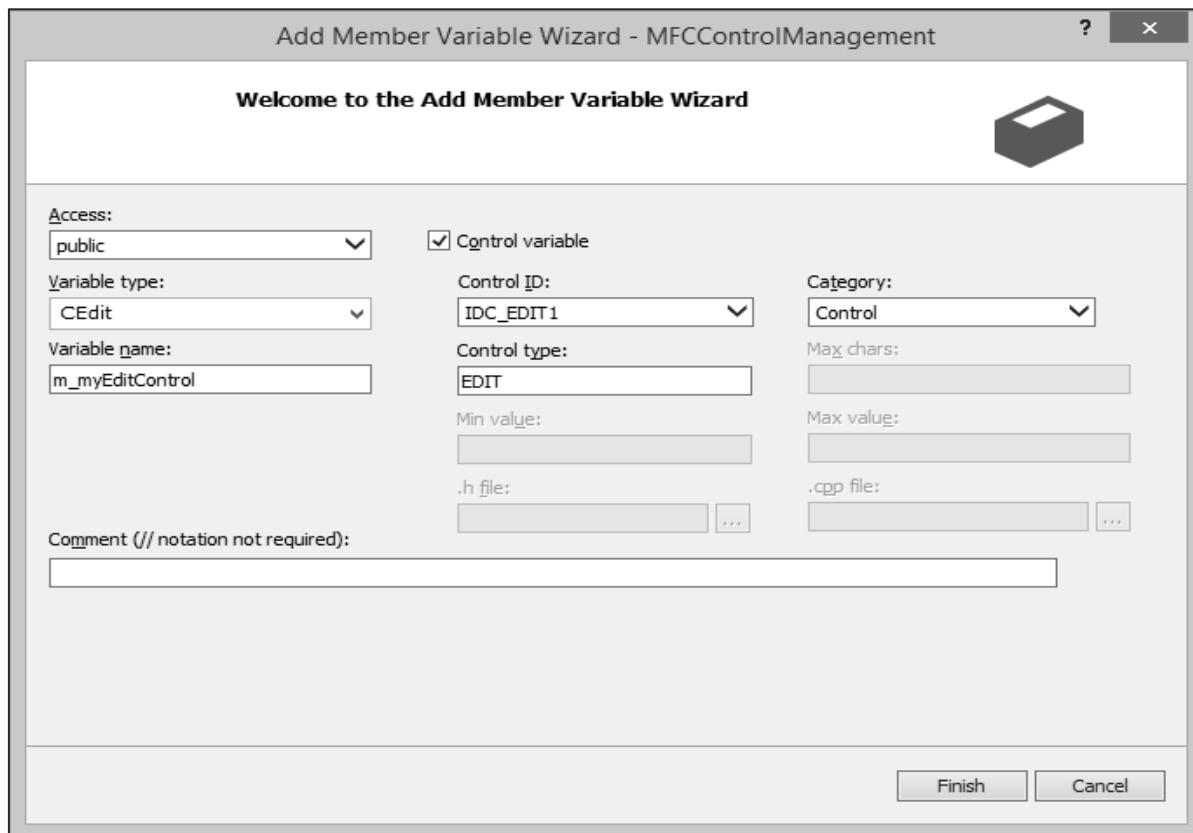
**Step 4:** You can now see the Add Member Variable Wizard.



You can select different options on this dialog box. For checkbox, the variable type is CButton. It is selected by default in this dialog box.

Similarly, the control ID is also selected by default now we need to select Control in the Category combo box, and type m\_enableDisableCheck in the Variable Name edit box and click finish.

**Step 5:** Similarly, add Control Variable of Edit control with the settings as shown in the following snapshot.



Observe the header file of the dialog class. You can see that the new variables have been added now.

```
CButton m_enableDisableCheck;
CEdit m_myEditControl;
```

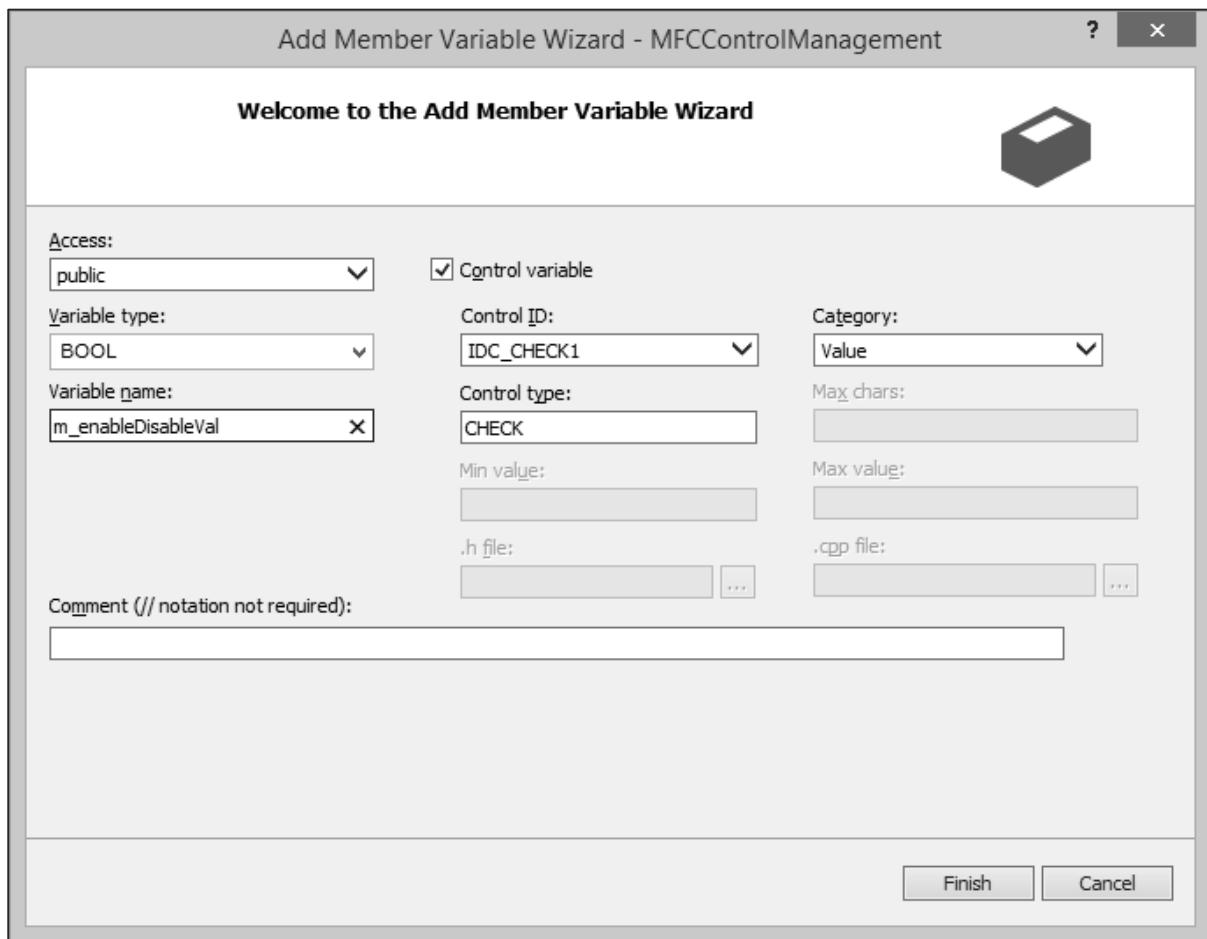
## Control Value Variable

Another type of variable you can declare for a control is the value variable. Not all controls provide a value variable.

- The value variable must be able to handle the type of value stored in the control it is intended to refer to.
- For example, because a text based control is used to handle text, you can declare a text-based data type for it. This would usually be a `CString` variable.

Let us look into this type of variable for checkbox and edit control.

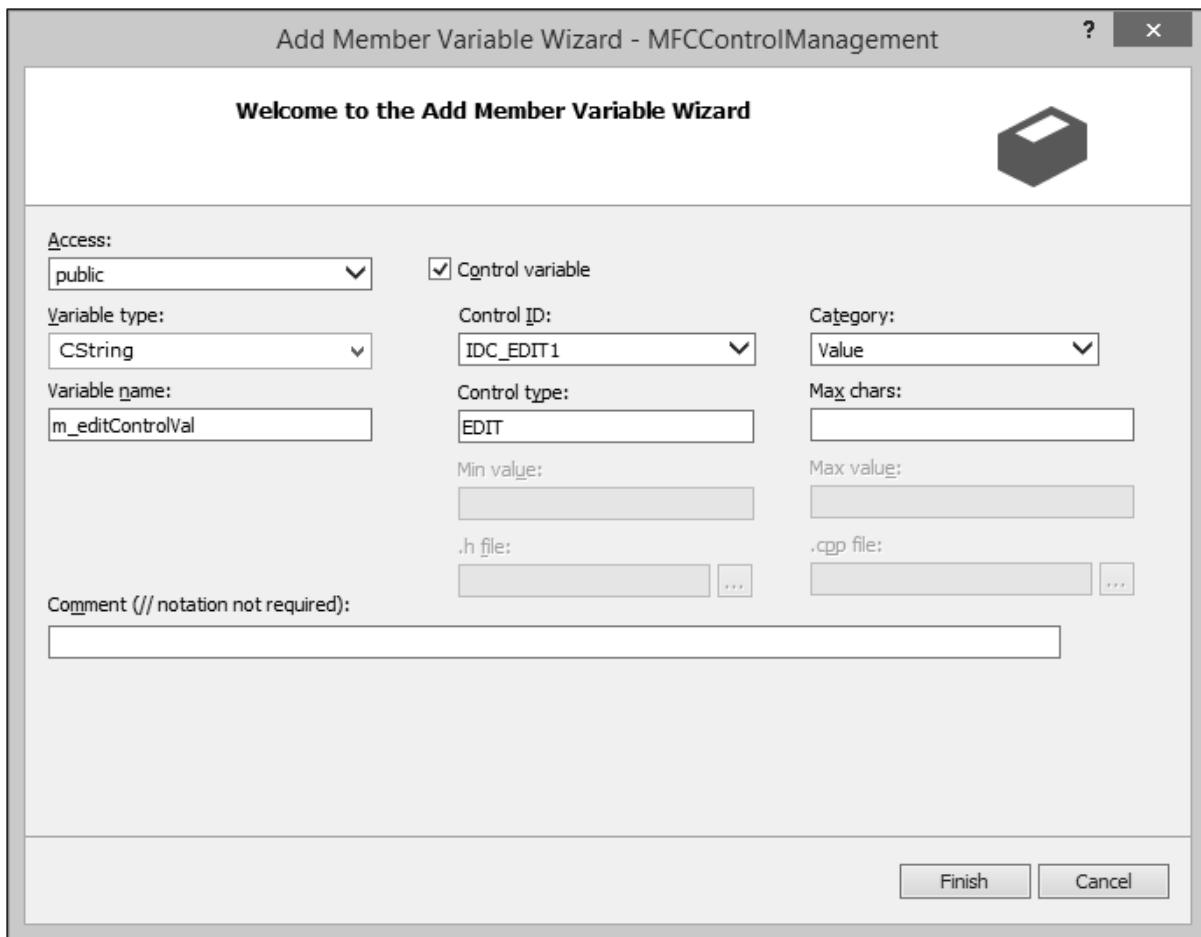
**Step 1:** Right-click on the checkbox and select Add Variable.



**Step 2:** The Variable type is BOOL. Select Value from the Category dropdown list.

**Step 3:** Click Finish to continue.

**Step 4:** Similarly, add value Variable for Edit control with the settings as shown in the following snapshot.



**Step 5:** Type CString in variable type and m\_editControlVal in the variable name field.

**Step 6:** You can now see these variables added in the Header file.

```
bool m_enableDisableVal;
CString m_editControlVal;
```

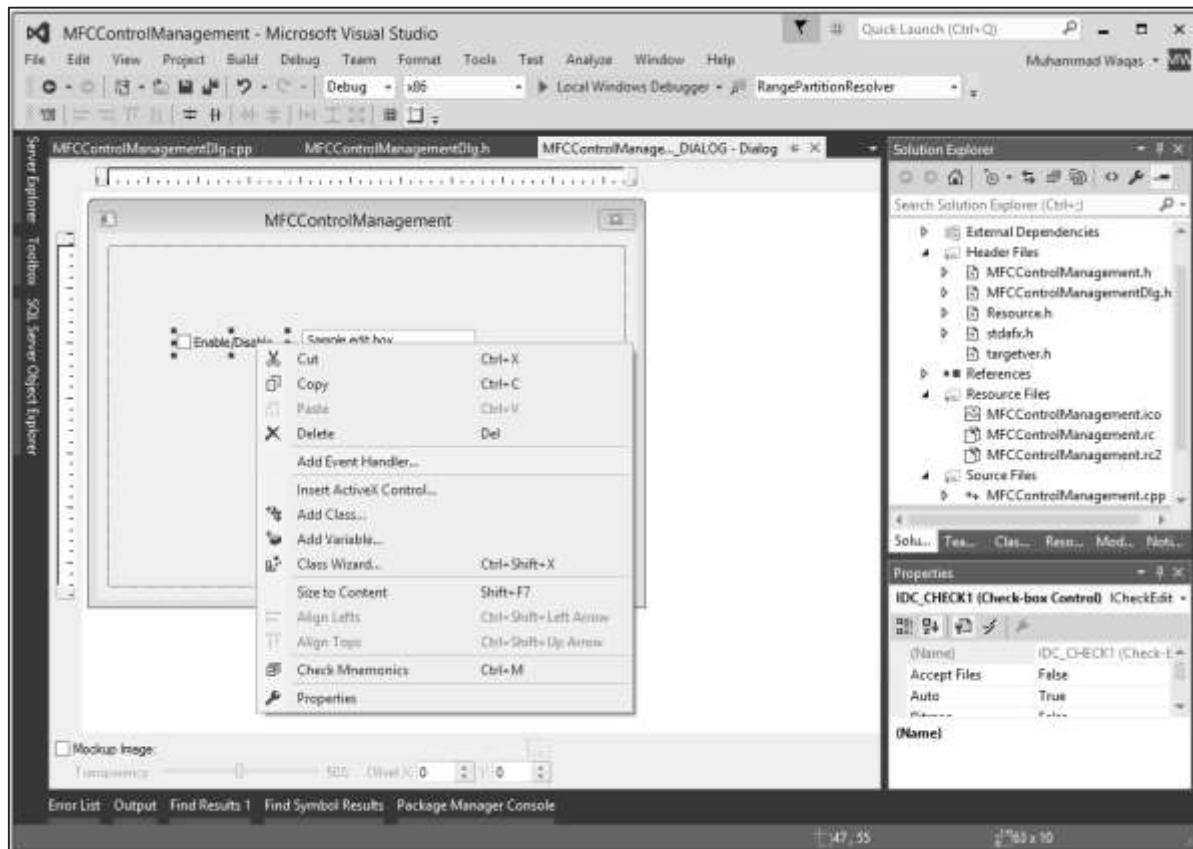
## Controls Event Handlers

After adding a control to your application, whether you visually added it or created it dynamically, you will also decide how to handle the possible actions that the user can perform on the control.

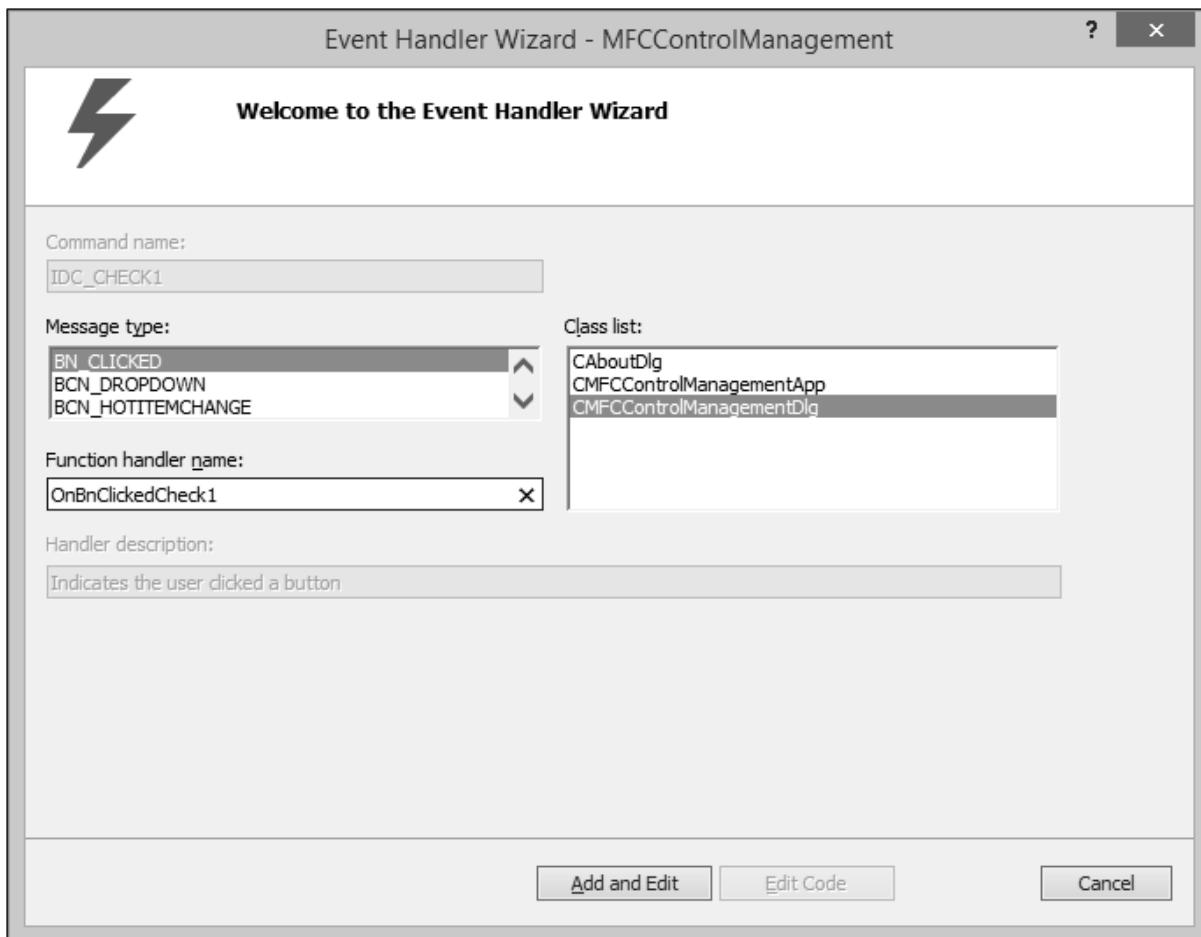
- For project dialog boxes that are already associated with a class, you can take advantage of some shortcuts when you create event handlers.
- You can quickly create a handler either for the default control notification event or for any applicable Windows message.

Let us look into the same example in which we added event handler for checkbox.

**Step 1:** Right-click the control for which you want to handle the notification event.



**Step 2:** On the shortcut menu, click Add Event Handler to display the Event Handler Wizard.



**Step 3:** Select the event in the Message type box to add to the class selected in the Class list box.

**Step 4:** Accept the default name in the Function handler name box, or provide the name of your choice.

**Step 5:** Click Add and edit to add the event handler.

**Step 6:** You can now see the following event added at the end of CMFCCControlManagementDlg.cpp file.

```
void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
}
```

## Controls Management

---

So far, we have seen how to add controls to an application. We will now see how to manage these controls as per user requirement. We can use the control variable/instance in a particular event handler.

**Step 1:** Let us look into the following example. Here, we will enable/disable the edit control when the checkbox is checked/unchecked.

**Step 2:** We have now added the checkbox click event handler. Here is the implementation:

```
void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    if (m_enableDisableVal)
        m_myEditControl.EnableWindow(TRUE);
    else
        m_myEditControl.EnableWindow(FALSE);
}
```

**Step 3:** When the dialog is created, we need to add the following code to CMFCCControlManagementDlg::OnInitDialog(). This will manage these controls.

```
UpdateData(TRUE);
if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
else
    m_myEditControl.EnableWindow(FALSE);
```

**Step 4:** Here is the complete implementation of CMFCCControlManagementDlg.cpp file.

```
// MFCCControlManagementDlg.cpp : implementation file
//

#include "stdafx.h"
#include "MFCCControlManagement.h"
#include "MFCCControlManagementDlg.h"
#include "afxdialogex.h"

#ifndef _DEBUG
```

```
#define new DEBUG_NEW
#endif

// CAboutDlg dialog used for App About

class CAboutDlg : public CDlgEx
{
public:
    CAboutDlg();

// Dialog Data
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_ABOUTBOX };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

// Implementation

protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDlgEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDlgEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDlgEx)
END_MESSAGE_MAP()
```

```

// CMFCCControlManagementDlg dialog

CMFCCControlManagementDlg::CMFCCControlManagementDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(IDD_MFCCONTROLMANAGEMENT_DIALOG, pParent)
    , m_enableDisableVal(FALSE)
    , m_editControlVal(_T(""))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMFCCControlManagementDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_CHECK1, m_enableDisableCheck);
    DDX_Control(pDX, IDC_EDIT1, m_myEditControl);
    DDX_Check(pDX, IDC_CHECK1, m_enableDisableVal);
    DDX_Text(pDX, IDC_EDIT1, m_editControlVal);
}

BEGIN_MESSAGE_MAP(CMFCCControlManagementDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_CHECK1, &CMFCCControlManagementDlg::OnBnClickedCheck1)
END_MESSAGE_MAP()

// CMFCCControlManagementDlg message handlers

BOOL CMFCCControlManagementDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.
}

```

```

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMen* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    BOOL bNameValid;
    CString strAboutMenu;
    bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
    ASSERT(bNameValid);
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

// TODO: Add extra initialization here
UpdateData(TRUE);
if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
else
    m_myEditControl.EnableWindow(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}

void CMFCControlManagementDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
    {
}

```

```

        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialogEx::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CMFCCControlManagementDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

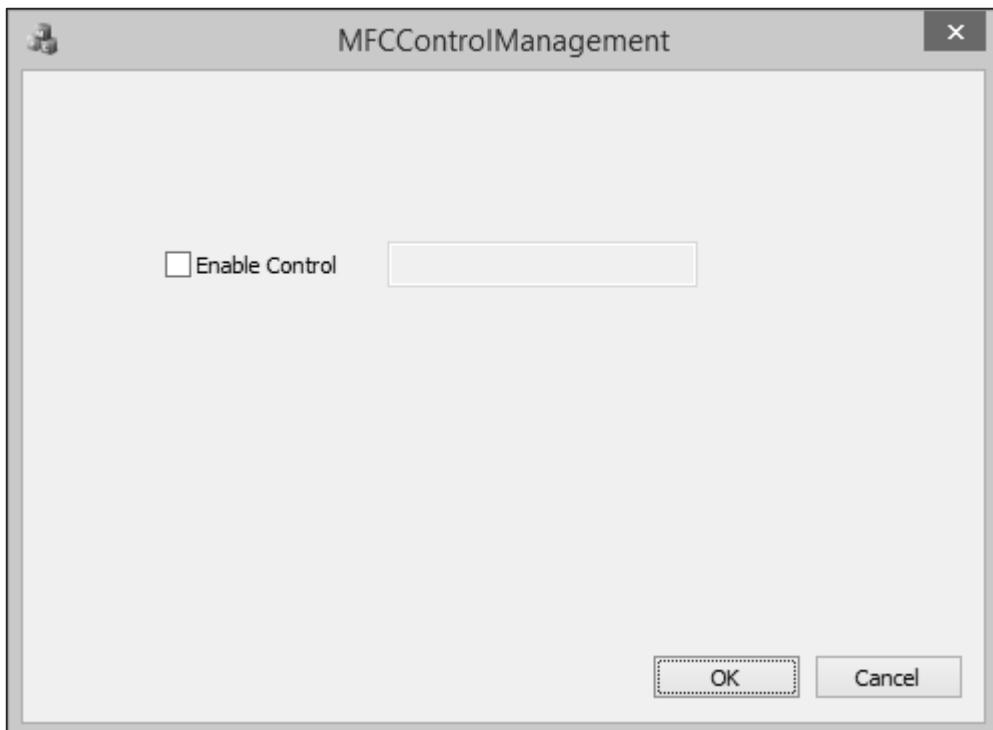
        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialogEx::OnPaint();
    }
}

```

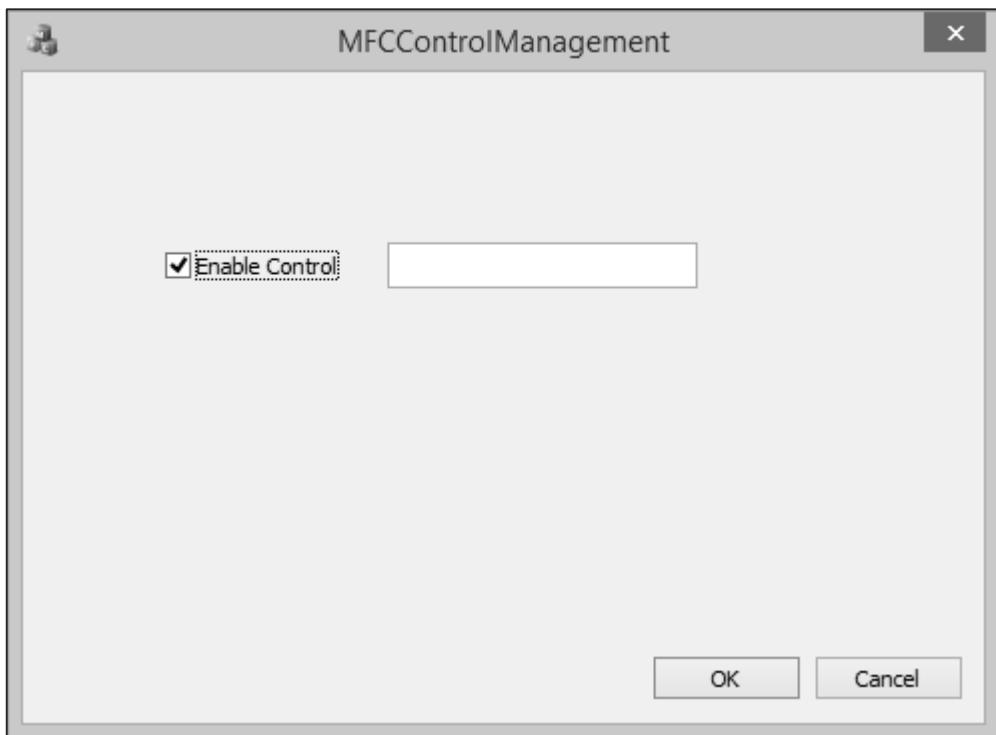
```
// The system calls this function to obtain the cursor to display while the
// user drags
// the minimized window.
HCURSOR CMFCCControlManagementDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    if (m_enableDisableVal)
        m_myEditControl.EnableWindow(TRUE);
    else
        m_myEditControl.EnableWindow(FALSE);
}
```

**Step 5:** When the above code is compiled and executed, you will see the following output. The checkbox is unchecked by default. This disables the edit control too.



**Step 6:** Check the Enable Control checkbox. This will automatically enable the edit control.



# 11. MFC - Windows Controls

**Windows controls** are objects that users can interact with to enter or manipulate data. They commonly appear in dialog boxes or on toolbars. There are various types of controls:

- A **text based control** which is used to display text to the user or request text from the user.
- A **list based control** displays a list of items.
- A **progress based control** is used to show the progress of an action.
- A **static control** can be used to show colors, a picture or something that does not regularly fit in the above categories.

## Static Control

---

A static control is an object that displays information to the user without his or her direct intervention. It can be used to show colors, a geometric shape, or a picture such as an icon, a bitmap, or an animation. Following are the static controls:

- Static Text
- Picture Control
- Group Box

## Animation Control

---

An animation control is a window that displays an Audio clip in AVI format. An AVI clip is a series of bitmap frames, like a movie. Animation controls can only play simple AVI clips, and they do not support sound. It is represented by the **CAnimateCtrl** class.

Following is the list of methods in CAnimateCtrl class:

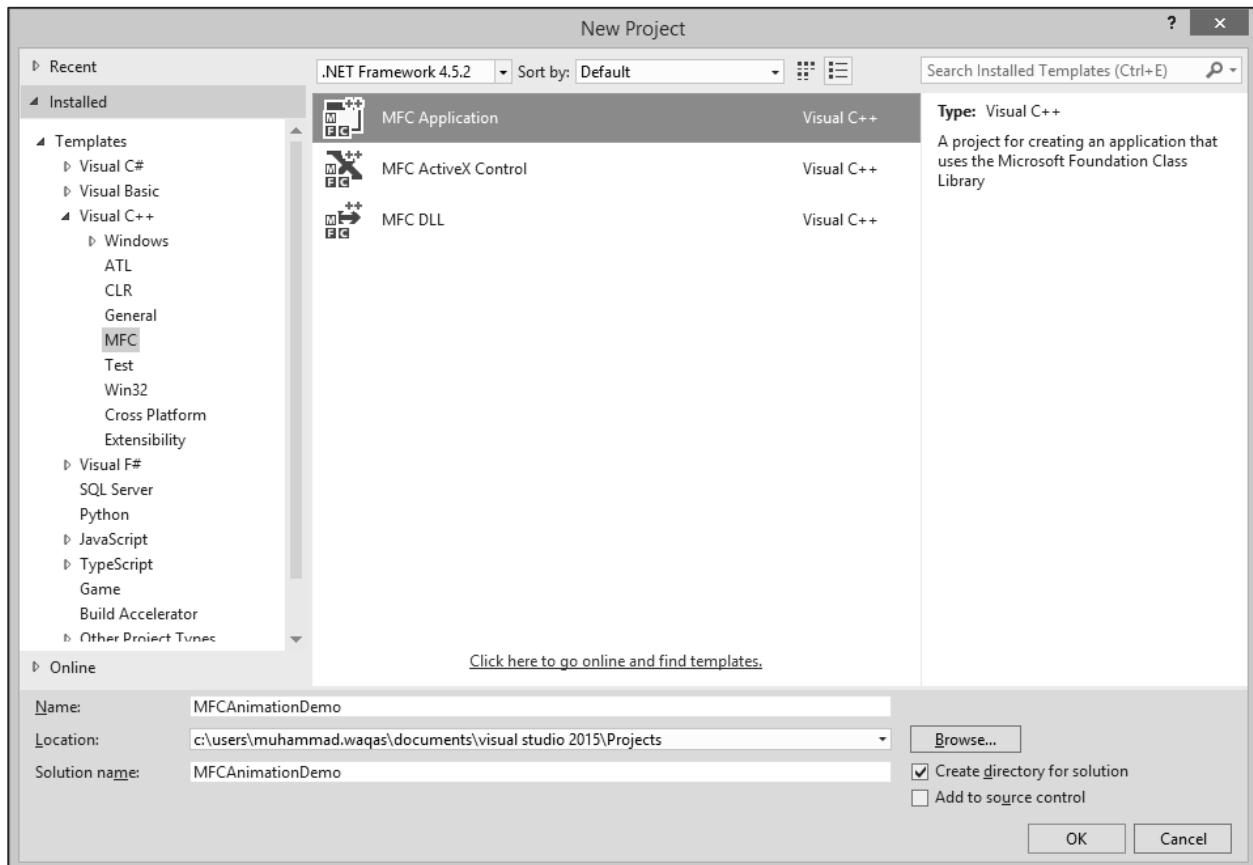
Name	Description
<b>Close</b>	Closes the AVI clip.
<b>Create</b>	Creates an animation control and attaches it to a CAnimateCtrl object.
<b>CreateEx</b>	Creates an animation control with the specified Windows extended styles and attaches it to a CAnimateCtrl object.
<b>IsPlaying</b>	Indicates whether an Audio-Video Interleaved (AVI) clip is playing.
<b>Open</b>	Opens an AVI clip from a file or resource and displays the first frame.
<b>Play</b>	Plays the AVI clip without sound.
<b>Seek</b>	Displays a selected single frame of the AVI clip.
<b>Stop</b>	Stops playing the AVI clip.

Here is the list of messages mapping for animation control:

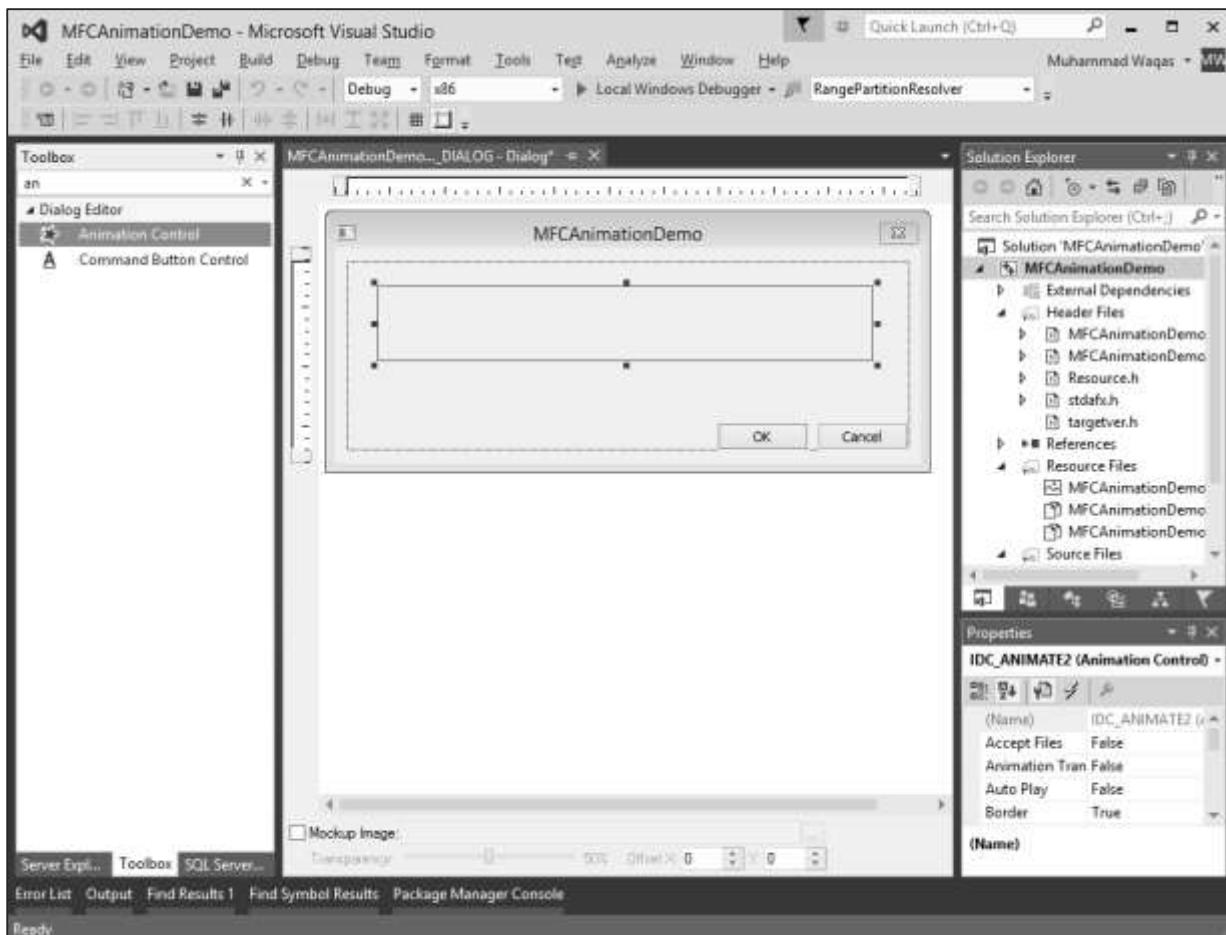
Message	Map entry	Description
<b>ACN_START</b>	ON_ACN_START ( <id>, <memberFxn> )	The framework calls this member function when an animation is being started
<b>ACN_STOP</b>	ON_ACN_STOP ( <id>, <memberFxn> )	The framework calls this member function when an animation stop

Let us look into a simple example of animation control.

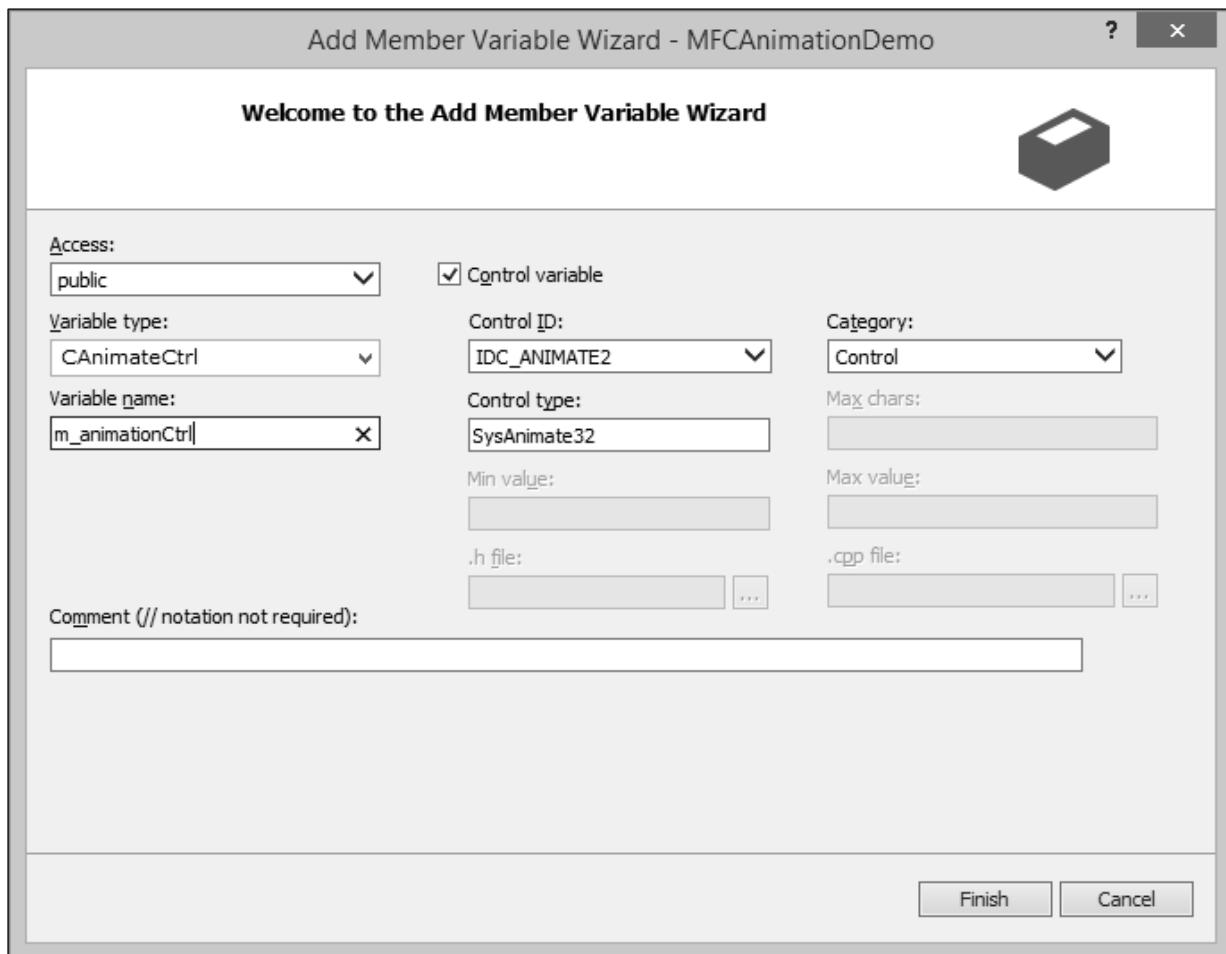
**Step 1:** Create a new MFC dialog based project.



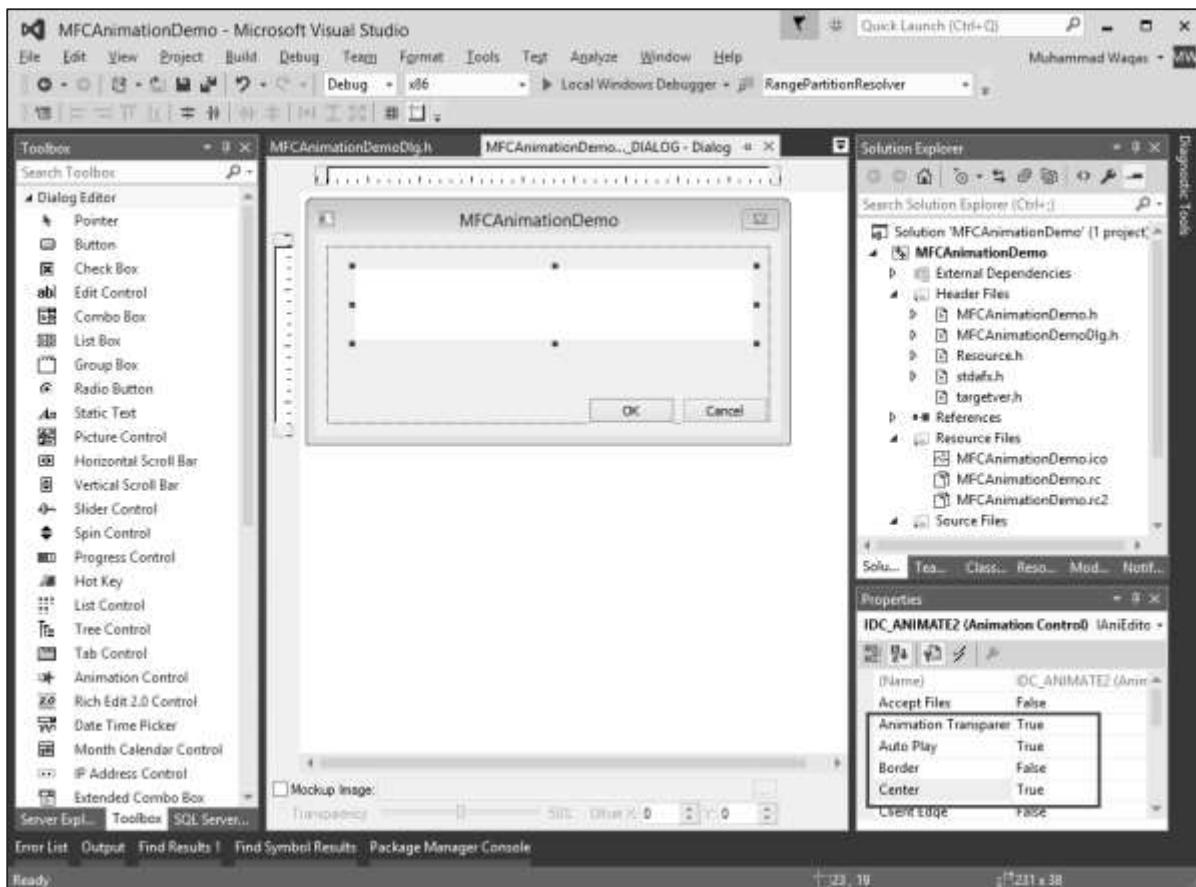
**Step 2:** Once the project is created, remove the TODO line and click on Animation Control in Toolbox and draw a rectangle as shown in the following snapshot.



**Step 3:** To add a control variable for animation control, right-click and select Add Variable.



**Step 4:** Enter the variable name and variable type, which is CAnimateCtrl for animation.



**Step 5:** Using the Properties window, set the Border value to False, Set the Auto Play, the Center, and the transparent values to True.

**Step 6:** Here we have \*.avi file in **res** folder, which is the default folder for any resources used in the project

**Step 7:** To start the animation, we need to call the Open method **CAnimateCtrl** class. Add the following line of code in CMFCAnimationDemoDlg::OnInitDialog()

```
m_animationCtrl.Open(L"res\\copyfile.avi");
```

**Step 8:** Here is the complete implementation of CMFCAnimationDemoDlg::OnInitDialog()

```
BOOL CMFCAnimationDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
```

```
ASSERT(IDM_ABOUTBOX < 0xF000);

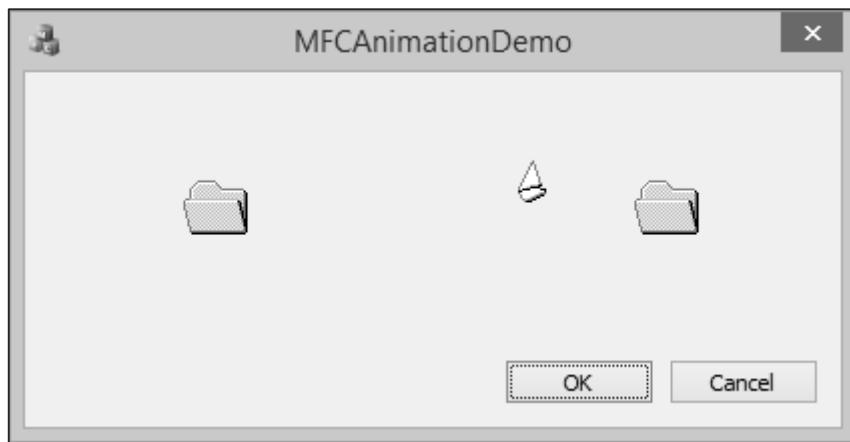
CMenus* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
    BOOL bNameValid;
    CString strAboutMenu;
    bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
    ASSERT(bNameValid);
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);          // Set small icon

// TODO: Add extra initialization here
m_animationCtrl.Open(L"res\\copyfile.avi");

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 9:** When the above code is compiled and executed, you will see the following output.



## Button

A **button** is an object that the user clicks to initiate an action. Button control is represented by **CButton class**.

Here is the list of methods in CButton class:

Name	Description
<b>Create</b>	Creates the Windows button control and attaches it to the CButton object.
<b>DrawItem</b>	Override to draw an owner-drawn CButton object.
<b>GetBitmap</b>	Retrieves the handle of the bitmap previously set with <a href="#">SetBitmap</a> .
<b>GetButtonStyle</b>	Retrieves information about the button control style.
<b>GetCheck</b>	Retrieves the check state of a button control.
<b>GetCursor</b>	Retrieves the handle of the cursor image previously set with <a href="#">SetCursor</a> . 
<b>GetIcon</b>	Retrieves the handle of the icon previously set with <a href="#">SetIcon</a> .
<b>GetIdealSize</b>	Retrieves the ideal size of the button control.
<b>GetImageList</b>	Retrieves the image list of the button control.
<b>GetNote</b>	Retrieves the note component of the current command link control.
<b>GetNoteLength</b>	Retrieves the length of the note text for the current command link control.
<b>GetSplitGlyph</b>	Retrieves the glyph associated with the current split button control.
<b>GetSplitImageList</b>	Retrieves the image list for the current split button control.
<b>GetSplitInfo</b>	Retrieves information that defines the current split button control.
<b>GetSplitSize</b>	Retrieves the bounding rectangle of the drop-down component of the current split button control.
<b>GetSplitStyle</b>	Retrieves the split button styles that define the current split button control.
<b>GetState</b>	Retrieves the check state, highlight state, and focus state of a button control.
<b>GetTextMargin</b>	Retrieves the text margin of the button control.
<b>SetBitmap</b>	Specifies a bitmap to be displayed on the button.

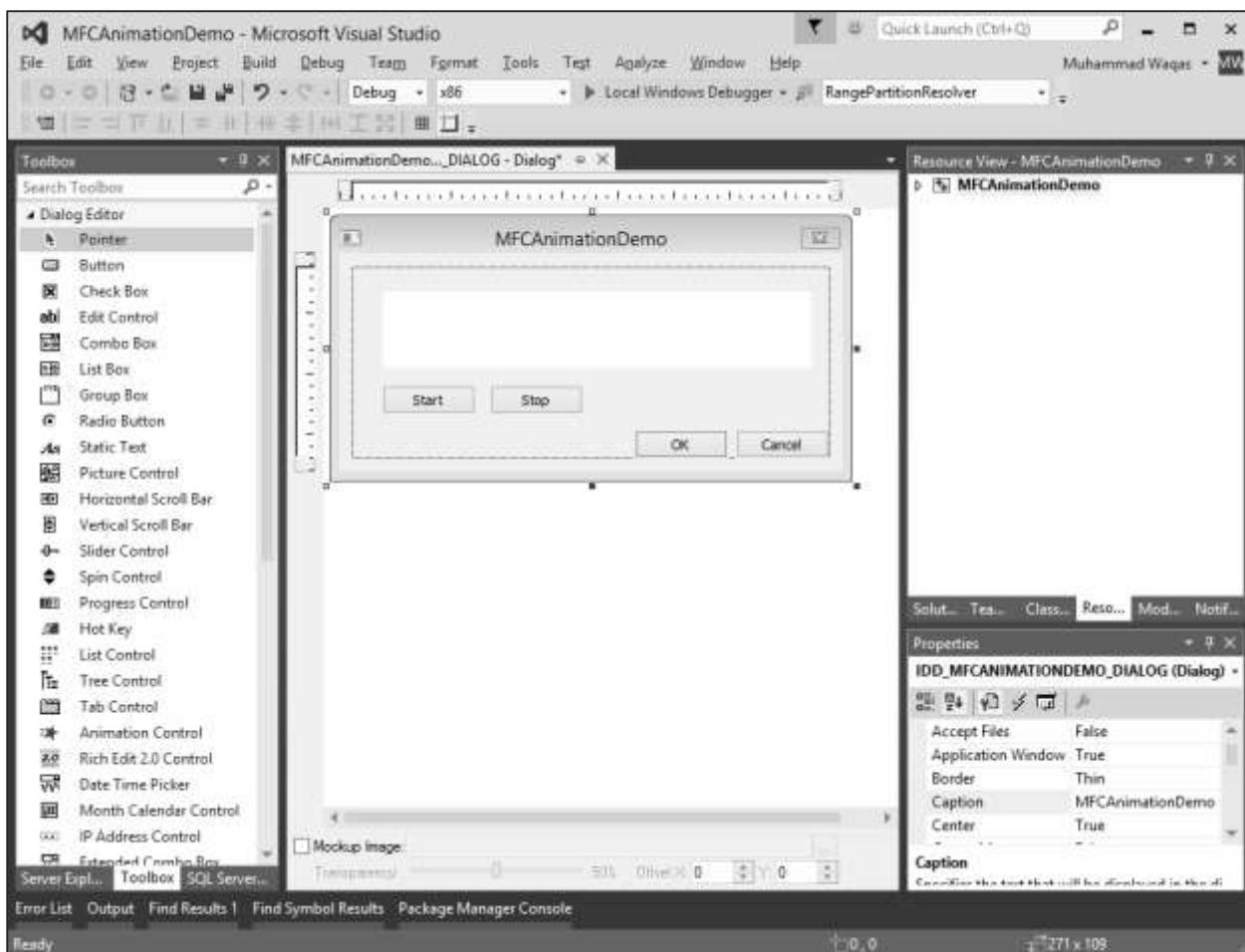
<b>SetButtonStyle</b>	Changes the style of a button.
<b>SetCheck</b>	Sets the check state of a button control.
<b>SetCursor</b>	Specifies a cursor image to be displayed on the button.
<b>SetDropDownState</b>	Sets the drop-down state of the current split button control.
<b>SetIcon</b>	Specifies an icon to be displayed on the button.
<b>SetImageList</b>	Sets the image list of the button control.
<b>SetNote</b>	Sets the note on the current command link control.
<b>SetSplitGlyph</b>	Associates a specified glyph with the current split button control.
<b>SetSplitImageList</b>	Associates an image list with the current split button control.
<b>SetSplitInfo</b>	Specifies information that defines the current split button control.
<b>SetSplitSize</b>	Sets the bounding rectangle of the drop-down component of the current split button control.
<b>SetSplitStyle</b>	Sets the style of the current split button control.
<b>SetState</b>	Sets the highlighting state of a button control.
<b>SetTextMargin</b>	Sets the text margin of the button control.

Here is the list of messages mapping for Button control:

Message	Map entry	Description
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is clicked.
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when the button is disabled.
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is double clicked.
<b>BN_PAINT</b>	ON_BN_PAINT( <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button.

Let us look into a simple example by dragging two buttons from the Toolbox.

**Step 1:** Change the Caption from Start, Stop and ID to IDC\_BUTTON\_START, IDC\_BUTTON\_STOP for both buttons.



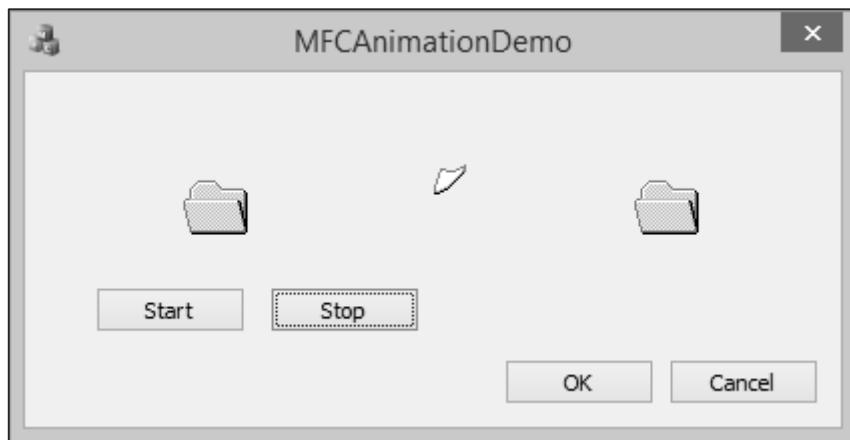
**Step 2:** Let us add event handler for both buttons.

**Step 3:** Here is an implementation of both events in which we will start and stop animation.

```
void CMFCAnimationDemoDlg::OnBnClickedButtonStart()
{
    // TODO: Add your control notification handler code here
    m_animationCtrl.Open(L"res\\copyfile.avi");
}

void CMFCAnimationDemoDlg::OnBnClickedButtonStop()
{
    // TODO: Add your control notification handler code here
    m_animationCtrl.Stop();
}
```

**Step 4:** When the above code is compiled and executed, you will see the following output.



**Step 5:** When you click the Stop button, the animation stops and when you press the Start button, it starts again.

## Bitmap Button

A **bitmap button** displays a picture or a picture and text on its face. This is usually intended to make the button a little explicit. A bitmap button is created using the **CBitmapButton class**, which is derived from CButton.

Here is the list of methods in CBitmapButton class.

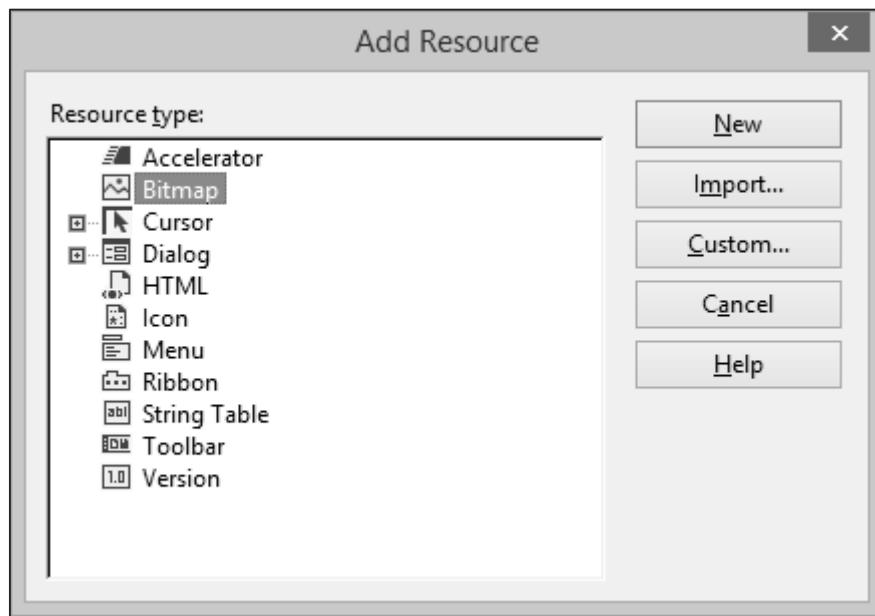
Name	Description
<b>AutoLoad</b>	Associates a button in a dialog box with an object of the CBitmapButton class, loads the bitmap(s) by name, and sizes the button to fit the bitmap.
<b>LoadBitmaps</b>	Initializes the object by loading one or more named bitmap resources from the application's resource file and attaching the bitmaps to the object.
<b>SizeToContent</b>	It resizes the button to the size of the bitmaps.

Here is the list of messages mapping for Bitmap Button control:

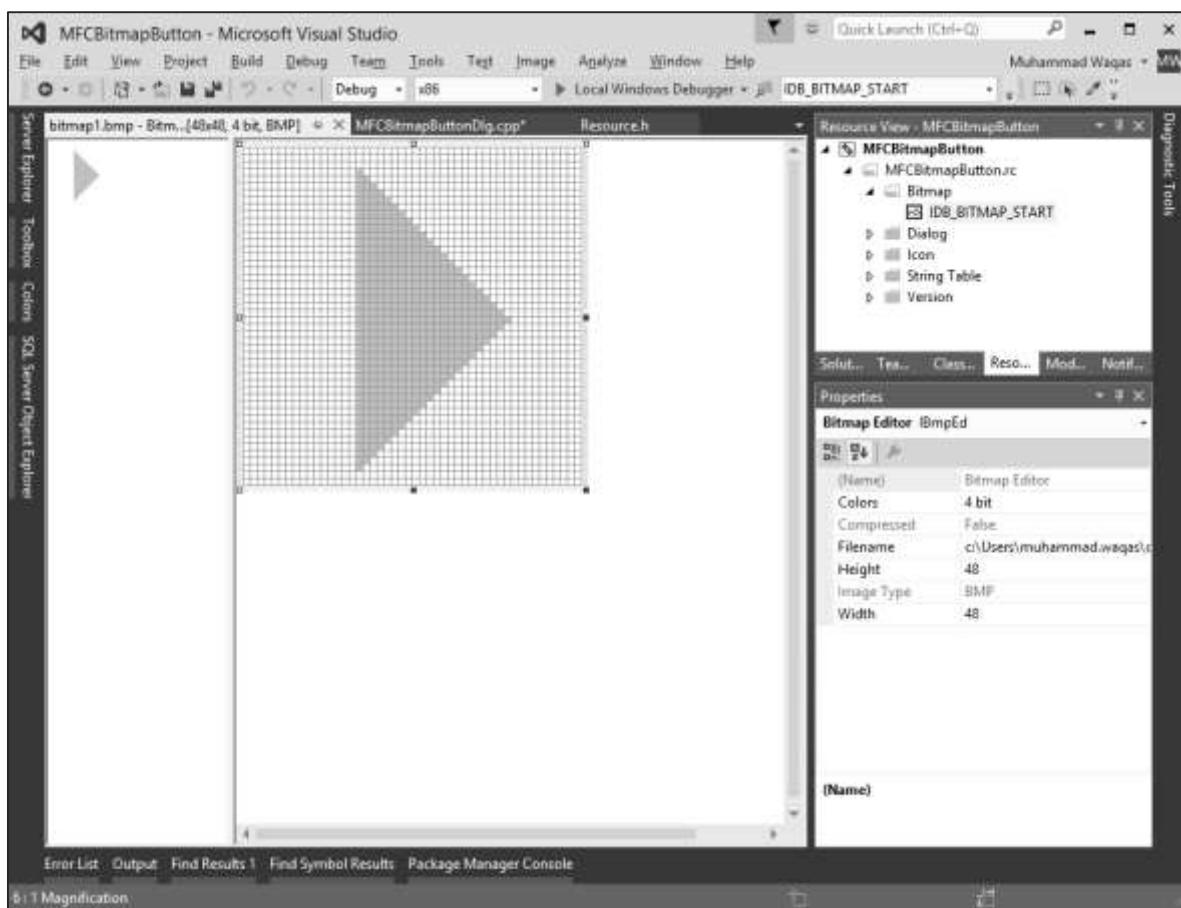
Message	Map entry	Description
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is clicked.
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when the button is disabled.
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is double clicked.
<b>BN_PAINT</b>	ON_BN_PAINT( <id>, <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button.

Let us look into a simple example by creating a new project.

**Step 1:** Add a Bitmap from Add Resource dialog box.



**Step 2:** Select Bitmap and click New.



**Step 3:** Design your bitmap and change its ID to IDB\_BITMAP\_START as shown above.

**Step 4:** Add a button to your dialog box and also add a control Variable m\_buttonStart for that button.

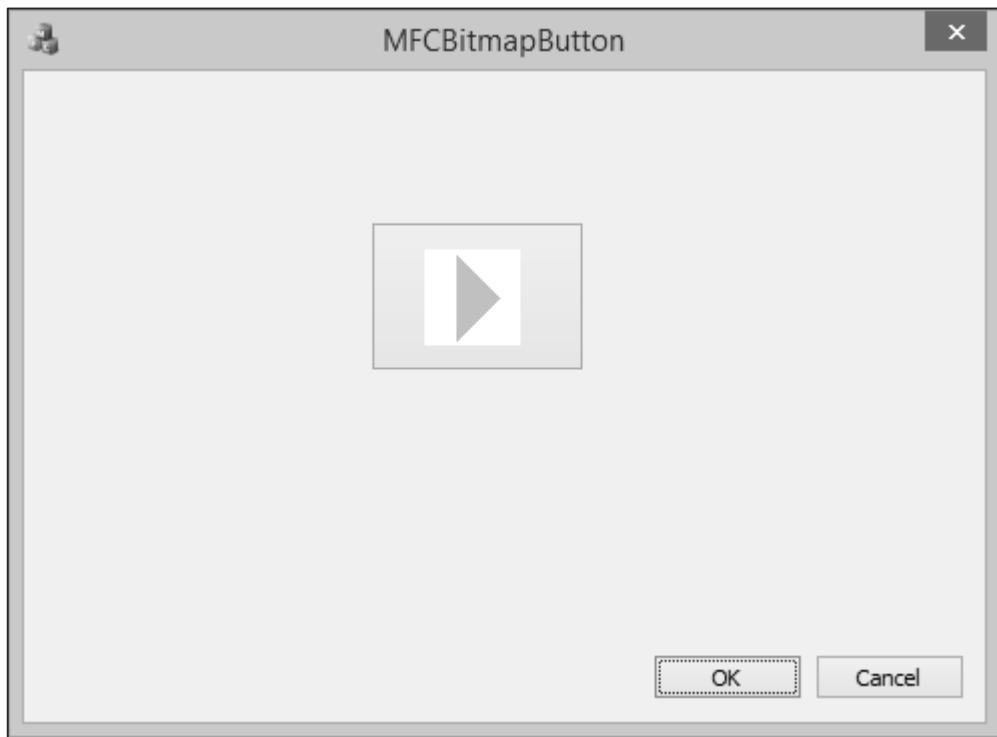
**Step 5:** Add a bitmap variable in your header file. You can now see the following two variables.

```
CBitmap m_bitmapStart;
CButton m_buttonStart;
```

**Step 6:** Modify your OnInitDialog() method as shown in the following code.

```
m_bitmapStart.LoadBitmap(IDB_BITMAP_START);
HBITMAP hBitmap = (HBITMAP)m_bitmapStart.GetSafeHandle();
m_buttonStart.SetBitmap(hBitmap);
```

**Step 7:** When the above code is compiled and executed, you will see the following output.



## Command Button

A **command button** is an enhanced version of the regular button. It displays a green arrow icon on the left, followed by a caption in regular size. Under the main caption, it can display another smaller caption that serves as a hint to provide more information.

Here is the list of methods in CommandButton class:

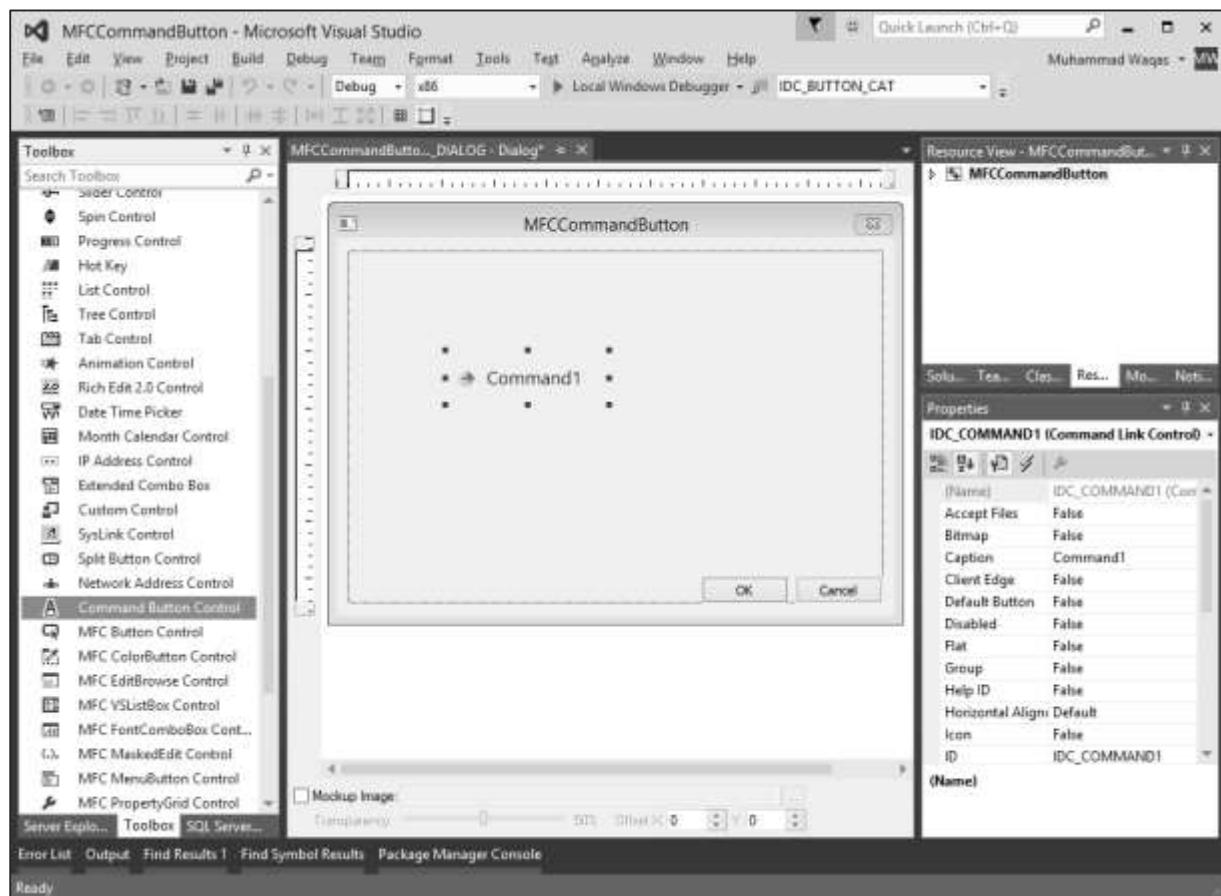
Name	Description
<b>Create</b>	Creates the Windows button control and attaches it to the CButton object.
<b>DrawItem</b>	Override to draw an owner-drawn CButton object.
<b>GetBitmap</b>	Retrieves the handle of the bitmap previously set with SetBitmap.
<b>GetButtonStyle</b>	Retrieves information about the button control style.
<b>GetCheck</b>	Retrieves the check state of a button control.
<b>GetCursor</b>	Retrieves the handle of the cursor image previously set with SetCursor.
<b>GetIcon</b>	Retrieves the handle of the icon previously set with SetIcon.
<b>GetIdealSize</b>	Retrieves the ideal size of the button control.
<b>GetImageList</b>	Retrieves the image list of the button control.
<b>GetNote</b>	Retrieves the note component of the current command link control.
<b>GetNoteLength</b>	Retrieves the length of the note text for the current command link control.
<b>GetSplitGlyph</b>	Retrieves the glyph associated with the current split button control.
<b>GetSplitImageList</b>	Retrieves the image list for the current split button control.
<b>GetSplitInfo</b>	Retrieves information that defines the current split button control.
<b>GetSplitSize</b>	Retrieves the bounding rectangle of the drop-down component of the current split button control.
<b>GetSplitStyle</b>	Retrieves the split button styles that define the current split button control.
<b>GetState</b>	Retrieves the check state, highlight state, and focus state of a button control.
<b>GetTextMargin</b>	Retrieves the text margin of the button control.
<b>SetBitmap</b>	Specifies a bitmap to be displayed on the button.
<b>SetButtonStyle</b>	Changes the style of a button.
<b>SetCheck</b>	Sets the check state of a button control.
<b>SetCursor</b>	Specifies a cursor image to be displayed on the button.
<b>SetDropDownState</b>	Sets the drop-down state of the current split button control.
<b>SetIcon</b>	Specifies an icon to be displayed on the button.
<b>SetImageList</b>	Sets the image list of the button control.
<b>SetNote</b>	Sets the note on the current command link control.
<b>SetSplitGlyph</b>	Associates a specified glyph with the current split button control.
<b>SetSplitImageList</b>	Associates an image list with the current split button control.
<b>SetSplitInfo</b>	Specifies information that defines the current split button control.
<b>SetSplitSize</b>	Sets the bounding rectangle of the drop-down component of the current split button control.
<b>SetSplitStyle</b>	Sets the style of the current split button control.
<b>SetState</b>	Sets the highlighting state of a button control.
<b>SetTextMargin</b>	Sets the text margin of the button control.

Here is the list of messages mapping for Command Button control:

Message	Map entry	Description
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is clicked.
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when the button is disabled.
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is double clicked.
<b>BN_PAINT</b>	ON_BN_PAINT( <id>, <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button.

Let us look into a simple example of command button by creating a new MFC dialog based project MFCCommandButton

**Step 1:** From the Toolbox, add Command Button Control.

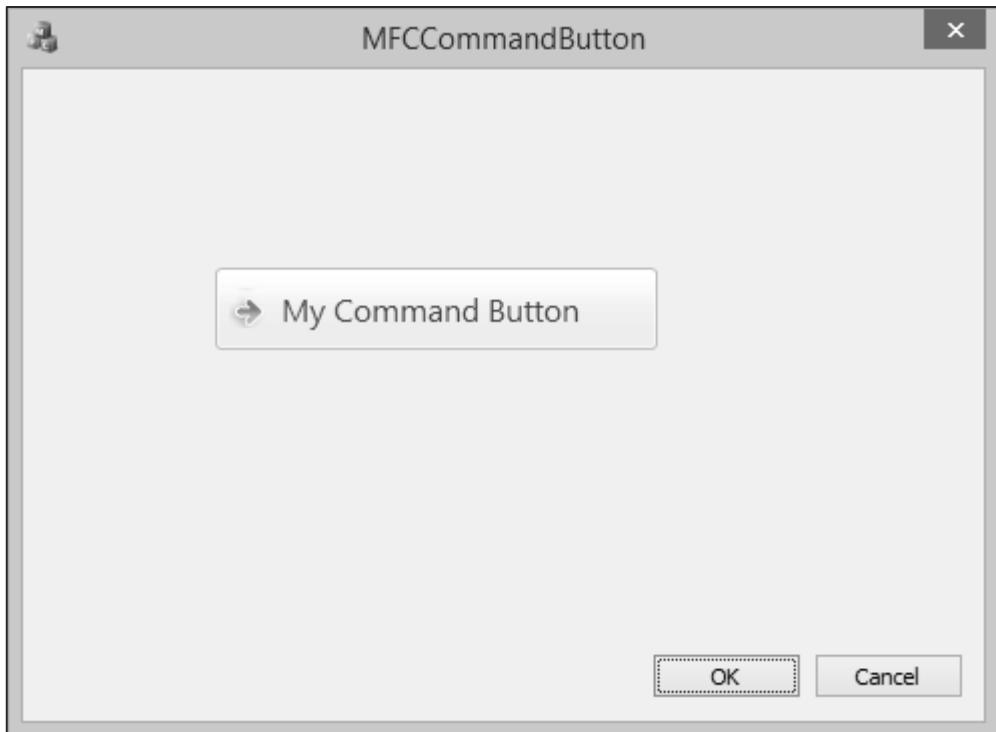


**Step 2:** Change the Caption to My Command button.

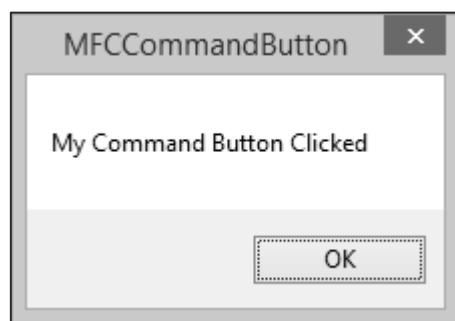
**Step 3:** Add the event handler for this button and add the following message in the event handler.

```
void CMFCCCommandEventDlg::OnBnClickedCommand1()
{
    // TODO: Add your control notification handler code here
    MessageBox(L"My Command Button Clicked");
}
```

**Step 4:** When the above code is compiled and executed, you will see the following output.



**Step 5:** When the My Command Button is clicked; the following message will be displayed.



## Static Text

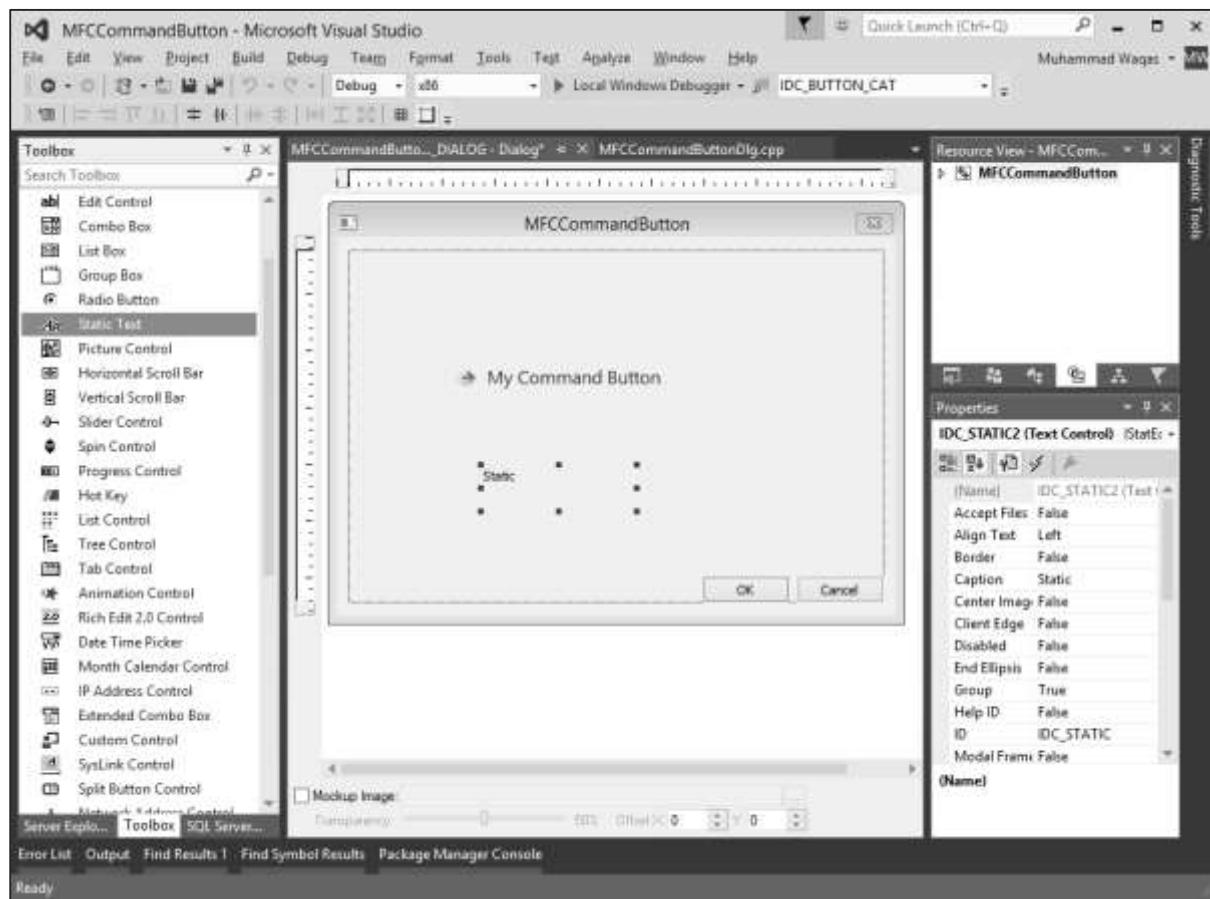
A **static control** displays a text string, box, rectangle, icon, cursor, bitmap, or enhanced metafile. It is represented by **CStatic class**. It can be used to label, box, or separate other controls. A static control normally takes no input and provides no output.

Here is the list of methods in CStatic class:

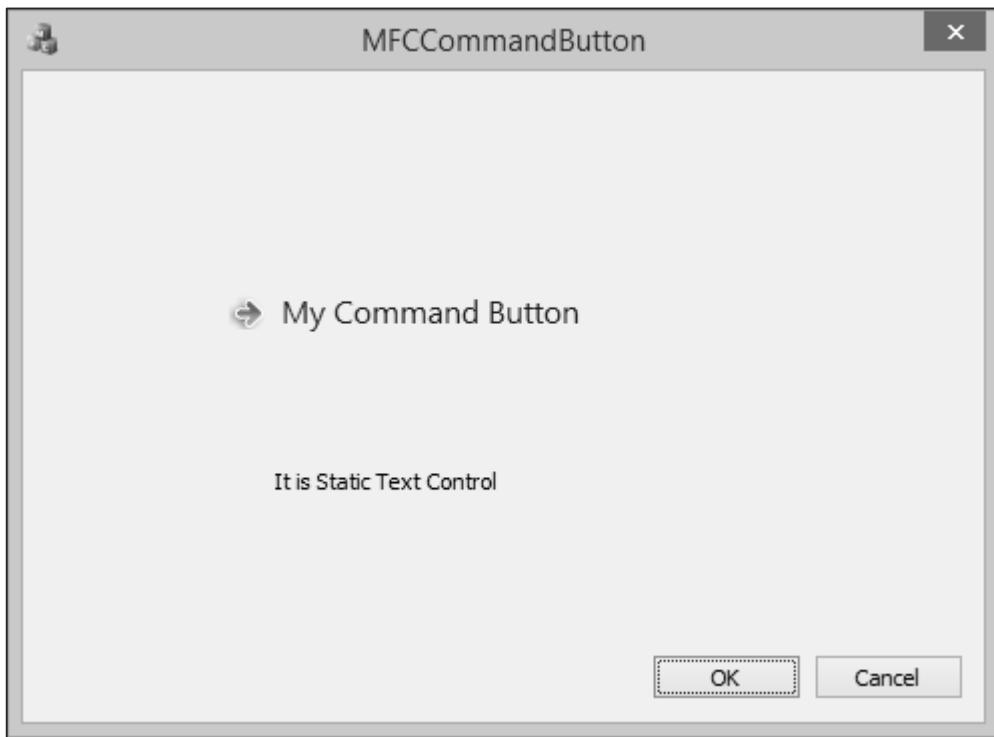
Name	Description
<b>Create</b>	Creates the Windows static control and attaches it to the CStatic object.
<b>DrawItem</b>	Override to draw an owner-drawn static control.
<b>GetBitmap</b>	Retrieves the handle of the bitmap previously set with SetBitmap.
<b>GetCursor</b>	Retrieves the handle of the cursor image previously set with SetCursor.
<b>GetEnhMetaFile</b>	Retrieves the handle of the enhanced metafile previously set with SetEnhMetaFile.
<b>GetIcon</b>	Retrieves the handle of the icon previously set with SetIcon.
<b>SetBitmap</b>	Specifies a bitmap to be displayed in the static control.
<b>SetCursor</b>	Specifies a cursor image to be displayed in the static control.
<b>SetEnhMetaFile</b>	Specifies an enhanced metafile to be displayed in the static control.
<b>SetIcon</b>	Specifies an icon to be displayed in the static control.

Let us look into a simple example of Static Text.

**Step 1:** Drag a Static Text control from Toolbox.



**Step 2:** Change the Caption to **It is Static Text Control** and run your application you will see the following dialog box, which now contains the Static Text control as well.



## List Box

A **list box** displays a list of items, such as filenames, that the user can view and select. A List box is represented by **CListBox class**. In a single-selection list box, the user can select only one item. In a multiple-selection list box, a range of items can be selected. When the user selects an item, it is highlighted and the list box sends a notification message to the parent window.

Here is the list of methods in CListBox class.

Name	Description
<b>AddString</b>	Adds a string to a list box.
<b>CharToItem</b>	Override to provide custom <b>WM_CHAR</b> handling for owner-draw list boxes which don't have strings.
<b>CompareItem</b>	Called by the framework to determine the position of a new item in a sorted owner-draw list box.
<b>Create</b>	Creates the Windows list box and attaches it to the CListBox object.
<b>DeleteItem</b>	Called by the framework when the user deletes an item from an owner-draw list box.
<b>DeleteString</b>	Deletes a string from a list box.
<b>Dir</b>	Adds filenames, drives, or both from the current directory to a list box.

<b>DrawItem</b>	Called by the framework when a visual aspect of an owner-draw list box changes.
<b>FindString</b>	Searches for a string in a list box.
<b>FindStringExact</b>	Finds the first list-box string that matches a specified string.
<b>GetAnchorIndex</b>	Retrieves the zero-based index of the current anchor item in a list box.
<b>GetCaretIndex</b>	Determines the index of the item that has the focus rectangle in a multiple-selection list box.
<b>GetCount</b>	Returns the number of strings in a list box.
<b>GetCurSel</b>	Returns the zero-based index of the currently selected string in a list box.
<b>GetHorizontalExtent</b>	Returns the width in pixels that a list box can be scrolled horizontally.
<b>GetItemData</b>	Returns the 32-bit value associated with the list-box item.
<b>GetItemDataPtr</b>	Returns a pointer to a list-box item.
<b>GetItemHeight</b>	Determines the height of items in a list box.
<b>GetItemRect</b>	Returns the bounding rectangle of the list-box item as it is currently displayed.
<b>GetListBoxInfo</b>	Retrieves the number of items per column.
<b>GetLocale</b>	Retrieves the locale identifier for a list box.
<b>GetSel</b>	Returns the selection state of a list-box item.
<b>GetSelCount</b>	Returns the number of strings currently selected in a multiple-selection list box.
<b>GetSelItems</b>	Returns the indices of the strings currently selected in a list box.
<b>GetText</b>	Copies a list-box item into a buffer.
<b>GetTextLen</b>	Returns the length in bytes of a list-box item.
<b>GetTopIndex</b>	Returns the index of the first visible string in a list box.
<b>InitStorage</b>	Preallocates blocks of memory for list box items and strings.
<b>InsertString</b>	Inserts a string at a specific location in a list box.
<b>ItemFromPoint</b>	Returns the index of the list-box item nearest a point.
<b>MeasureItem</b>	Called by the framework when an owner-draw list box is created to determine list-box dimensions.
<b>ResetContent</b>	Clears all the entries from a list box.
<b>SelectString</b>	Searches for and selects a string in a single-selection list box.
<b>SelItemRange</b>	Selects or deselects a range of strings in a multiple-selection list box.

<b>SetAnchorIndex</b>	Sets the anchor in a multiple-selection list box to begin an extended selection.
<b>SetCaretIndex</b>	Sets the focus rectangle to the item at the specified index in a multiple-selection list box.
<b>SetColumnWidth</b>	Sets the column width of a multicolumn list box.
<b>SetCurSel</b>	Selects a list-box string.
<b>SetHorizontalExtent</b>	Sets the width in pixels that a list box can be scrolled horizontally.
<b>SetItemData</b>	Sets the 32-bit value associated with the list-box item.
<b>SetItemDataPtr</b>	Sets a pointer to the list-box item.
<b>SetItemHeight</b>	Sets the height of items in a list box.
<b>SetLocale</b>	Sets the locale identifier for a list box.
<b>SetSel</b>	Selects or deselects a list-box item in a multiple-selection list box.
<b>SetTabStops</b>	Sets the tab-stop positions in a list box.
<b>SetTopIndex</b>	Sets the zero-based index of the first visible string in a list box.
<b>VKeyToItem</b>	Override to provide custom <b>WM_KEYDOWN</b> handling for list boxes with the <b>LBS_WANTKEYBOARDINPUT</b> style set.

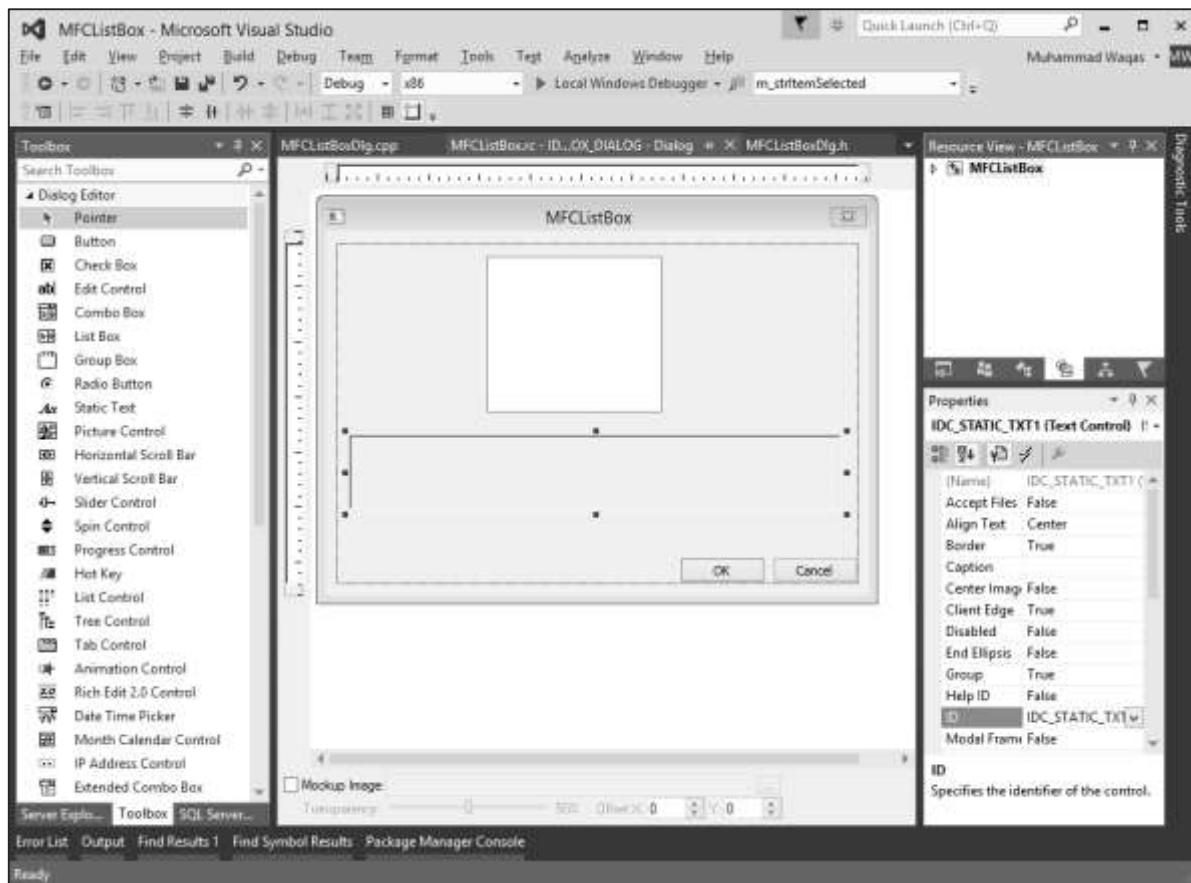
Here are some mapping entries for Listbox:

Message	Map entry	Description
<b>LBN_DBCLK</b>	ON_LBN_DBCLK( <memberFxn> )	<id>, The framework calls this member function when list item is double clicked.
<b>LBN_KILLFOCUS</b>	ON_LBN_KILLFOCUS( <memberFxn> )	<id>, The framework calls this member function immediately before losing the input focus.
<b>LBN_SELCHANGE</b>	ON_LBN_SELCHANGE( <memberFxn> )	<id>, The framework calls this member function when selection is changed.
<b>LBN_SETFOCUS</b>	ON_LBN_SETFOCUS( <memberFxn> )	<id>, The framework calls this member function after gaining the input focus.

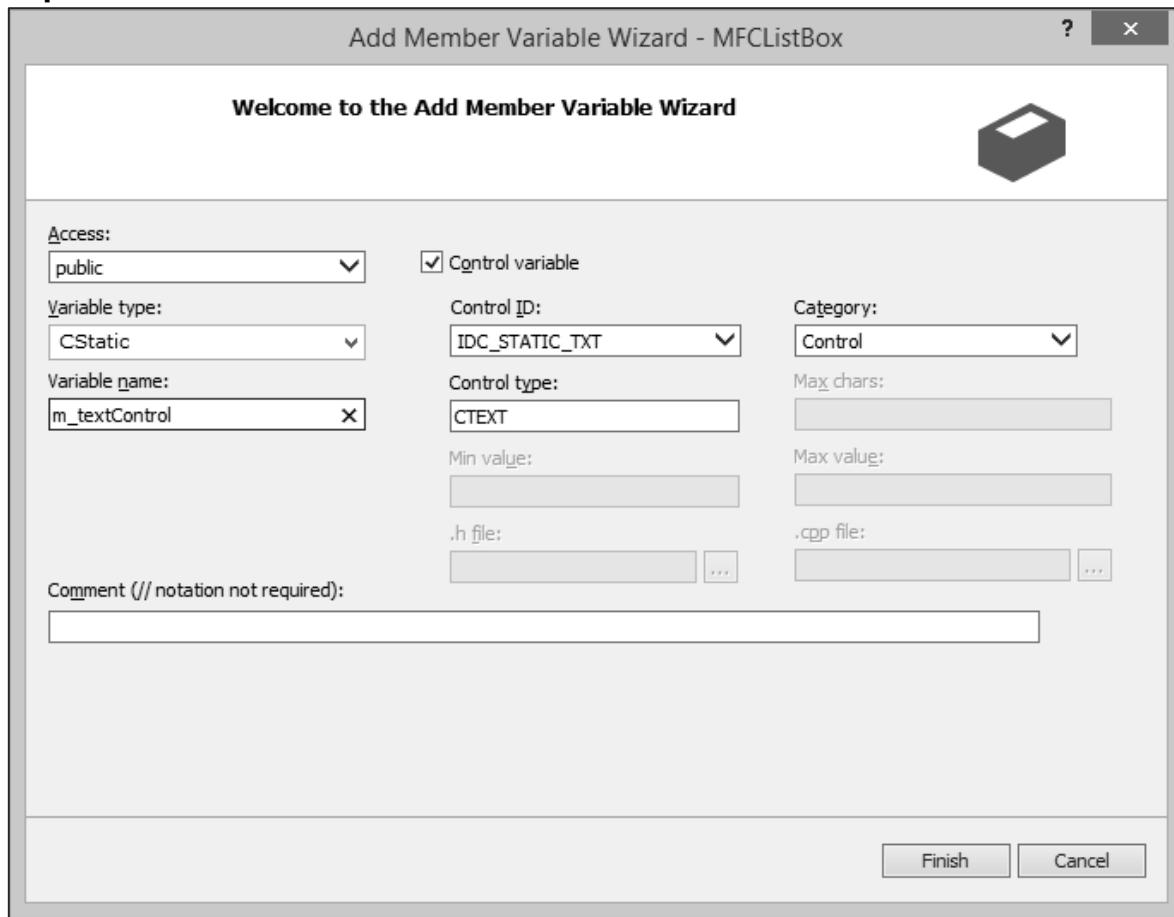
Let us look into a simple example of List box by creating a new MFC dialog based application.

**Step 1:** Once the project is created, you will see the TODO line which is the Caption of Text Control. Remove the Caption and set its ID to IDC\_STATIC\_TXT.

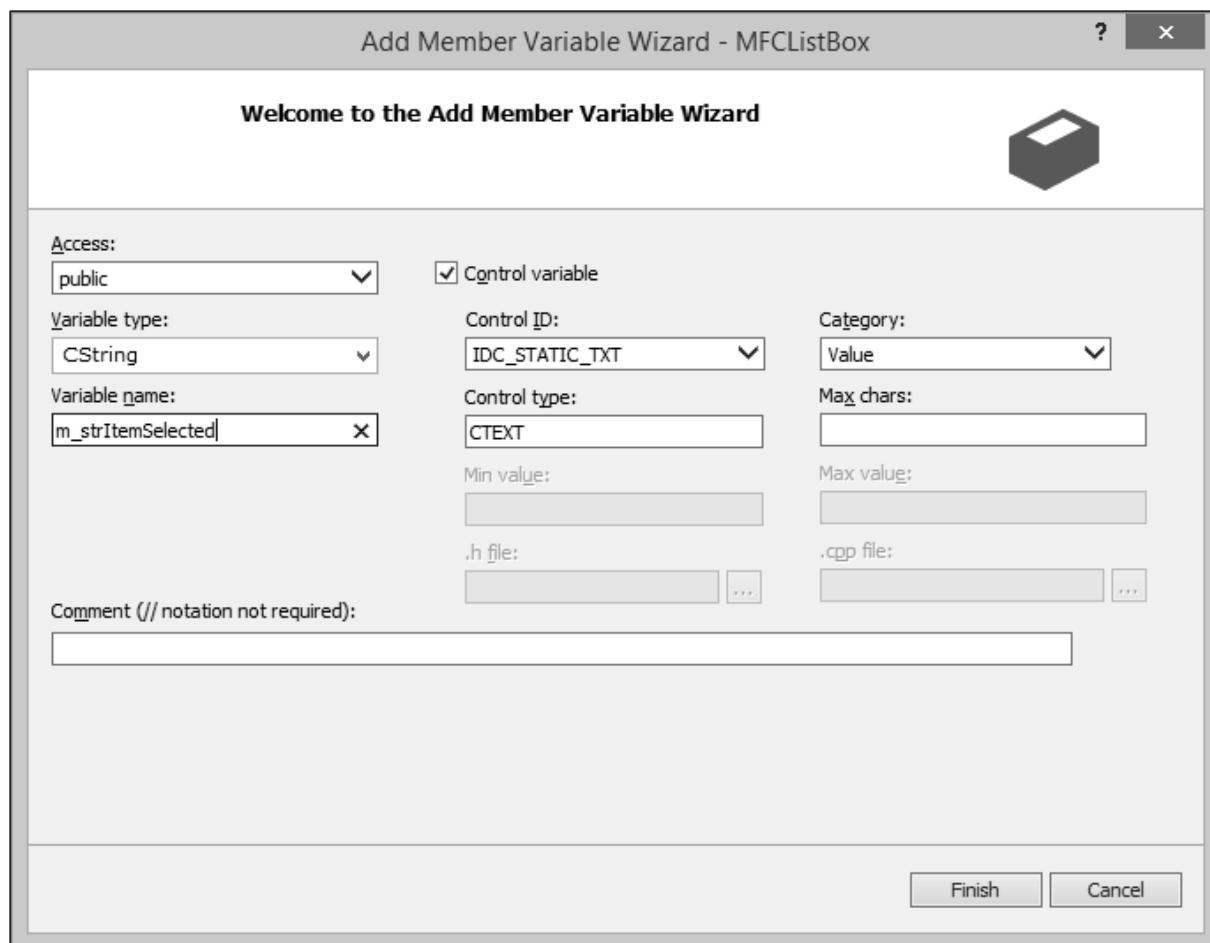
**Step 2:** Drag List Box from the Toolbox.



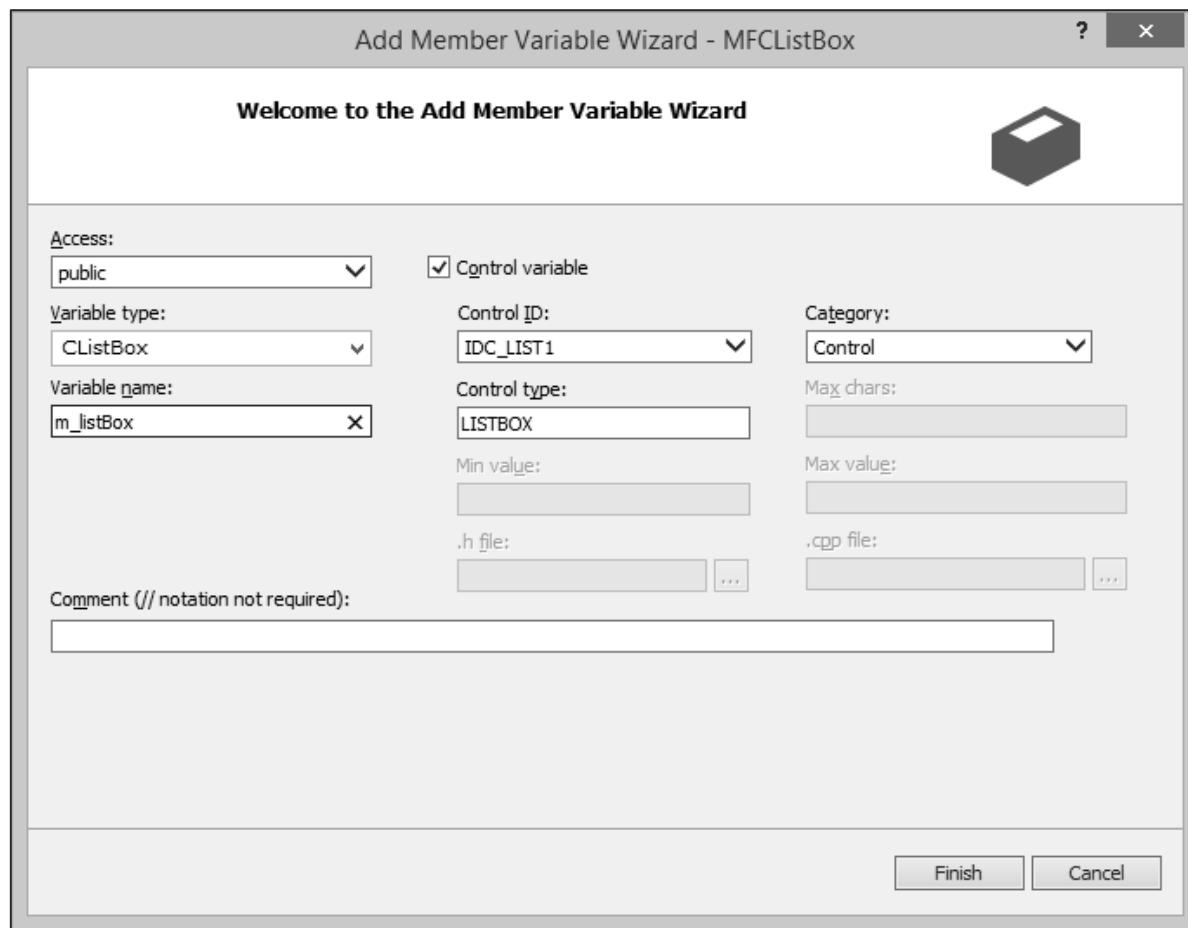
**Step 3:** Add the control variable for the Text control.



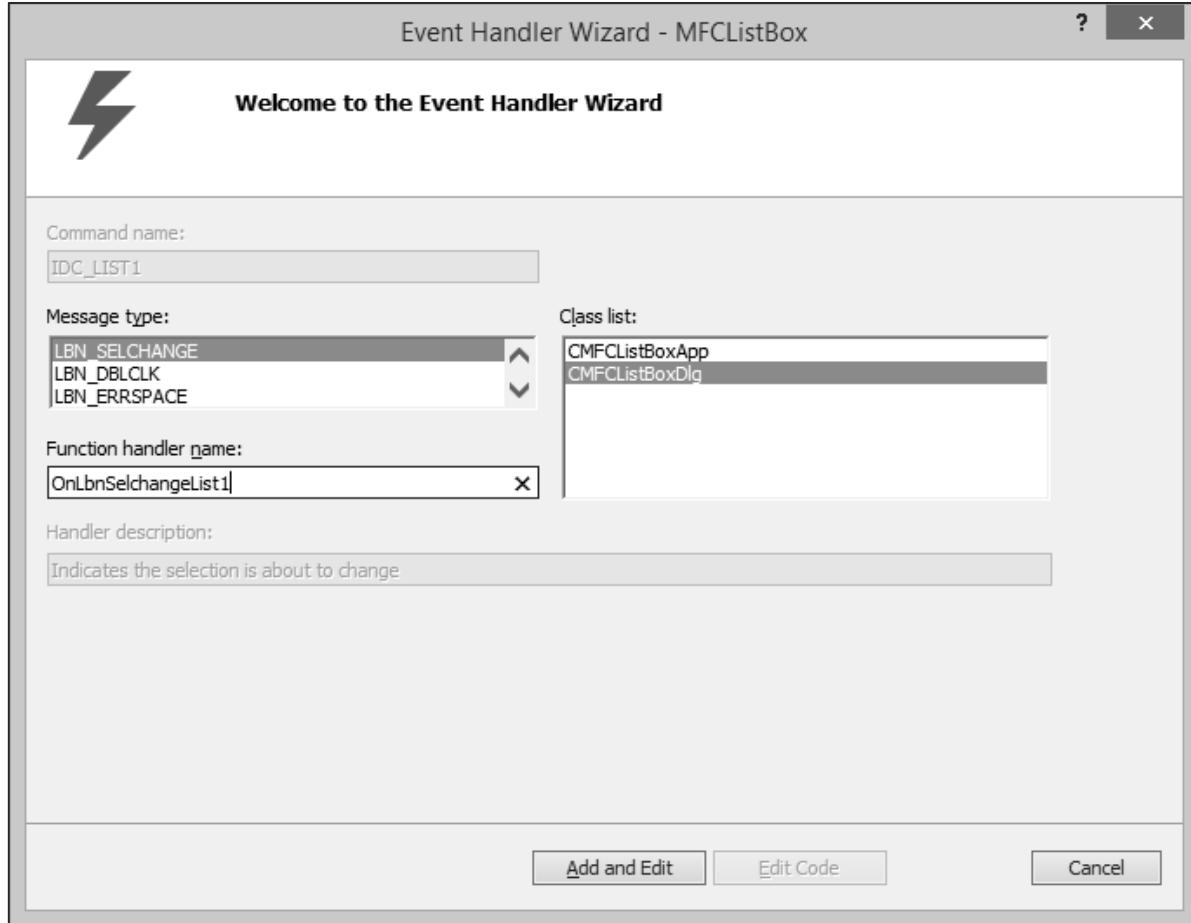
**Step 4:** Add the Value variable for the Text control.



**Step 5:** Add the control variable for the List Box control.



**Step 6:** Add the event handler for the List Box control.



**Step 7:** Select the LBN\_SELCHANGE from the message type and enter name for the event handler.

**Step 8:** Add one function, which will load the list box.

```
void CMFCLListBoxDlg::LoadListBox()
{
    CString str = _T("");
    for (int i = 0; i<10; i++)
    {
        str.Format(_T("Item %d"), i);
        m_listBox.AddString(str);
    }
}
```

**Step 9:** Call the function from CMFCLListBoxDlg::OnInitDialog() as shown in the following code.

```
BOOL CMFCLListBoxDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

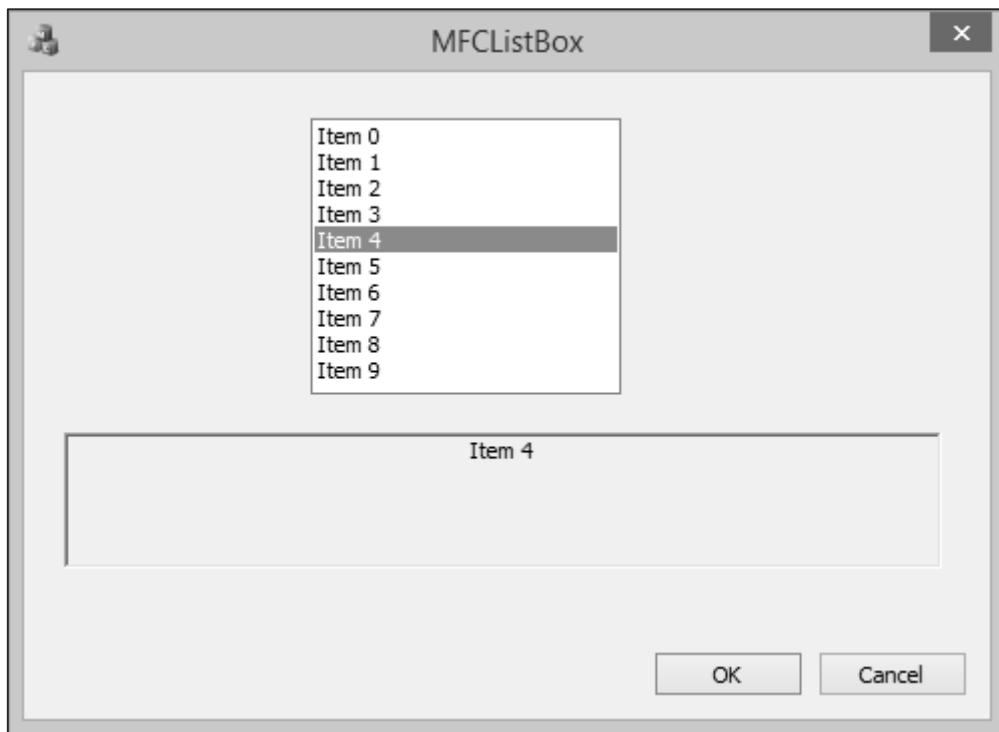
    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // TODO: Add extra initialization here
    LoadListBox();
    return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 10:** Here is the event handler implementation. This will display the current selected item from the List Box.

```
void CMFCLListBoxDlg::OnLbnSelchangeList1()
{
    // TODO: Add your control notification handler code here
    m_listBox.GetText(m_listBox.GetCurSel(),m_strItemSelected);
    UpdateData(FALSE);
}
```

**Step 11:** When the above code is compiled and executed, you will see the following output.



**Step 12:** When you select any item, it will be displayed on the Text Control.

## Combo Boxes

A **combo box** consists of a list box combined with either a static control or edit control. It is represented by **CComboBox** class. The list-box portion of the control may be displayed at all times or may only drop down when the user selects the drop-down arrow next to the control.

Here is the list of methods of CComboBox class:

Name	Description
<b>AddString</b>	Adds a string to the end of the list in the list box of a combo box, or at the sorted position for list boxes with the <b>CBS_SORT</b> style.
<b>Clear</b>	Deletes (clears) the current selection, if any, in the edit control.
<b>CompareItem</b>	Called by the framework to determine the relative position of a new list item in a sorted owner-drawn combo box.
<b>Copy</b>	Copies the current selection, if any, onto the Clipboard in <b>CF_TEXT</b> format.
<b>Create</b>	Creates the combo box and attaches it to the CComboBox object.

<b>Cut</b>	Deletes (cuts) the current selection, if any, in the edit control and copies the deleted text onto the Clipboard in CF_TEXT format.
<b>DeleteItem</b>	Called by the framework when a list item is deleted from an owner-drawn combo box.
<b>DeleteString</b>	Deletes a string from the list box of a combo box.
<b>Dir</b>	Adds a list of file names to the list box of a combo box.
<b>DrawItem</b>	Called by the framework when a visual aspect of an owner-drawn combo box changes.
<b>FindString</b>	Finds the first string that contains the specified prefix in the list box of a combo box.
<b>FindStringExact</b>	Finds the first list-box string (in a combo box) that matches the specified string.
<b>GetComboBoxInfo</b>	Retrieves information about the CComboBox object.
<b>GetCount</b>	Retrieves the number of items in the list box of a combo box.
<b>GetCueBanner</b>	Gets the cue text that is displayed for a combo box control.
<b>GetCurSel</b>	Retrieves the index of the currently selected item, if any, in the list box of a combo box.
<b>GetDroppedControlRect</b>	Retrieves the screen coordinates of the visible (dropped down) list box of a drop-down combo box.
<b>GetDroppedState</b>	Determines whether the list box of a drop-down combo box is visible (dropped down).
<b>GetDroppedWidth</b>	Retrieves the minimum allowed width for the drop-down list-box portion of a combo box.
<b>GetEditSel</b>	Gets the starting and ending character positions of the current selection in the edit control of a combo box.
<b>GetExtendedUI</b>	Determines whether a combo box has the default user interface or the extended user interface.
<b>GetHorizontalExtent</b>	Returns the width in pixels that the list-box portion of the combo box can be scrolled horizontally.
<b>GetItemData</b>	Retrieves the application-supplied 32-bit value associated

	with the specified combo-box item.
<b>GetItemDataPtr</b>	Retrieves the application-supplied 32-bit pointer that is associated with the specified combo-box item.
<b>GetItemHeight</b>	Retrieves the height of list items in a combo box.
<b>GetLBText</b>	Gets a string from the list box of a combo box.
<b>GetLBTextLen</b>	Gets the length of a string in the list box of a combo box.
<b>GetLocale</b>	Retrieves the locale identifier for a combo box.
<b>GetMinVisible</b>	Gets the minimum number of visible items in the drop-down list of the current combo box.
<b>GetTopIndex</b>	Returns the index of the first visible item in the list-box portion of the combo box.
<b>InitStorage</b>	Preallocates blocks of memory for items and strings in the list-box portion of the combo box.
<b>InsertString</b>	Inserts a string into the list box of a combo box.
<b>LimitText</b>	Limits the length of the text that the user can enter into the edit control of a combo box.
<b>MeasureItem</b>	Called by the framework to determine combo box dimensions when an owner-drawn combo box is created.
<b>Paste</b>	Inserts the data from the Clipboard into the edit control at the current cursor position. Data is inserted only if the Clipboard contains data in CF_TEXT format.
<b>ResetContent</b>	Removes all items from the list box and edit control of a combo box.
<b>SelectString</b>	Searches for a string in the list box of a combo box and, if the string is found, selects the string in the list box and copies the string to the edit control.
<b>SetCueBanner</b>	Sets the cue text that is displayed for a combo box control.
<b>SetCurSel</b>	Selects a string in the list box of a combo box.
<b>SetDroppedWidth</b>	Sets the minimum allowed width for the drop-down list-box portion of a combo box.

<b>SetEditSel</b>	Selects characters in the edit control of a combo box.
<b>SetExtendedUI</b>	Selects either the default user interface or the extended user interface for a combo box that has the <b>CBS_DROPDOWN</b> or <b>CBS_DROPDOWNLIST</b> style.
<b>SetHorizontalExtent</b>	Sets the width in pixels that the list-box portion of the combo box can be scrolled horizontally.
<b>SetItemData</b>	Sets the 32-bit value associated with the specified item in a combo box.
<b>SetItemDataPtr</b>	Sets the 32-bit pointer associated with the specified item in a combo box.
<b>SetItemHeight</b>	Sets the height of list items in a combo box or the height of the edit-control (or static-text) portion of a combo box.
<b>SetLocale</b>	Sets the locale identifier for a combo box.
<b>SetMinVisibleItems</b>	Sets the minimum number of visible items in the drop-down list of the current combo box.
<b>SetTopIndex</b>	Tells the list-box portion of the combo box to display the item with the specified index at the top.
<b>ShowDropDown</b>	Shows or hides the list box of a combo box that has the <b>CBS_DROPDOWN</b> or <b>CBS_DROPDOWNLIST</b> style.

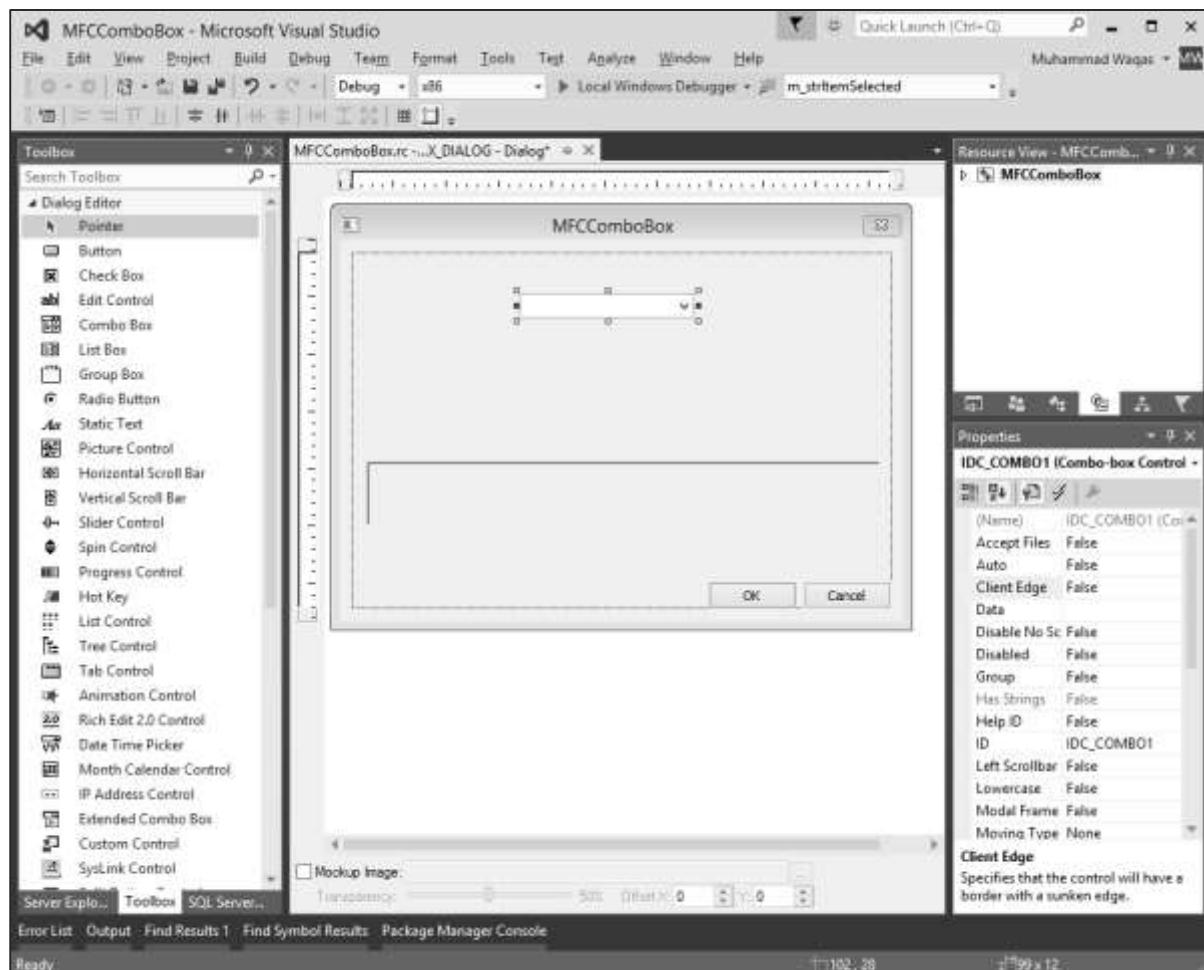
Here is the list of messages mapping for Combobox control:

Message	Map entry	Description
<b>CBN_DBCLK</b>	ON_CBN_DBCLK( <id>, <memberFxn> )	The user double-clicks a string in the list box of a combo box
<b>CBN_DROPDOWN</b>	ON_CBN_DROPDOWN( <id>, <memberFxn> )	The list box of a combo box is about to drop down (be made visible).
<b>CBN_EDITCHANGE</b>	ON_CBN_EDITCHANGE( <id>, <memberFxn> )	The user has taken an action that may have altered the text in the edit-control portion of a combo box.
<b>CBN_EDITUPDATE</b>	ON_CBN_EDITUPDATE( <id>, <memberFxn> )	The edit-control portion of a combo box is about to display altered text.
<b>CBN_KILLFOCUS</b>	ON_CBN_KILLFOCUS( <id>, <memberFxn> )	The combo box is losing the input focus.

<b>CBN_SELCHANGE</b>	ON_CBN_SELCHANGE( <id>, <memberFxn> )	The selection in the list box of a combo box is about to be changed as a result of the user either clicking in the list box or changing the selection by using the arrow keys.
<b>CBN_SETFOCUS</b>	ON_CBN_SETFOCUS( <id>, <memberFxn> )	The combo box receives the input focus.

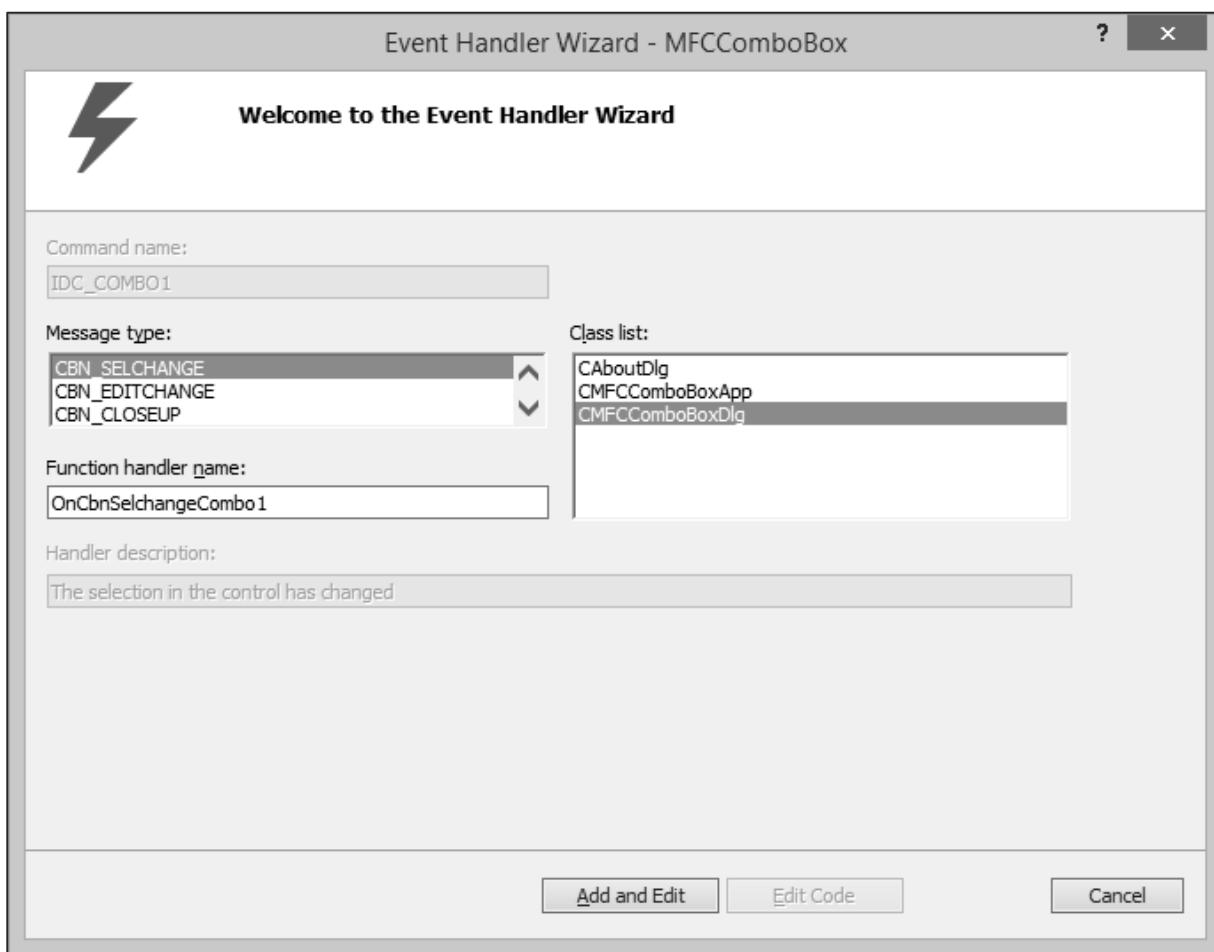
Let us look into an example of Radio button by creating a new MFC dialog based application.

**Step 1:** Drag a Combo box and remove the Caption of Static Text control.



**Step 2:** Add a control variable `m_comboBoxCtrl` for combobox and value variable `m_strTextCtrl` for Static Text control.

**Step 3:** Add event handler for selection change of combo box.



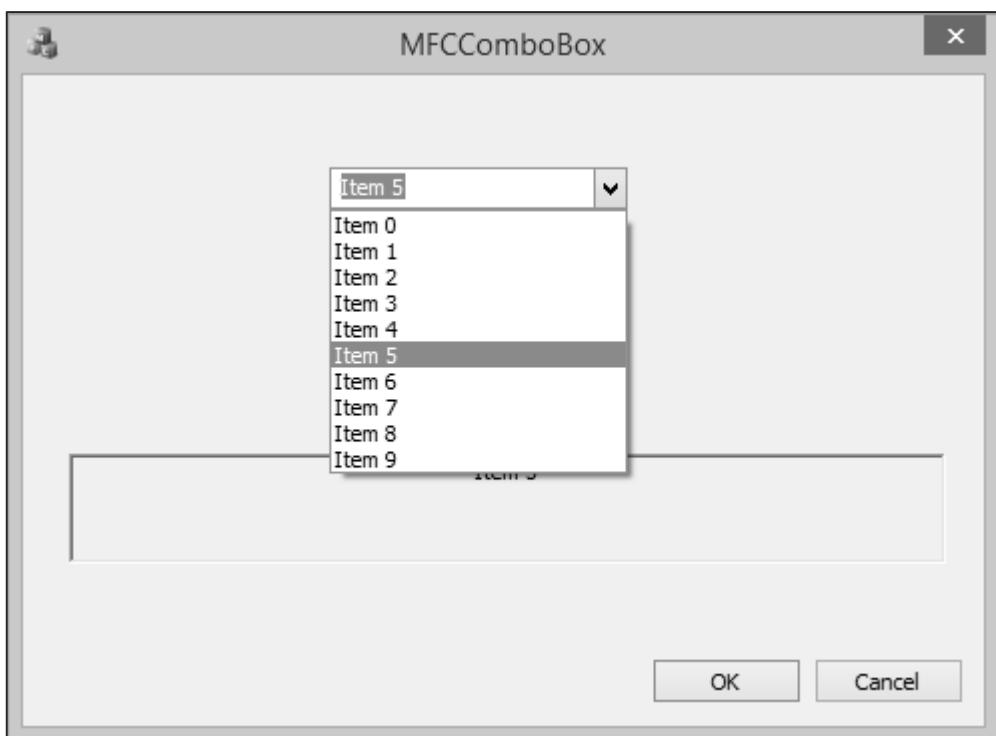
**Step 4:** Add the following code in OnInitDialog() to load the combo box.

```
for (int i = 0; i<10; i++)
{
    str.Format(_T("Item %d"), i);
    m_comboBoxCtrl.AddString(str);
}
```

**Step 5:** Here is the implementation of event handler.

```
void CMFCCComboBoxDlg::OnCbnSelchangeCombo1()
{
    // TODO: Add your control notification handler code here
    m_comboBoxCtrl.GetLBText(m_comboBoxCtrl.GetCurSel(), m_strTextCtrl);
    UpdateData(FALSE);
}
```

**Step 6:** When the above code is compiled and executed, you will see the following output.



**Step 7:** When you select any item then it will be displayed on the Text Control.

## Radio Buttons

A **radio button** is a control that appears as a dot surrounded by a round box. In reality, a radio button is accompanied by one or more other radio buttons that appear and behave as a group.

Here is the list of methods in **Radio Button class**.

Name	Description
<b>Create</b>	Creates the Windows button control and attaches it to the CButton object.
<b>DrawItem</b>	Override to draw an owner-drawn CButton object.

<b>GetBitmap</b>	Retrieves the handle of the bitmap previously set with SetBitmap.
<b>GetButtonStyle</b>	Retrieves information about the button control style.
<b>GetCheck</b>	Retrieves the check state of a button control.
<b>GetCursor</b>	Retrieves the handle of the cursor image previously set with SetCursor.
<b>GetIcon</b>	Retrieves the handle of the icon previously set with SetIcon.
<b>GetIdealSize</b>	Retrieves the ideal size of the button control.
<b>GetImageList</b>	Retrieves the image list of the button control.
<b>GetNote</b>	Retrieves the note component of the current command link control.
<b>GetNoteLength</b>	Retrieves the length of the note text for the current command link control.
<b>GetSplitGlyph</b>	Retrieves the glyph associated with the current split button control.
<b>GetSplitImageList</b>	Retrieves the image list for the current split button control.
<b>GetSplitInfo</b>	Retrieves information that defines the current split button control.
<b>GetSplitSize</b>	Retrieves the bounding rectangle of the drop-down component of the current split button control.
<b>GetSplitStyle</b>	Retrieves the split button styles that define the current split button control.
<b>GetState</b>	Retrieves the check state, highlight state, and focus state of a button control.
<b>GetTextMargin</b>	Retrieves the text margin of the button control.
<b>SetBitmap</b>	Specifies a bitmap to be displayed on the button.
<b>SetButtonStyle</b>	Changes the style of a button.
<b>SetCheck</b>	Sets the check state of a button control.
<b>SetCursor</b>	Specifies a cursor image to be displayed on the button.
<b>SetDropDownState</b>	Sets the drop-down state of the current split button control.
<b>SetIcon</b>	Specifies an icon to be displayed on the button.
<b>SetImageList</b>	Sets the image list of the button control.
<b>SetNote</b>	Sets the note on the current command link control.
<b>SetSplitGlyph</b>	Associates a specified glyph with the current split button control.
<b>SetSplitImageList</b>	Associates an image list with the current split button control.
<b>SetSplitInfo</b>	Specifies information that defines the current split button control.
<b>SetSplitSize</b>	Sets the bounding rectangle of the drop-down component of the current split button control.
<b>SetSplitStyle</b>	Sets the style of the current split button control.
<b>SetState</b>	Sets the highlighting state of a button control.
<b>SetTextMargin</b>	Sets the text margin of the button control.

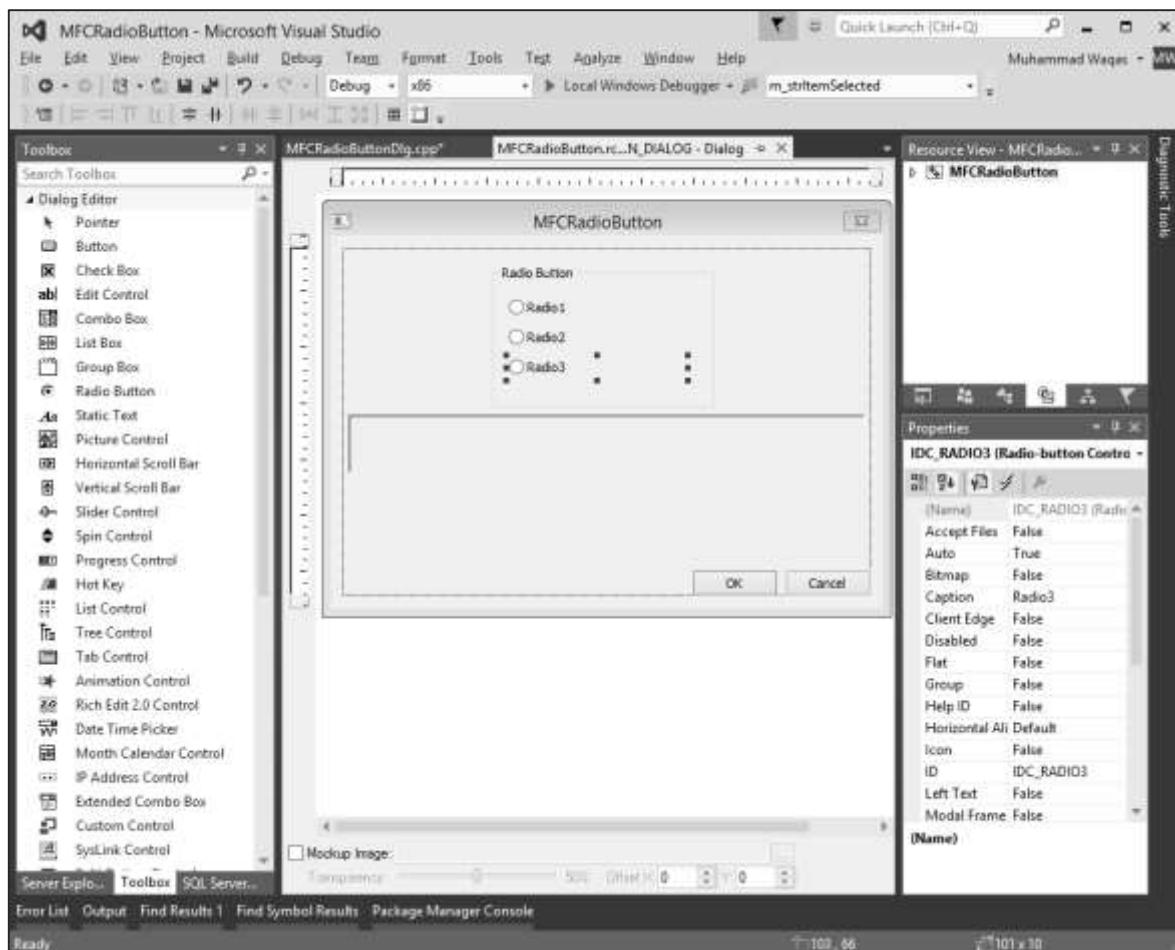
Here is the list of messages mapping for Radio Button control:

Message	Map entry	Description
---------	-----------	-------------

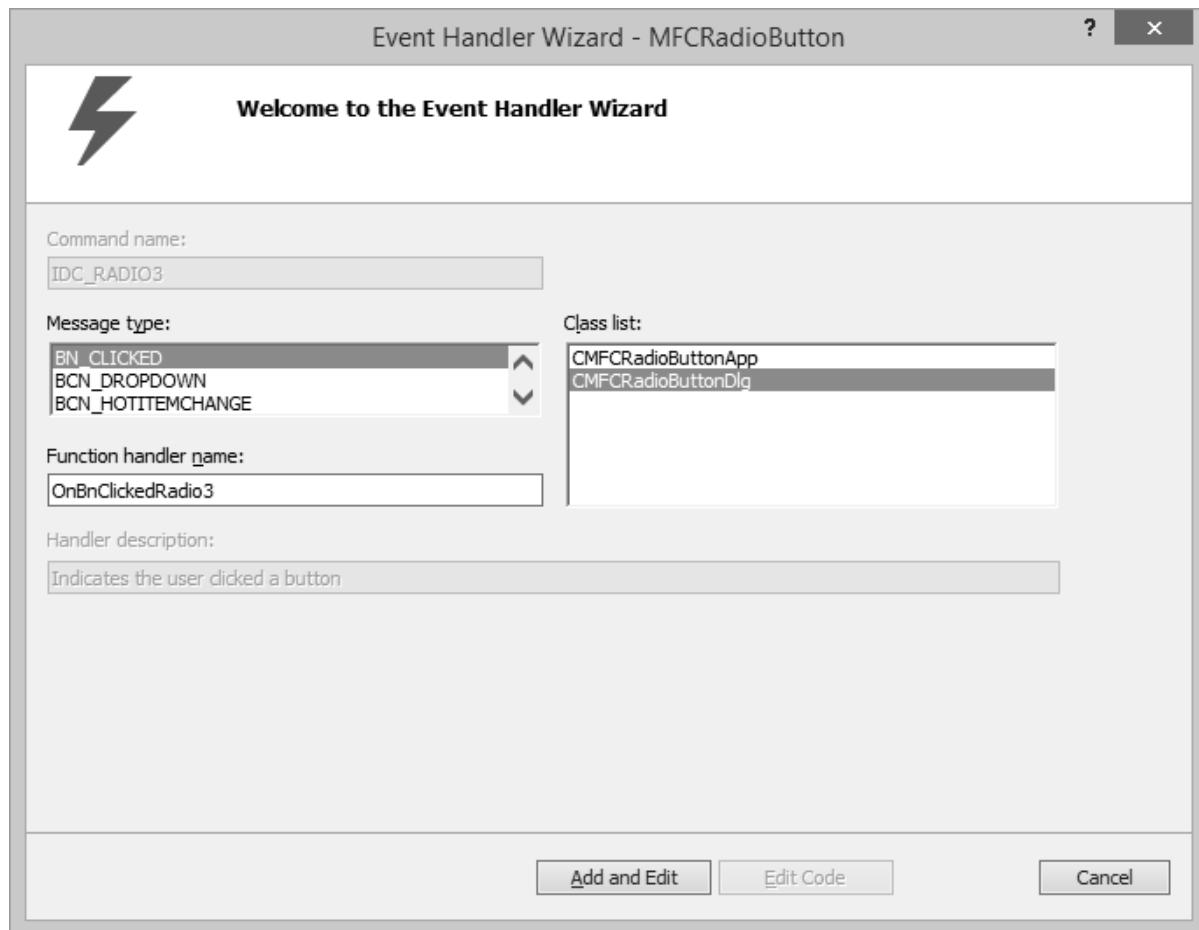
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when a button is clicked
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when a button is disabled
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when a button is double clicked
<b>BN_PAINT</b>	ON_BN_PAINT( <id>, <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button

Let us look into an example of Radio button by creating a new MFC dialog based application

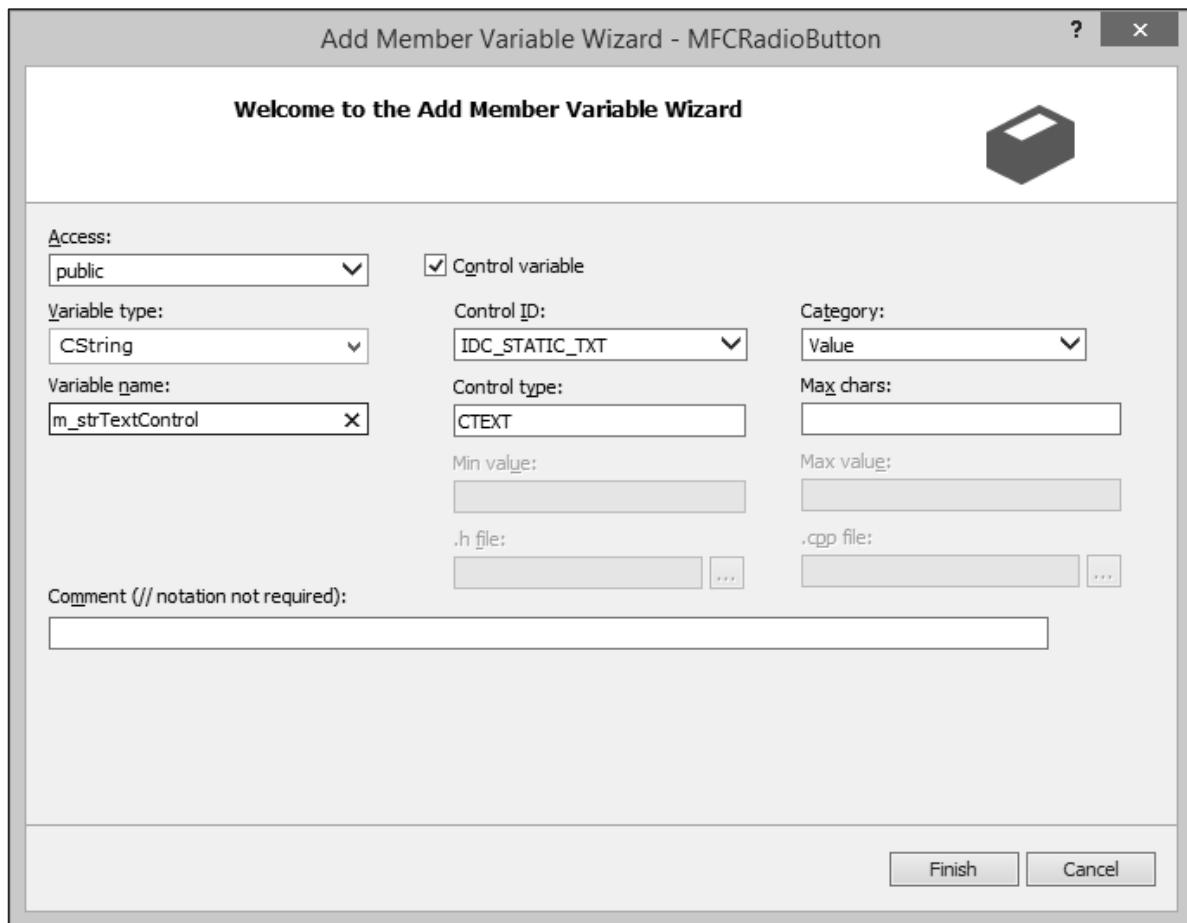
**Step 1:** Drag a group box and three radio buttons and remove the Caption of Static Text control.



**Step 2:** Add event handler for all the three radio buttons.



**Step 3:** Add the Value variable for the Static Text control.



**Step 4:** Here is the implementation of three event handlers.

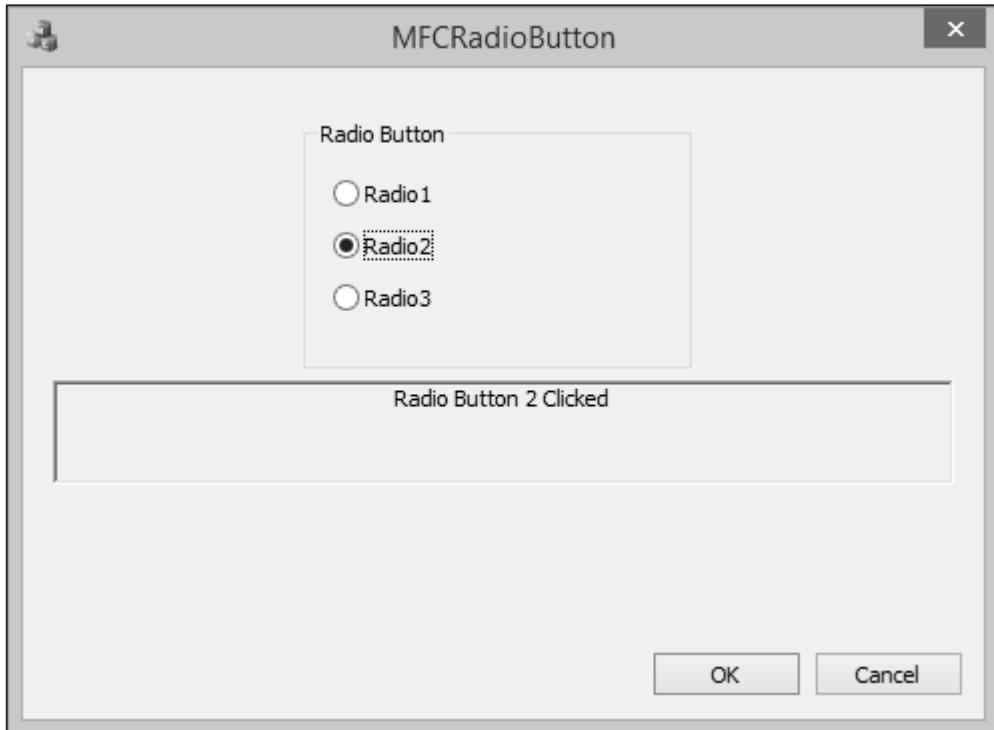
```
void CMFCRadioButtonDlg::OnBnClickedRadio1()
{
    // TODO: Add your control notification handler code here
    m_strTextControl = _T("Radio Button 1 Clicked");
    UpdateData(FALSE);
}

void CMFCRadioButtonDlg::OnBnClickedRadio2()
{
    // TODO: Add your control notification handler code here
    m_strTextControl = _T("Radio Button 2 Clicked");
    UpdateData(FALSE);
}

void CMFCRadioButtonDlg::OnBnClickedRadio3()
{
```

```
// TODO: Add your control notification handler code here
m_strTextControl = _T("Radio Button 3 Clicked");
UpdateData(FALSE);
}
```

**Step 5:** When the above code is compiled and executed, you will see the following output. When you select any radio button, the message is displayed on Static Text control.



## Checkboxes

A checkbox is a Windows control that allows the user to set or change the value of an item as true or false.

Here is the list of methods in **Checkbox class**:

Name	Description
<b>Create</b>	Creates the Windows button control and attaches it to the <b>CButton</b> object.
<b>DrawItem</b>	Override to draw an owner-drawn <b>CButton</b> object.
<b>GetBitmap</b>	Retrieves the handle of the bitmap previously set with <a href="#">SetBitmap</a> .
<b>GetButtonStyle</b>	Retrieves information about the button control style.
<b>GetCheck</b>	Retrieves the check state of a button control.
<b>GetCursor</b>	Retrieves the handle of the cursor image previously set with <a href="#">SetCursor</a> .
<b>GetIcon</b>	Retrieves the handle of the icon previously set with <a href="#">SetIcon</a> .

<b>GetIdealSize</b>	Retrieves the ideal size of the button control.
<b>GetImageList</b>	Retrieves the image list of the button control.
<b>GetNote</b>	Retrieves the note component of the current command link control.
<b>GetNoteLength</b>	Retrieves the length of the note text for the current command link control.
<b>GetSplitGlyph</b>	Retrieves the glyph associated with the current split button control.
<b>GetSplitImageList</b>	Retrieves the image list for the current split button control.
<b>GetSplitInfo</b>	Retrieves information that defines the current split button control.
<b>GetSplitSize</b>	Retrieves the bounding rectangle of the drop-down component of the current split button control.
<b>GetSplitStyle</b>	Retrieves the split button styles that define the current split button control.
<b>GetState</b>	Retrieves the check state, highlight state, and focus state of a button control.
<b>GetTextMargin</b>	Retrieves the text margin of the button control.
<b>SetBitmap</b>	Specifies a bitmap to be displayed on the button.
<b>SetButtonStyle</b>	Changes the style of a button.
<b>SetCheck</b>	Sets the check state of a button control.
<b>SetCursor</b>	Specifies a cursor image to be displayed on the button.
<b>SetDropDownState</b>	Sets the drop-down state of the current split button control.
<b>SetIcon</b>	Specifies an icon to be displayed on the button.
<b>SetImageList</b>	Sets the image list of the button control.
<b>SetNote</b>	Sets the note on the current command link control.
<b>SetSplitGlyph</b>	Associates a specified glyph with the current split button control.
<b>SetSplitImageList</b>	Associates an image list with the current split button control.
<b>SetSplitInfo</b>	Specifies information that defines the current split button control.
<b>SetSplitSize</b>	Sets the bounding rectangle of the drop-down component of the current split button control.
<b>SetSplitStyle</b>	Sets the style of the current split button control.
<b>SetState</b>	Sets the highlighting state of a button control.
<b>SetTextMargin</b>	Sets the text margin of the button control.

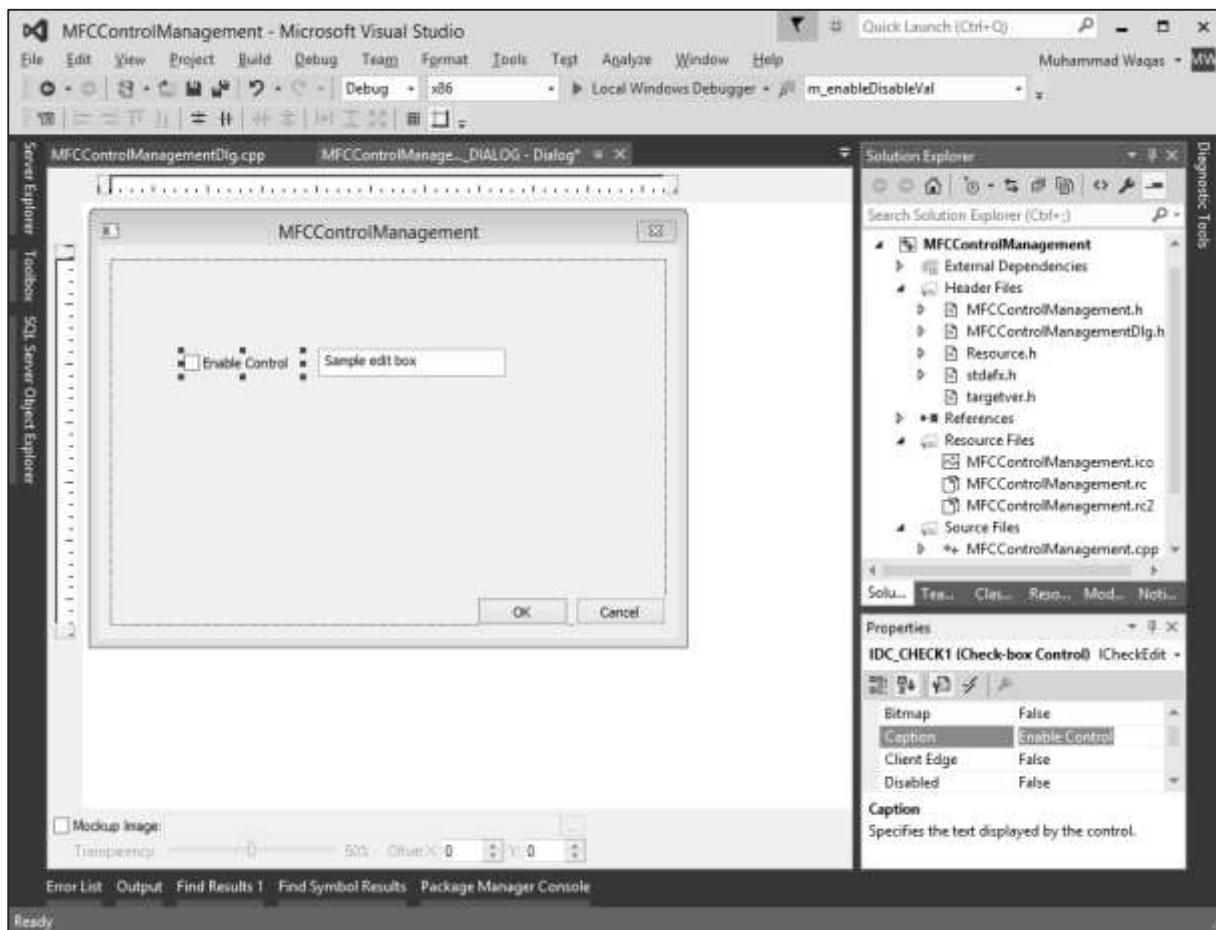
Here is the list of messages mapping for checkbox control:

Message	Map entry	Description
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is clicked.
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when the button is disabled.
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is double clicked.
<b>BN_PAINT</b>	ON_BN_PAINT( <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button.

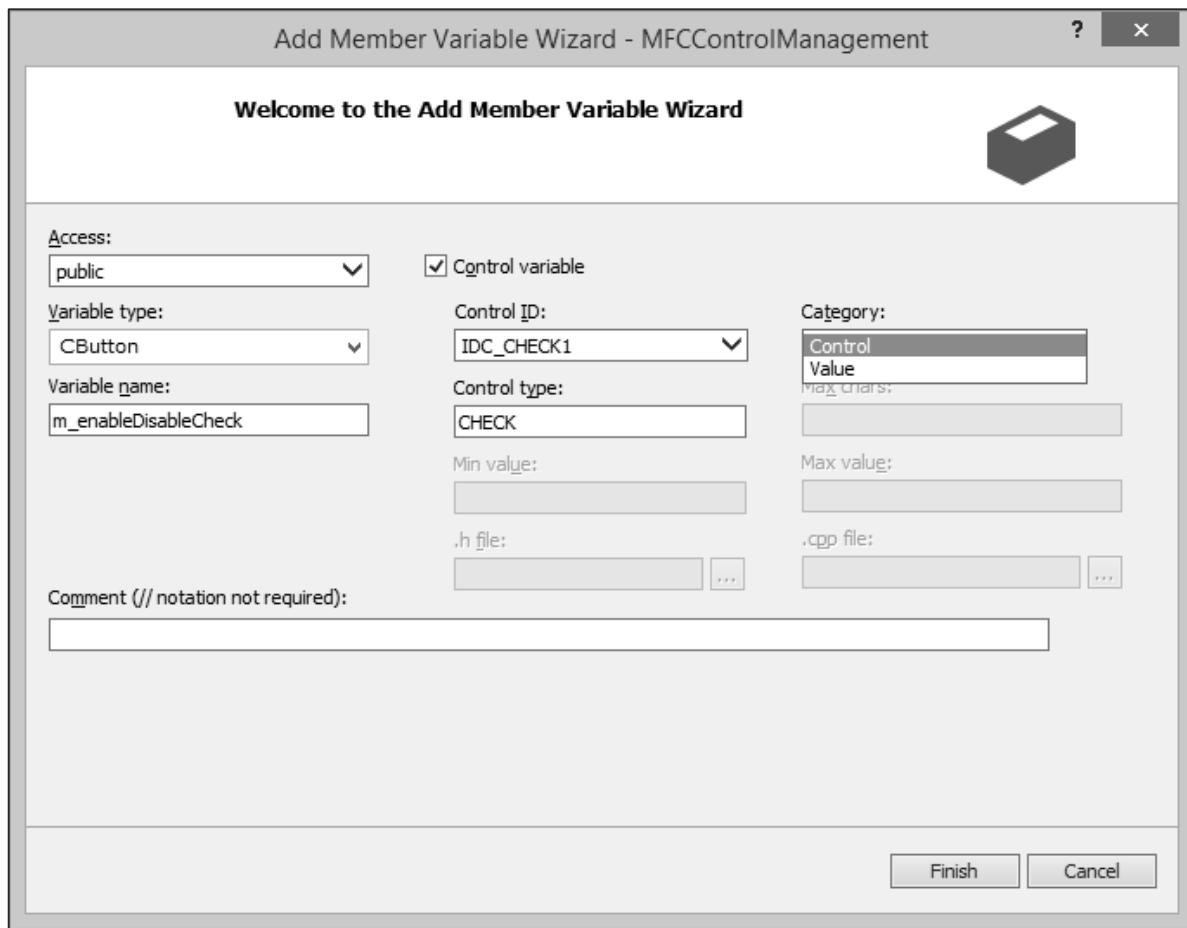
Let us create a new MFC dialog based project.

Once the project is created, you will see the following dialog box in designer window.

**Step 1:** Delete the TODO line and drag one checkbox and one Edit control as shown in the following snapshot. Also change the caption of checkbox to Enable Control.



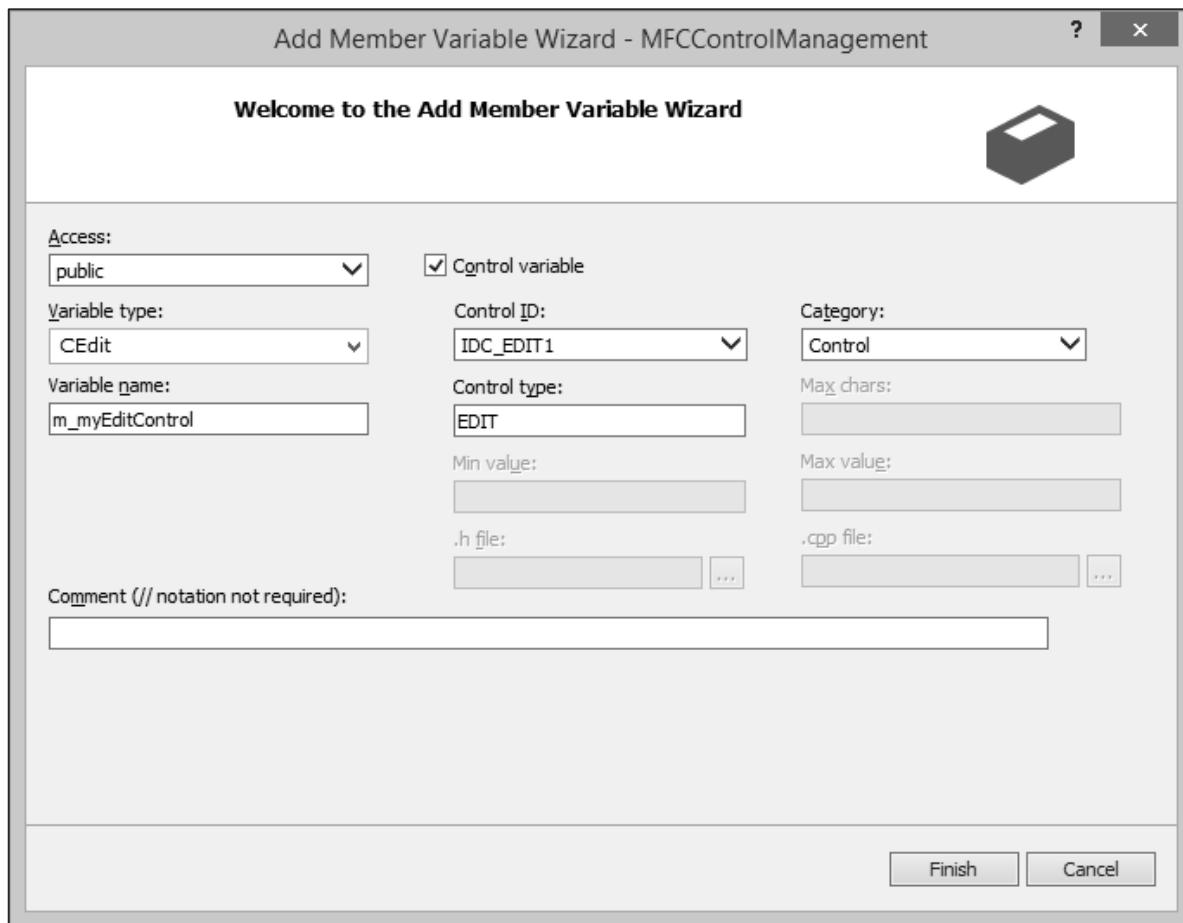
**Step 2:** Right-click on the checkbox and select Add Variable.



**Step 3:** You can select different options on this dialog box. For checkbox, the CButton variable type is selected by default.

**Step 4:** Similarly, the control ID is also selected by default. We now need to select Control in the Category combo box, and type m\_enableDisableCheck in the Variable Name edit box and click Finish.

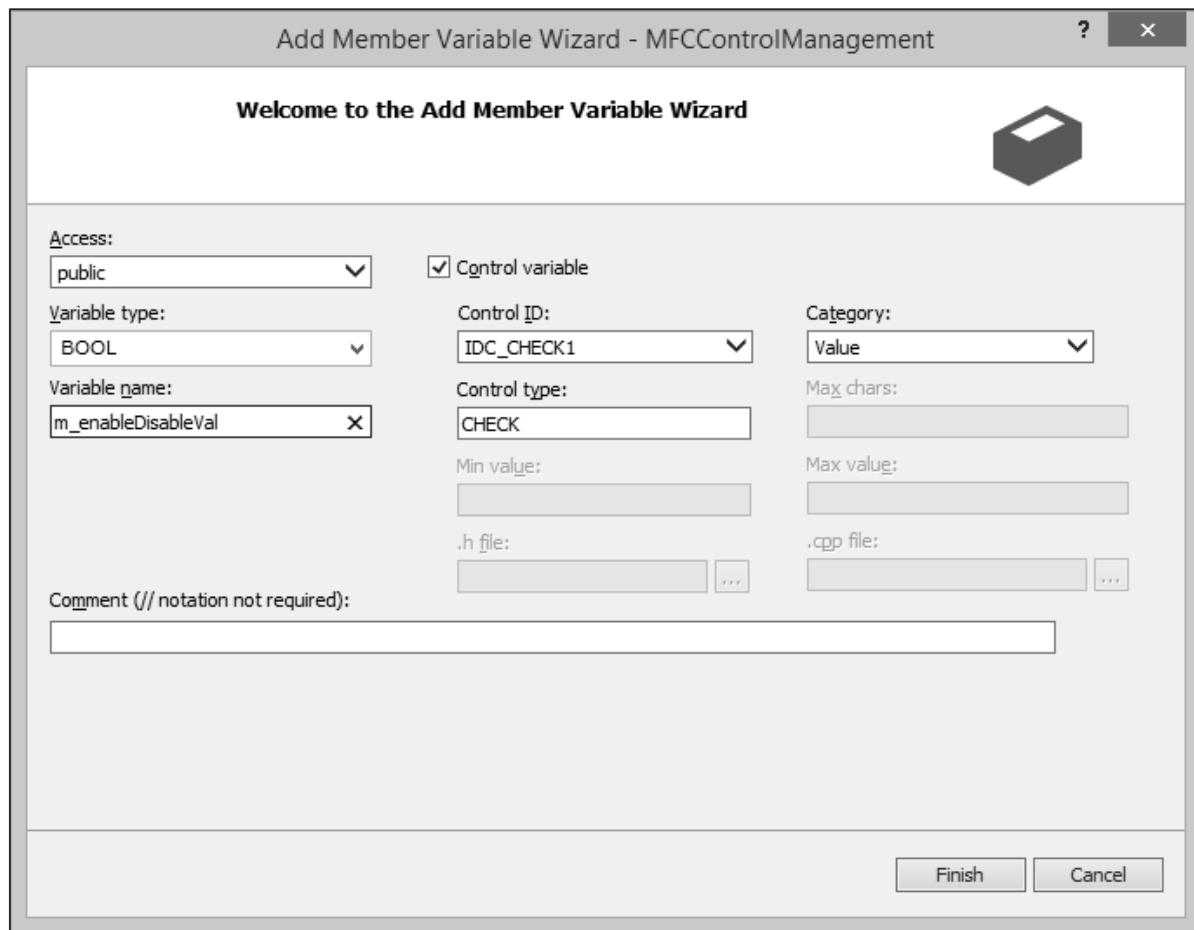
**Step 5:** Add Control Variable of Edit control with the settings as shown in the following snapshot.



**Step 6:** Observe the header file of the dialog class. You can see that these two variables have been added now.

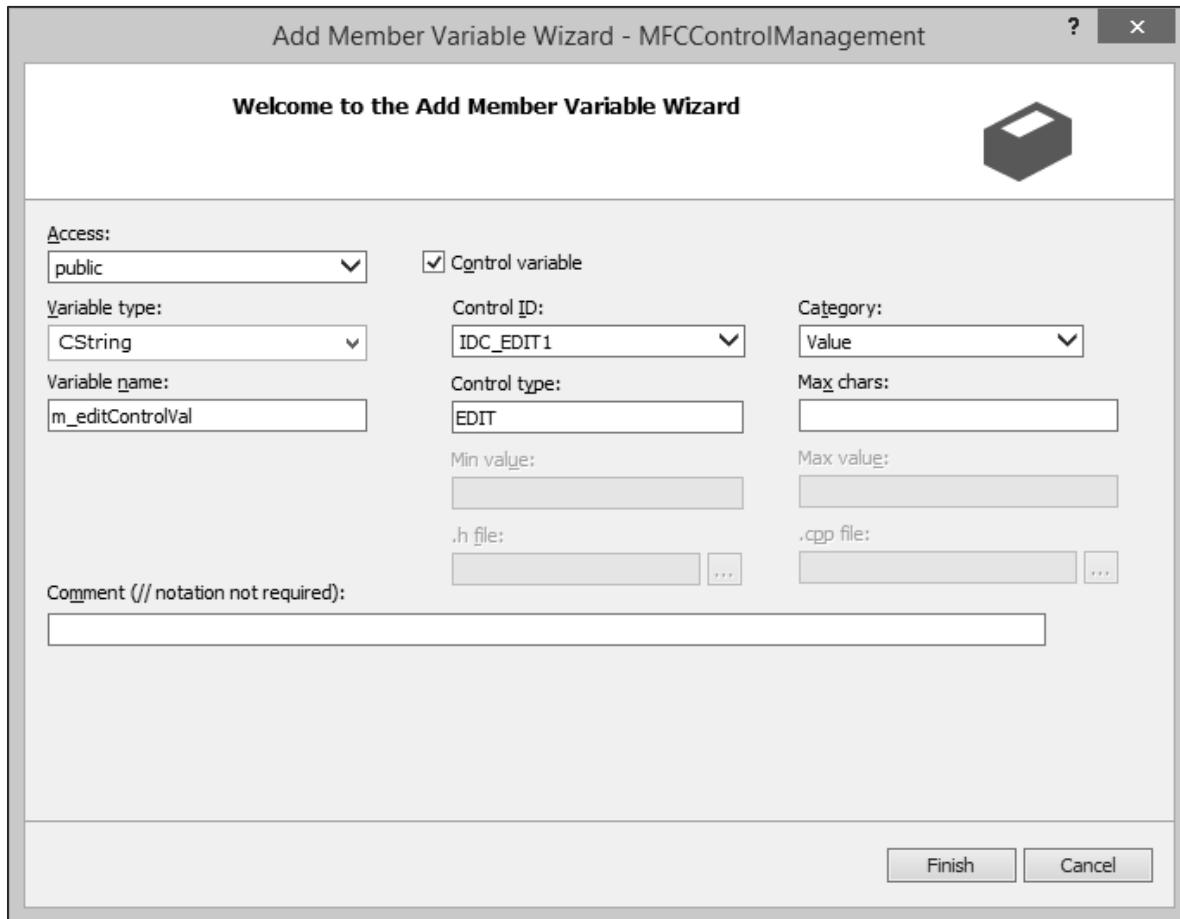
```
CButton m_enableDisableCheck;  
CEdit m_myEditControl;
```

**Step 7:** Right-click on the checkbox and select Add Variable.



**Step 8:** Click Finish to continue.

**Step 9:** Add value Variable for Edit control with the settings as shown in the following snapshot.

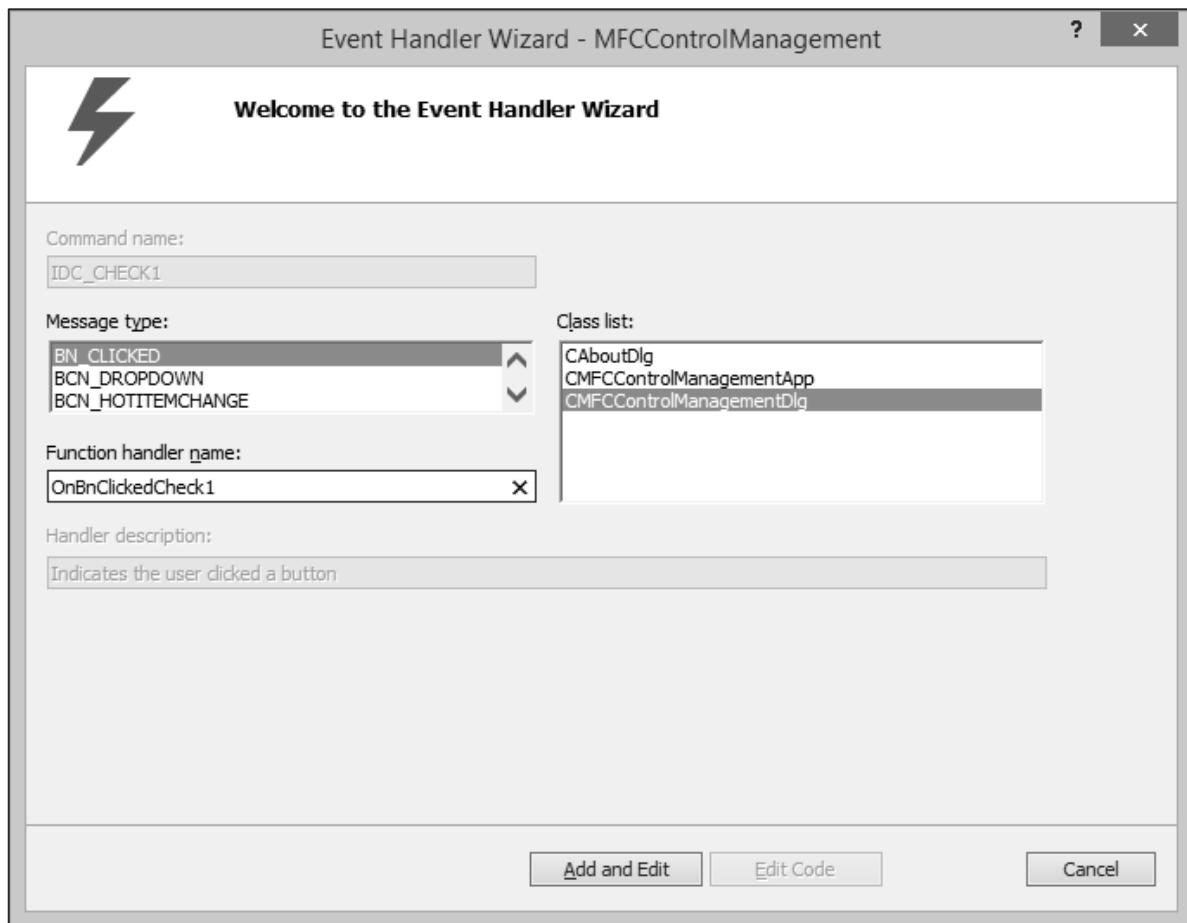


**Step 10:** Observe the header file. You can see that the new variables have been added now.

```
bool m_enableDisableVal;
CString m_editControlVal;
```

**Step 11:** Now we will add event handler for checkbox.

**Step 12:** Right-click the control for which you want to handle the notification event.



**Step 13:** Select the event in the Message type box to add to the class selected in the Class list box.

**Step 14:** Accept the default name in the Function handler name box, or provide a name of your choice.

**Step 15:** Click Add and Edit to add the event handler.

**Step 16:** You can now see the following event added at the end of CMFCCControlManagementDlg.cpp file.

```
void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
}
```

**Step 17:** This enables/disables the edit control when the checkbox is checked/unchecked.

**Step 18:** We have now added the checkbox click event handler. Here is the implementation of event handler for checkbox.

```
void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    if (m_enableDisableVal)
        m_myEditControl.EnableWindow(TRUE);
    else
        m_myEditControl.EnableWindow(FALSE);
}
```

**Step 19:** We need to add the following code to CMFCCControlManagementDlg::OnInitDialog(). When the dialog is created, it will manage these controls.

```
UpdateData(TRUE);
if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
else
    m_myEditControl.EnableWindow(FALSE);
```

**Step 20:** Here is the complete implementation of CMFCCControlManagementDlg.cpp file.

```
// MFCCControlManagementDlg.cpp : implementation file
//

#include "stdafx.h"
#include "MFCCControlManagement.h"
#include "MFCCControlManagementDlg.h"
#include "afxdialogex.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif
```

```

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx
{
public:
    CAboutDlg();

// Dialog Data
#ifndef AFX DESIGN TIME
    enum { IDD = IDD_ABOUTBOX };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

// Implementation
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

CMFCCControlManagementDlg::CMFCCControlManagementDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(IDD_MFCCONTROLMANAGEMENT_DIALOG, pParent)
    , m_enableDisableVal(FALSE)
    , m_editControlVal(_T(""))
{
}

```

```

m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMFCCControlManagementDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_CHECK1, m_enableDisableCheck);
    DDX_Control(pDX, IDC_EDIT1, m_myEditControl);
    DDX_Check(pDX, IDC_CHECK1, m_enableDisableVal);
    DDX_Text(pDX, IDC_EDIT1, m_editControlVal);
}

BEGIN_MESSAGE_MAP(CMFCCControlManagementDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_CHECK1, &CMFCCControlManagementDlg::OnBnClickedCheck1)
END_MESSAGE_MAP()

// CMFCCControlManagementDlg message handlers

BOOL CMFCCControlManagementDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;
}

```

```

bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
    pSysMenu->AppendMenu(MF_SEPARATOR);
    pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
}

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

// TODO: Add extra initialization here
UpdateData(TRUE);
if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
else
    m_myEditControl.EnableWindow(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}

void CMFCControlManagementDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFFF0) == IDM_ABOUTBOX)
    {
        CAaboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialogEx::OnSysCommand(nID, lParam);
    }
}
}

```

```

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CMFCCControlManagementDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

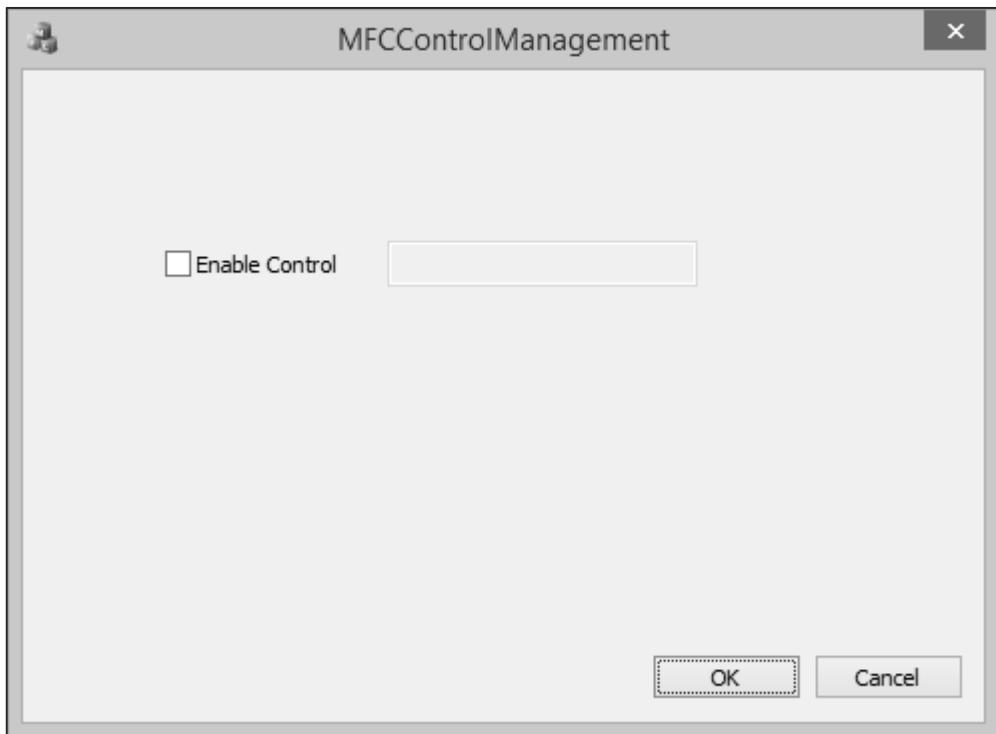
        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialogEx::OnPaint();
    }
}

// The system calls this function to obtain the cursor to display while the
// user drags
// the minimized window.
HCURSOR CMFCCControlManagementDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

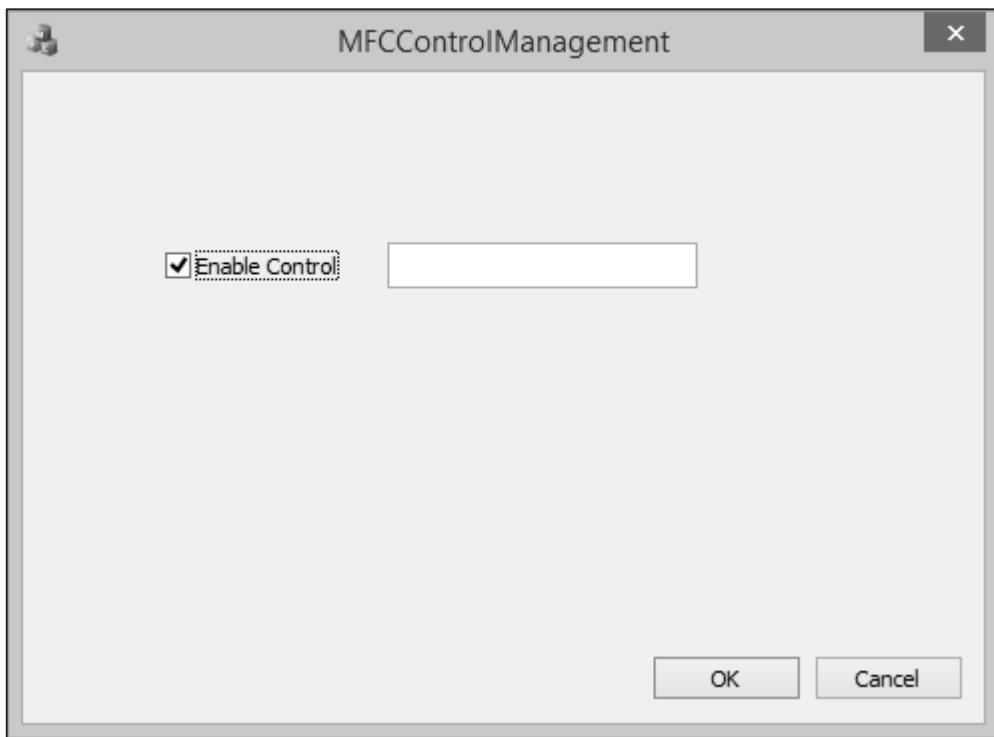
```

```
void CMFCCControlManagementDlg::OnBnClickedCheck1()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    if (m_enableDisableVal)
        m_myEditControl.EnableWindow(TRUE);
    else
        m_myEditControl.EnableWindow(FALSE);
}
```

**Step 21:** When the above code is compiled and executed, you will see the following output. You can now see the checkbox is unchecked by default. This disables the edit control.



**Step 22:** Now when you check the checkbox, the edit control is enabled.



## Image Lists

An **Image List** is a collection of same-sized images, each of which can be referred to by its zero-based index. Image lists are used to efficiently manage large sets of icons or bitmaps. Image lists are represented by **CImageList class**.

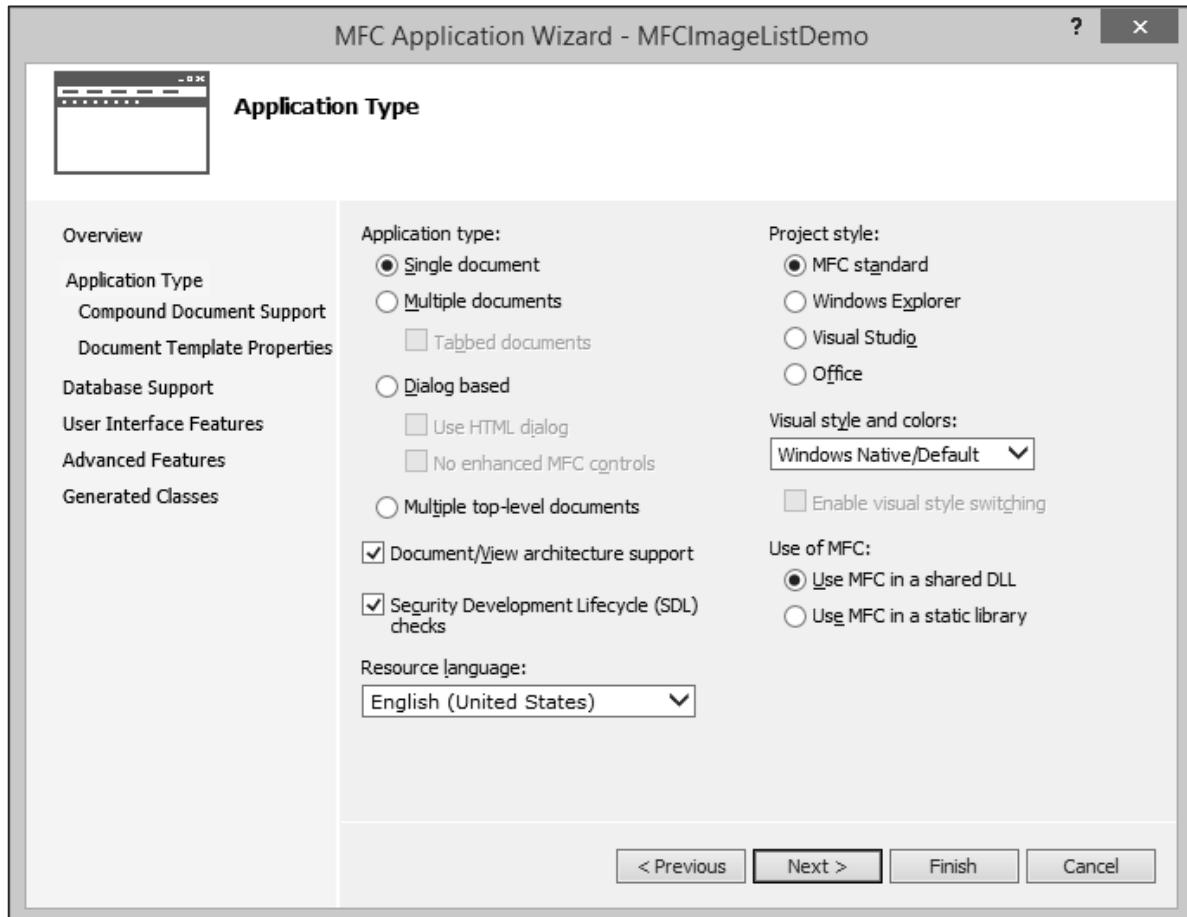
Here is the list of methods in CImageList class:

Name	Description
<b>Add</b>	Adds an image or images to an image list.
<b>Attach</b>	Attaches an image list to a CImageList object.
<b>BeginDrag</b>	Begins dragging an image.
<b>Copy</b>	Copies an image within a CImageList object.
<b>Create</b>	Initializes an image list and attaches it to a CImageList object.
<b>DeleteImageList</b>	Deletes an image list.
<b>DeleteTempMap</b>	Called by the <a href="#">CWinApp</a> idle-time handler to delete any temporary CImageList object created by <b>FromHandle</b> .
<b>Detach</b>	Detaches an image list object from a CImageList object and returns a handle to an image list.
<b>DragEnter</b>	Locks updates during a drag operation and displays the drag image at a specified position.

<b>DragLeave</b>	Unlocks the window and hides the drag image so that the window can be updated.
<b>DragMove</b>	Moves the image that is being dragged during a drag-and-drop operation.
<b>DragShowNolock</b>	Shows or hides the drag image during a drag operation, without locking the window.
<b>Draw</b>	Draws the image that is being dragged during a drag-and-drop operation.
<b>DrawEx</b>	Draws an image list item in the specified device context. The function uses the specified drawing style and blends the image with the specified color.
<b>DrawIndirect</b>	Draws an image from an image list.
<b>EndDrag</b>	Ends a drag operation.
<b>ExtractIcon</b>	Creates an icon based on an image and mask in an image list.
<b>FromHandle</b>	Returns a pointer to a CImageList object when given a handle to an image list. If a CImageList object is not attached to the handle, a temporary CImageList object is created and attached.
<b>FromHandlePermanent</b>	Returns a pointer to a CImageList object when given a handle to an image list. If a CImageList object is not attached to the handle, NULL is returned.
<b>GetBkColor</b>	Retrieves the current background color for an image list.
<b>GetDragImage</b>	Gets the temporary image list that is used for dragging.
<b>GetImageCount</b>	Retrieves the number of images in an image list.
<b>GetImageInfo</b>	Retrieves information about an image.
<b>GetSafeHandle</b>	Retrieves <b>m_hImageList</b> .
<b>Read</b>	Reads an image list from an archive.
<b>Remove</b>	Removes an image from an image list.
<b>Replace</b>	Replaces an image in an image list with a new image.
<b>SetBkColor</b>	Sets the background color for an image list.
<b>SetDragCursorImage</b>	Creates a new drag image.
<b>SetImageCount</b>	Resets the count of images in an image list.

<b>SetOverlayImage</b>	Adds the zero-based index of an image to the list of images to be used as overlay masks.
<b>Write</b>	Writes an image list to an archive.

Let us create a new MFC Application **MFCImageListDemo** with the following settings.



**Step 1:** Add bmp file as a resource in your application.

**Step 2:** In header file of CMFCImageListDemoView class, add the following two variables.

```
CImageList ImageList;
int nImage;
```

**Step 3:** Add the following code in constructor of CMFCImageListDemoView.

```
CMFCImageListDemoView::CMFCImageListDemoView()
{
    // TODO: add construction code here
    ImageList.Create(800, 800, ILC_COLOR, 4, 1);

    CBitmap bmp;
```

```
bmp.LoadBitmap(IDB_BITMAP1);
ImageList.Add(&bmp, RGB(0, 0, 0));

}
```

**Step 4:** Call the CImageList::Draw() method as follows.

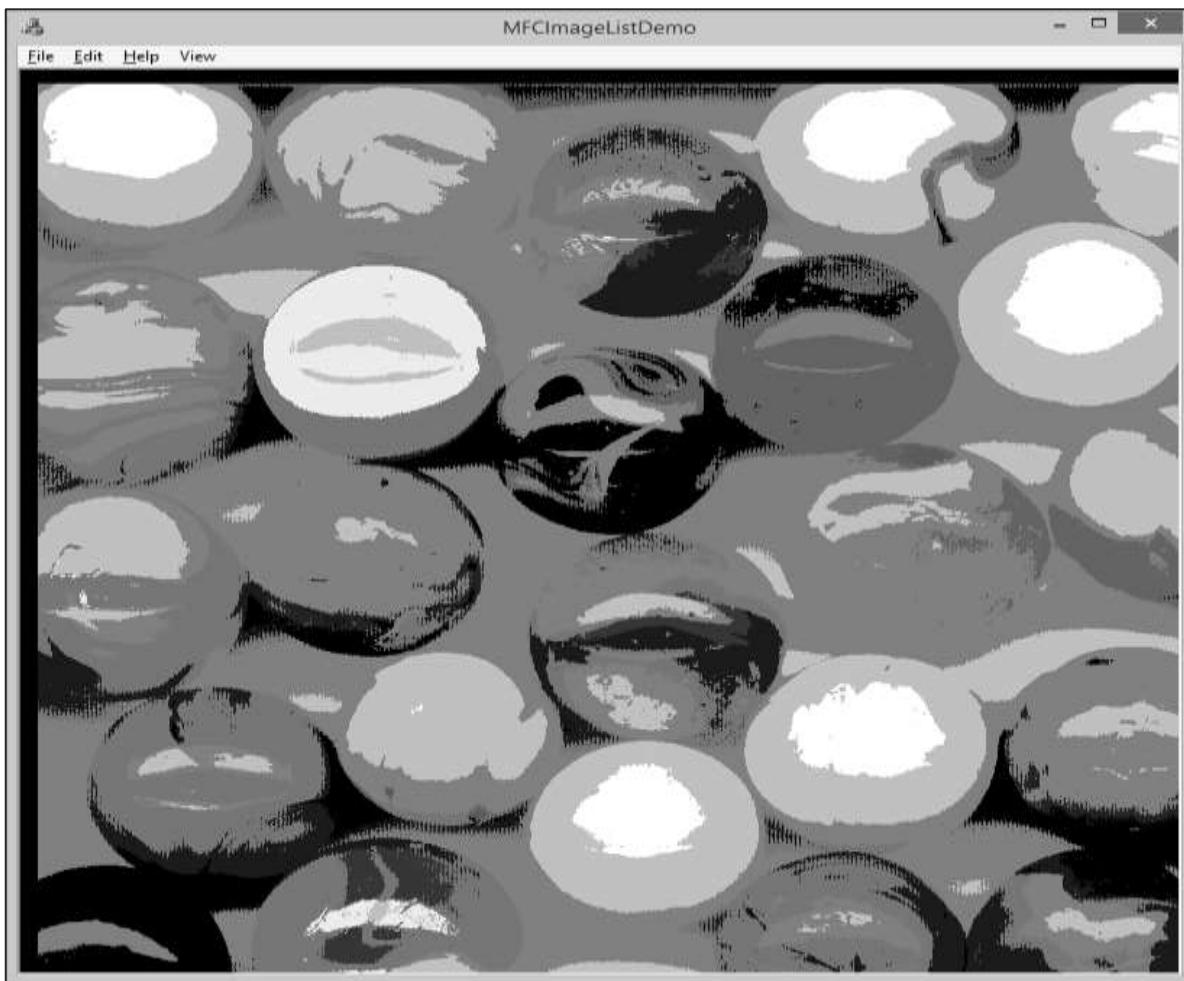
```
void CMFCImageListDemoView::OnDraw(CDC* pDC)
{
    CMFCImageListDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    nImage = 0;
    ImageList.Draw(pDC, nImage, CPoint(0,0), ILD_NORMAL);
    Invalidate();

    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

**Step 5:** When you run this application, you will see the following output.



## Edit Box

An **Edit Box** is a rectangular child window in which the user can enter text. It is represented by **CEdit class**.

Here is the list of methods in CEdit class:

Name	Description
<b>CanUndo</b>	Determines whether an edit-control operation can be undone.
<b>CharFromPos</b>	Retrieves the line and character indexes for the character closest to a specified position.
<b>Clear</b>	Deletes (clears) the current selection (if any) in the edit control.
<b>Copy</b>	Copies the current selection (if any) in the edit control to the Clipboard in <b>CF_TEXT</b> format.
<b>Create</b>	Creates the Windows edit control and attaches it to the CEdit object.
<b>Cut</b>	Deletes (cuts) the current selection (if any) in the edit control and copies the deleted text to the Clipboard in <b>CF_TEXT</b> format.

<b>EmptyUndoBuffer</b>	Resets (clears) the undo flag of an edit control.
<b>FmtLines</b>	Sets the inclusion of soft line-break characters on or off within a multiple-line edit control.
<b>GetCueBanner</b>	Retrieves the text that is displayed as the text cue, or tip, in an edit control when the control is empty and does not have focus.
<b>GetFirstVisibleLine</b>	Determines the topmost visible line in an edit control.
<b>GetHandle</b>	Retrieves a handle to the memory that is currently allocated for a multiple-line edit control.
<b>GetHighlight</b>	Gets the indexes of the starting and ending characters in a range of text that is highlighted in the current edit control.
<b>GetLimitText</b>	Gets the maximum amount of text this <b>CEdit</b> can contain.
<b>GetLine</b>	Retrieves a line of text from an edit control.
<b>GetLineCount</b>	Retrieves the number of lines in a multiple-line edit control.
<b>GetMargins</b>	Gets the left and right margins for this <b>CEdit</b> .
<b>GetModify</b>	Determines whether the contents of an edit control have been modified.
<b>GetPasswordChar</b>	Retrieves the password character displayed in an edit control when the user enters text.
<b>GetRect</b>	Gets the formatting rectangle of an edit control.
<b>GetSel</b>	Gets the first and last character positions of the current selection in an edit control.
<b>HideBalloonTip</b>	Hides any balloon tip associated with the current edit control.
<b>LimitText</b>	Limits the length of the text that the user can enter into an edit control.
<b>LineFromChar</b>	Retrieves the line number of the line that contains the specified character index.
<b>LineIndex</b>	Retrieves the character index of a line within a multiple-line edit control.
<b>LineLength</b>	Retrieves the length of a line in an edit control.
<b>LineScroll</b>	Scrolls the text of a multiple-line edit control.
<b>Paste</b>	Inserts the data from the Clipboard into the edit control at the current cursor position. Data is inserted only if the Clipboard contains data in CF_TEXT format.
<b>PosFromChar</b>	Retrieves the coordinates of the upper-left corner of a specified character index.
<b>ReplaceSel</b>	Replaces the current selection in an edit control with the specified text.
<b>SetCueBanner</b>	Sets the text that is displayed as the text cue, or tip, in an edit control when the control is empty and does not have focus.

<b>SetHandle</b>	Sets the handle to the local memory that will be used by a multiple-line edit control.
<b>SetHighlight</b>	Highlights a range of text that is displayed in the current edit control.
<b>SetLimitText</b>	Sets the maximum amount of text this CEdit can contain.
<b>SetMargins</b>	Sets the left and right margins for this CEdit.
<b>SetModify</b>	Sets or clears the modification flag for an edit control.
<b>SetPasswordChar</b>	Sets or removes a password character displayed in an edit control when the user enters text.
<b>SetReadOnly</b>	Sets the read-only state of an edit control.
<b>SetRect</b>	Sets the formatting rectangle of a multiple-line edit control and updates the control.
<b>SetRectNP</b>	Sets the formatting rectangle of a multiple-line edit control without redrawing the control window.
<b>SetSel</b>	Selects a range of characters in an edit control.
<b>SetTabStops</b>	Sets the tab stops in a multiple-line edit control.
<b>ShowBalloonTip</b>	Displays a balloon tip that is associated with the current edit control.
<b>Undo</b>	Reverses the last edit-control operation.

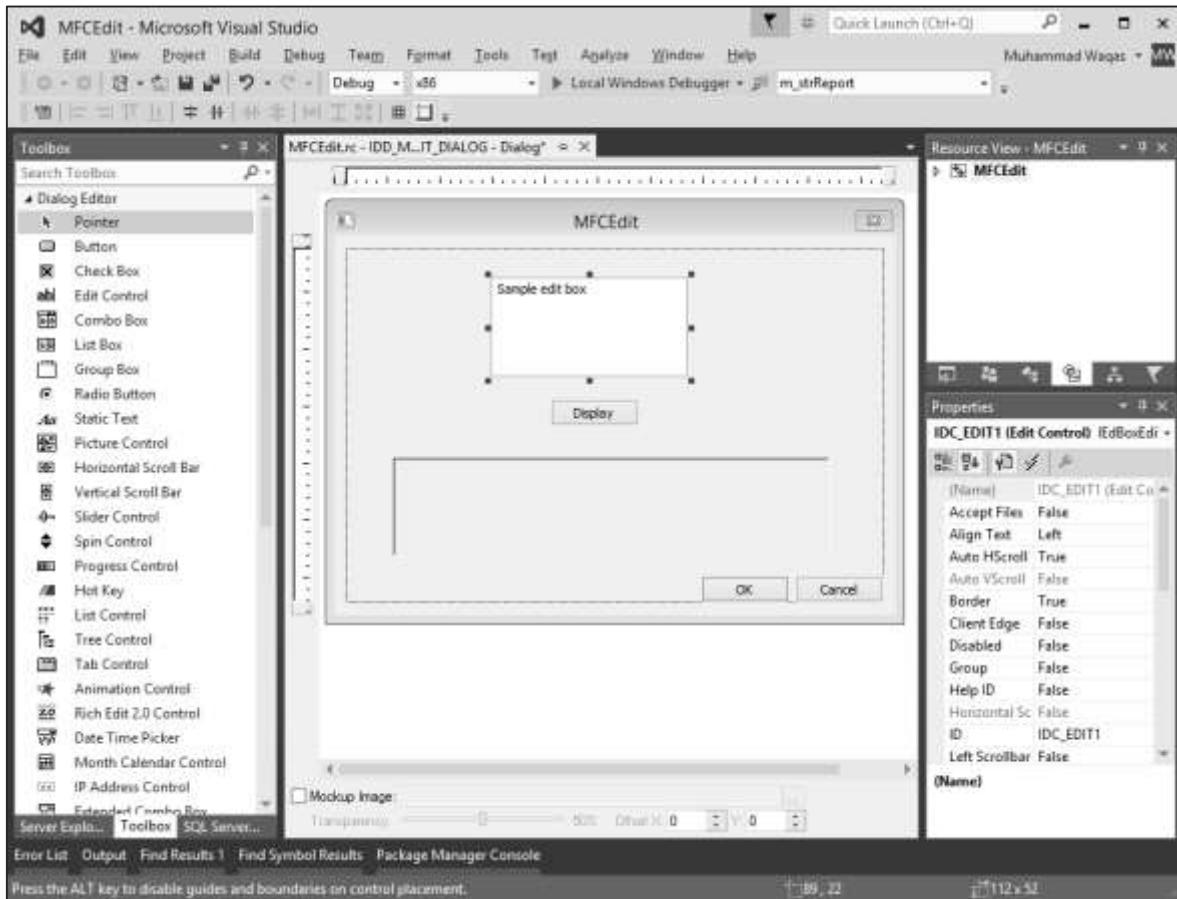
Here is the list of messages mapping for Edit Control.

Message	Map entry	Description
<b>EN_CHANGE</b>	ON_EN_CHANGE( <id>, <memberFxn> )	The user has taken an action that may have altered text in an edit control
<b>EN_ERRSPACE</b>	ON_EN_ERRSPACE( <id>, <memberFxn> )	The edit control cannot allocate enough memory to meet a specific request.
<b>EN_HSCROLL</b>	ON_EN_HSCROLL( <id>, <memberFxn> )	The user clicks an edit control's horizontal scroll bar. The parent window is notified before the screen is updated.
<b>EN_KILLFOCUS</b>	ON_EN_KILLFOCUS( <id>, <memberFxn> )	The edit control loses the input focus.
<b>MAXTEXT</b>	ON_EN_MAXTEXT( <id>, <memberFxn> )	The current insertion has exceeded the specified number of characters for the

		edit control and has been truncated.
<b>EN_SETFOCUS</b>	ON_EN_SETFOCUS( <id>, <memberFxn> )	Sent when an edit control receives the input focus.
<b>EN_UPDATE</b>	ON_EN_UPDATE( <id>, <memberFxn> )	The edit control is about to display altered text. Sent after the control has formatted the text but before it screens the text so that the window size can be altered, if necessary.
<b>EN_VSCROLL</b>	ON_EN_VSCROLL( <id>, <memberFxn> )	The user clicks an edit control's vertical scroll bar. The parent window is notified before the screen is updated.

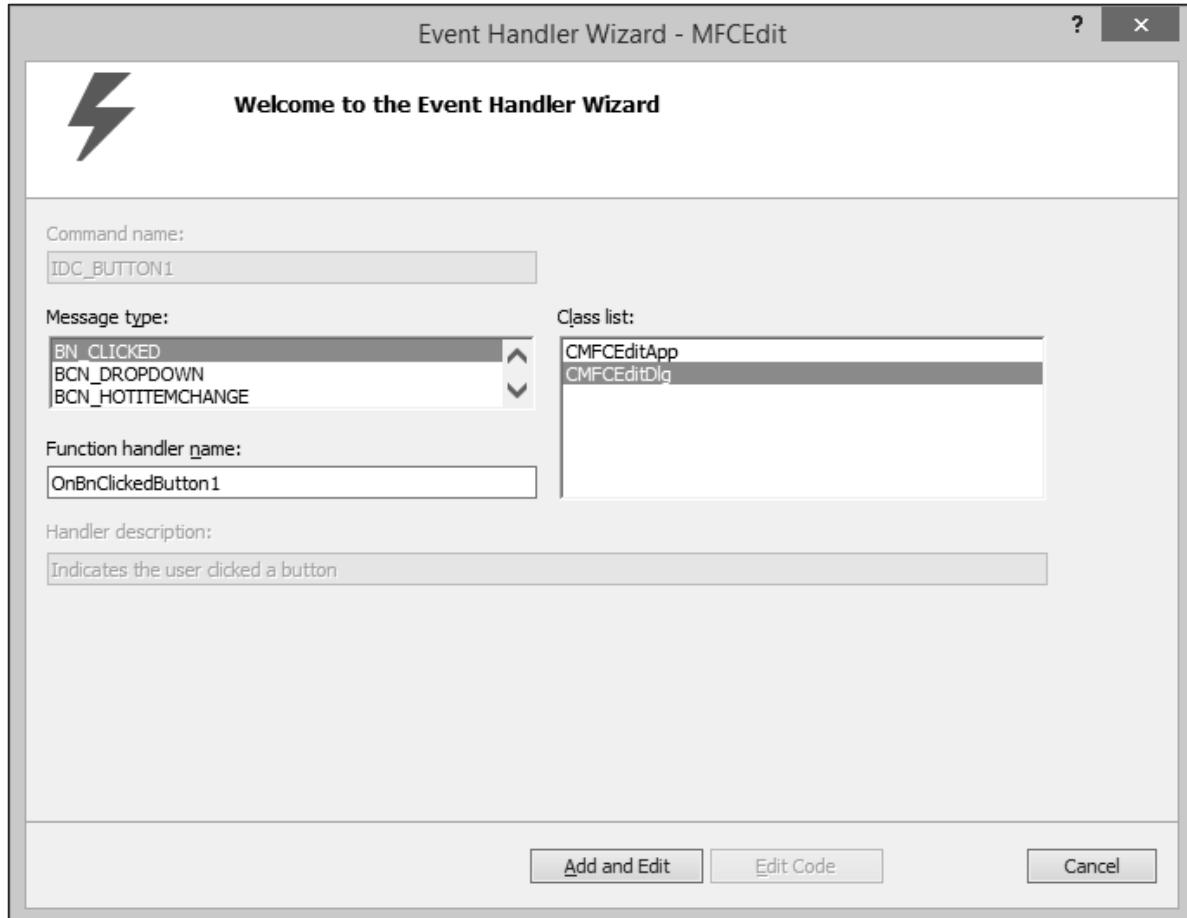
Let us into a simple example by creating a new MFC dialog based project.

**Step 1:** Remove the caption of Static Text control and drag one button and one Edit control.



**Step 2:** Add a control variable `m_editCtrl` for edit control and value variable `m_strTextCtrl` for Static text control.

**Step 3:** Add the event handler for button click event.

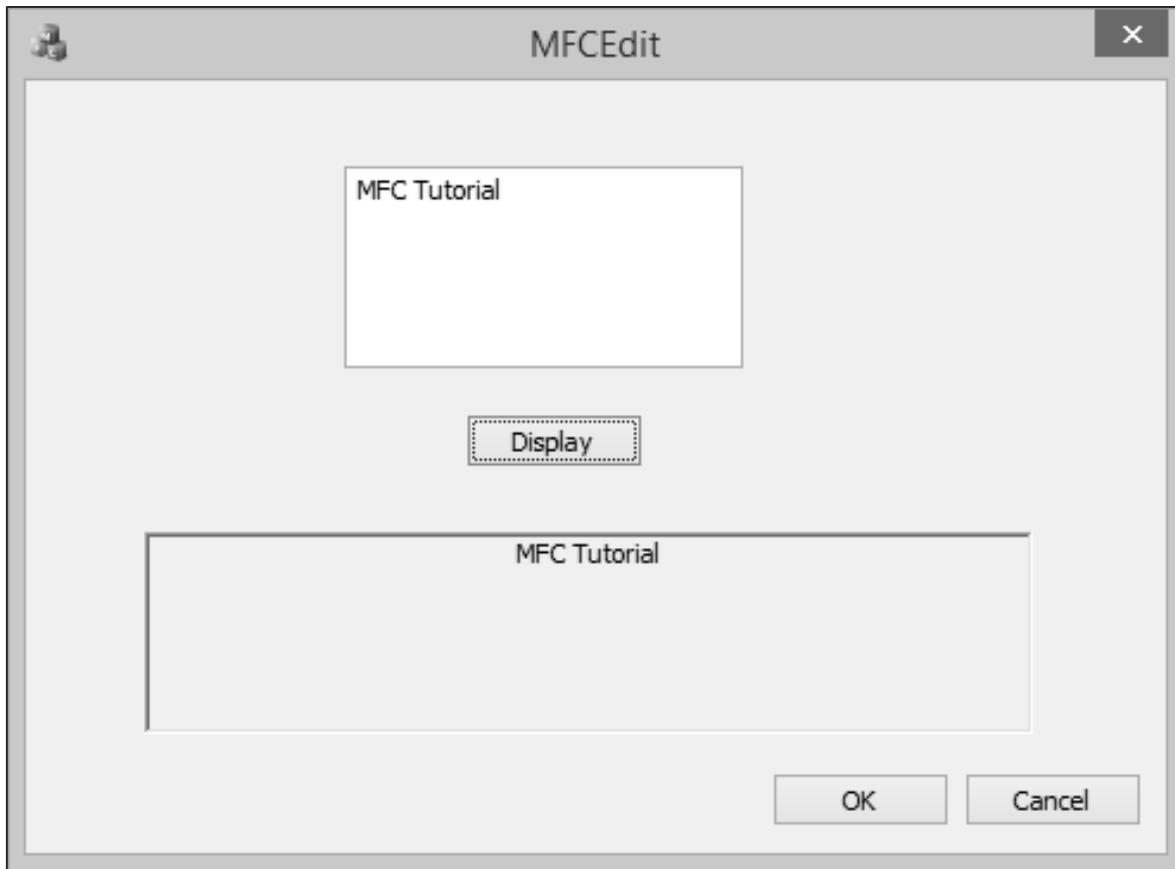


**Step 4:** Here is the implementation of event handler for button click event.

```
void CMFCEditDlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here
    CString str = _T("");
    m_editCtrl.GetWindowTextW(str);

    if (!str.IsEmpty())
        m_strTextCtrl = str;
    else
        m_strTextCtrl = _T("Write Something");
    UpdateData(FALSE);
}
```

**Step 5:** When the above code is compiled and executed, you will see the following.



**Step 6:** When you write text in the edit control and click Display, it will update that text on Static Text Control.

## Rich Edit

A **Rich Edit** Control is a window in which the user can enter and edit text. The text can be assigned character and paragraph formatting, and can include embedded OLE objects. It is represented by **CRichEditCtrl class**.

Here is the list of methods in CRichEditCtrl class:

Name	Description
<b>CanPaste</b>	Determines if the contents of the Clipboard can be pasted into this rich edit control.
<b>CanRedo</b>	Determines whether there are any actions in the controls redo queue.
<b>CanUndo</b>	Determines if an editing operation can be undone.

<b>CharFromPos</b>	Retrieves information about the character closest to a specified point in the client area of an edit control.
<b>Clear</b>	Clears the current selection.
<b>Copy</b>	Copies the current selection to the Clipboard.
<b>Create</b>	Creates the Windows rich edit control and associates it with this CRichEditCtrl object.
<b>CreateEx</b>	Creates the Windows rich edit control with the specified extended Windows styles and associates it with this CRichEditCtrl object.
<b>Cut</b>	Cuts the current selection to the Clipboard.
<b>DisplayBand</b>	Displays a portion of the contents of this CRichEditCtrl object.
<b>EmptyUndoBuffer</b>	Resets (clears) the undo flag of this CRichEditCtrl object.
<b>FindText</b>	Locates text within this CRichEditCtrl object.
<b>FindWordBreak</b>	Finds the next word break before or after the specified character position, or retrieves information about the character at that position.
<b>FormatRange</b>	Formats a range of text for the target output device.
<b>GetCharPos</b>	Determines the location of a given character within this CRichEditCtrl object.
<b>GetDefaultCharFormat</b>	Retrieves the current default character formatting attributes in this CRichEditCtrl object.
<b>GetEventMask</b>	Retrieves the event mask for this CRichEditCtrl object.
<b>GetFirstVisibleLine</b>	Determines the topmost visible line in this CRichEditCtrl object.
<b>GetIRichEditOle</b>	Retrieves a pointer to the <b>IRichEditOle</b> interface for this rich edit control.
<b>GetLimitText</b>	Gets the limit on the amount of text a user can enter into this CRichEditCtrl object.
<b>GetLine</b>	Retrieves a line of text from this CRichEditCtrl object.
<b>GetLineCount</b>	Retrieves the number of lines in this CRichEditCtrl object.
<b>GetModify</b>	Determines if the contents of this CRichEditCtrl object have changed since the last save.

<b>GetOptions</b>	Retrieves the rich edit control options.
<b>GetParaFormat</b>	Retrieves the paragraph formatting attributes in the current selection in this CRichEditCtrl object.
<b>GetPunctuation</b>	Retrieves the current punctuation characters for the rich edit control. This message is available only in Asian-language versions of the operating system.
<b>GetRect</b>	Retrieves the formatting rectangle for this CRichEditCtrl object.
<b>GetRedoName</b>	Retrieves the type of the next action, if any, in the control's redo queue.
<b>GetSel</b>	Gets the starting and ending positions of the current selection in this CRichEditCtrl object.
<b>GetSelectionCharFormat</b>	Retrieves the character formatting attributes in the current selection in this CRichEditCtrl object.
<b>GetSelectionType</b>	Retrieves the type of contents in the current selection in this CRichEditCtrl object.
<b>GetSelText</b>	Gets the text of the current selection in this CRichEditCtrl object
<b>GetTextLength</b>	Retrieves the length of the text, in characters, in this CRichEditCtrl object. Does not include the terminating null character.
<b>GetTextLengthEx</b>	Retrieves the number of characters or bytes in the rich edit view. Accepts a list of flags to indicate the method of determining length of the text in a rich edit control
<b>GetTextMode</b>	Retrieves the current text mode and undo level of a rich edit control.
<b>GetTextRange</b>	Retrieves the specified range of text.
<b>GetUndoName</b>	Retrieves the type of the next undo action, if any.
<b>GetWordWrapMode</b>	Retrieves the current word wrapping and word breaking options for the rich edit control. This message is available only

	in Asian-language versions of the operating system.
<b>HideSelection</b>	Shows or hides the current selection.
<b>LimitText</b>	Limits the amount of text a user can enter into the CRichEditCtrl object.
<b>LineFromChar</b>	Determines which line contains the given character.
<b>LineIndex</b>	Retrieves the character index of a given line in this CRichEditCtrl object.
<b>LineLength</b>	Retrieves the length of a given line in this CRichEditCtrl object.
<b>LineScroll</b>	Scrolls the text in this CRichEditCtrl object.
<b>Paste</b>	Inserts the contents of the Clipboard into this rich edit control.
<b>PasteSpecial</b>	Inserts the contents of the Clipboard into this rich edit control in the specified data format.
<b>PosFromChar</b>	Retrieves the client area coordinates of a specified character in an edit control.
<b>Redo</b>	Redoes the next action in the control's redo queue.
<b>ReplaceSel</b>	Replaces the current selection in this CRichEditCtrl object with specified text.
<b>RequestResize</b>	Forces this CRichEditCtrl object to send request resize notifications.
<b>SetAutoURLDetect</b>	Indicates if the auto URL detection is active in a rich edit control.
<b>SetBackgroundColor</b>	Sets the background color in this CRichEditCtrl object.
<b>SetDefaultCharFormat</b>	Sets the current default character formatting attributes in this CRichEditCtrl object.
<b>SetEventMask</b>	Sets the event mask for this CRichEditCtrl object.
<b>SetModify</b>	Sets or clears the modification flag for this CRichEditCtrl object.
<b>SetOLECallback</b>	Sets the <b>IRichEditOleCallback</b> COM object for this rich edit control.
<b>SetOptions</b>	Sets the options for this CRichEditCtrl object.
<b>SetParaFormat</b>	Sets the paragraph formatting attributes in the current

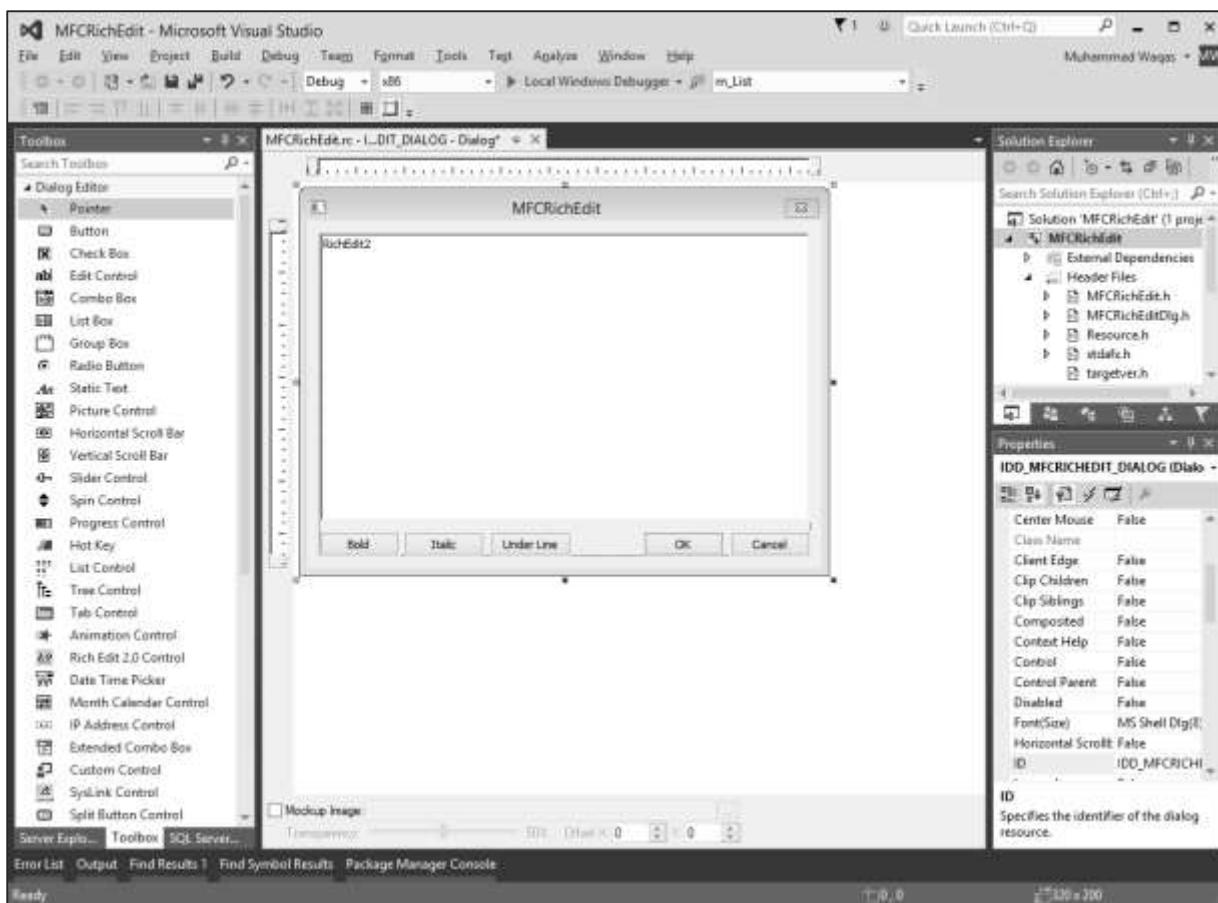
	selection in this CRichEditCtrl object.
<b>SetPunctuation</b>	Sets the punctuation characters for a rich edit control. This message is available only in Asian-language versions of the operating system.
<b>SetReadOnly</b>	Sets the read-only option for this CRichEditCtrl object.
<b>SetRect</b>	Sets the formatting rectangle for this CRichEditCtrl object.
<b>SetSel</b>	Sets the selection in this CRichEditCtrl object.
<b>SetSelectionCharFormat</b>	Sets the character formatting attributes in the current selection in this CRichEditCtrl object.
<b>SetTargetDevice</b>	Sets the target output device for this CRichEditCtrl object.
<b>SetTextMode</b>	Sets the text mode or undo level of a rich edit control. The message fails if the control contains text.
<b>SetUndoLimit</b>	Sets the maximum number of actions that can be stored in the undo queue.
<b>SetWordCharFormat</b>	Sets the character formatting attributes in the current word in this CRichEditCtrl object.
<b>SetWordWrapMode</b>	Sets the word-wrapping and word-breaking options for the rich edit control. This message is available only in Asian-language versions of the operating system.
<b>StopGroupTyping</b>	Stops the control from collecting additional typing actions into the current undo action. The control stores the next typing action, if any, into a new action in the undo queue.
<b>StreamIn</b>	Inserts text from an input stream into this CRichEditCtrl object.
<b>StreamOut</b>	Stores text from this CRichEditCtrl object into an output stream.
<b>Undo</b>	Reverses the last editing operation.

Here is the list of messages mapping for Rich Edit Control:

<b>Message</b>	<b>Map entry</b>	<b>Description</b>
<b>EN_CHANGE</b>	ON_EN_CHANGE( <id>, <memberFxn> )	The user has taken an action that may have altered text in an edit control.
<b>EN_ERRSPACE</b>	ON_EN_ERRSPACE( <id>, <memberFxn> )	The edit control cannot allocate enough memory to meet a specific request.
<b>EN_HSCROLL</b>	ON_EN_HSCROLL( <id>, <memberFxn> )	The user clicks an edit control's horizontal scroll bar. The parent window is notified before the screen is updated.
<b>EN_KILLFOCUS</b>	ON_EN_KILLFOCUS( <id>, <memberFxn> )	The edit control loses the input focus.
<b>MAXTEXT</b>	ON_EN_MAXTEXT( <id>, <memberFxn> )	The current insertion has exceeded the specified number of characters for the edit control and has been truncated.
<b>EN_SETFOCUS</b>	ON_EN_SETFOCUS( <id>, <memberFxn> )	Sent when an edit control receives the input focus.
<b>EN_UPDATE</b>	ON_EN_UPDATE( <id>, <memberFxn> )	The edit control is about to display altered text. Sent after the control has formatted the text but before it screens the text so that the window size can be altered, if necessary.
<b>EN_VSCROLL</b>	ON_EN_VSCROLL( <id>, <memberFxn> )	The user clicks an edit control's vertical scroll bar. The parent window is notified before the screen is updated.

Let us into a simple example by creating a new MFC dialog based application.

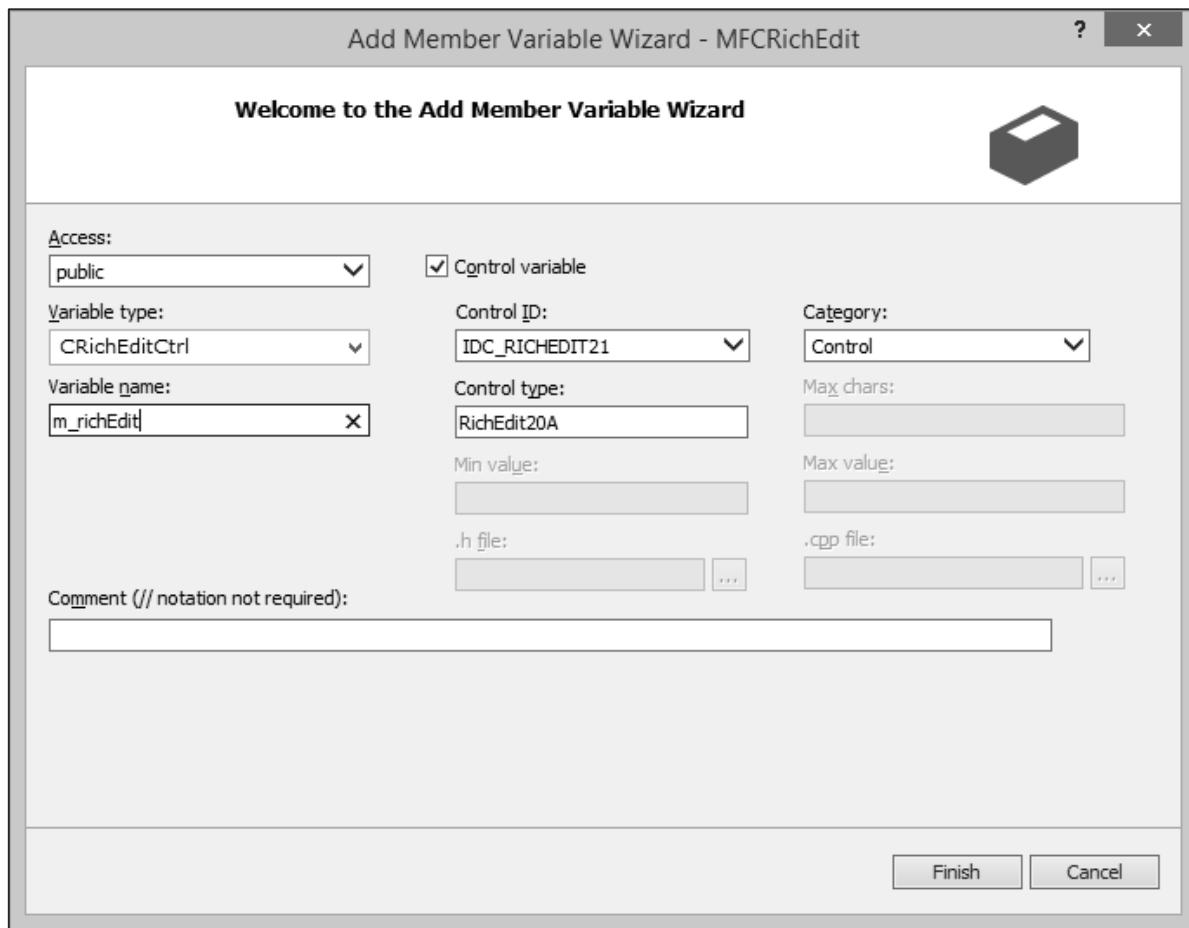
**Step 1:** Delete the TODO line and drag one Rich Edit Control and three buttons from the Toolbox.



**Step 2:** Change the Caption of these three buttons from Bold, Italic and Under Line to IDC\_BUTTON\_BOLD, IDC\_BUTTON\_ITALIC and IDC\_BUTTON\_UNDERLINE respectively.

**Step 3:** Set the following properties to True: Multiline, Want Return, Vertical Scroll.

**Step 4:** Add the control variable m\_richEdit for Rich Edit Control.



**Step 5:** Go to the CMFCRichEditApp and call the **::AfxInitRichEdit2()** in CMFCRichEditApp::InitInstance() function as shown in the following code.

```
BOOL CMFCRichEditApp::InitInstance()
{
//TODO: call AfxInitRichEdit2() to initialize richedit2 library.
    // InitCommonControlsEx() is required on Windows XP if an application
    // manifest specifies use of ComCtl32.dll version 6 or later to enable
    // visual styles. Otherwise, any window creation will fail.
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // Set this to include all the common control classes you want to use
    // in your application.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    ::AfxInitRichEdit2();
}
```

```
CWinApp::InitInstance();

AfxEnableControlContainer();

// Create the shell manager, in case the dialog contains
// any shell tree view or shell list view controls.
CShellManager *pShellManager = new CShellManager;

// Activate "Windows Native" visual manager for enabling themes in MFC
controls
CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindow
s));

// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need
// Change the registry key under which our settings are stored
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization
SetRegistryKey(_T("Local AppWizard-Generated Applications"));

CMFCRichEditDlg dlg;
m_pMainWnd = &dlg;
INT_PTR nResponse = dlg.DoModal();
if (nResponse == IDOK)
{
    // TODO: Place code here to handle when the dialog is
    // dismissed with OK
}
else if (nResponse == IDCANCEL)
{
    // TODO: Place code here to handle when the dialog is
    // dismissed with Cancel
}
else if (nResponse == -1)
{
```

```

        TRACE(traceAppMsg, 0, "Warning: dialog creation failed, so
application is terminating unexpectedly.\n");

        TRACE(traceAppMsg, 0, "Warning: if you are using MFC controls on
the dialog, you cannot #define _AFX_NO_MFC_CONTROLS_IN_DIALOGS.\n");

    }

// Delete the shell manager created above.

if (pShellManager != NULL)
{
    delete pShellManager;
}

// Since the dialog has been closed, return FALSE so that we exit the
// application, rather than start the application's message pump.

return FALSE;
}

```

**Step 6:** Add the Click event handler for all the three buttons. Here is the implementation for these events.

```

void CMFCRichEditDlg::OnBnClickedButtonBold()
{
    // TODO: Add your control notification handler code here

    CHARFORMAT Cfm;

    m_richEdit.GetSelectionCharFormat(Cfm);

    Cfm.cbSize = sizeof(CHARFORMAT);
    Cfm.dwMask = CFM_BOLD;
    Cfm.dwEffects ^= CFE_BOLD;

    m_richEdit.SetSelectionCharFormat(Cfm);
    m_richEdit.SetFocus();
}

void CMFCRichEditDlg::OnBnClickedButtonItalic()
{
    // TODO: Add your control notification handler code here
}

```

```
CHARFORMAT Cfm;

m_richEdit.GetSelectionCharFormat(Cfm);

Cfm.cbSize = sizeof(CHARFORMAT);
Cfm.dwMask = CFM_ITALIC;
Cfm.dwEffects ^= CFE_ITALIC;

m_richEdit.SetSelectionCharFormat(Cfm);
m_richEdit.SetFocus();

}

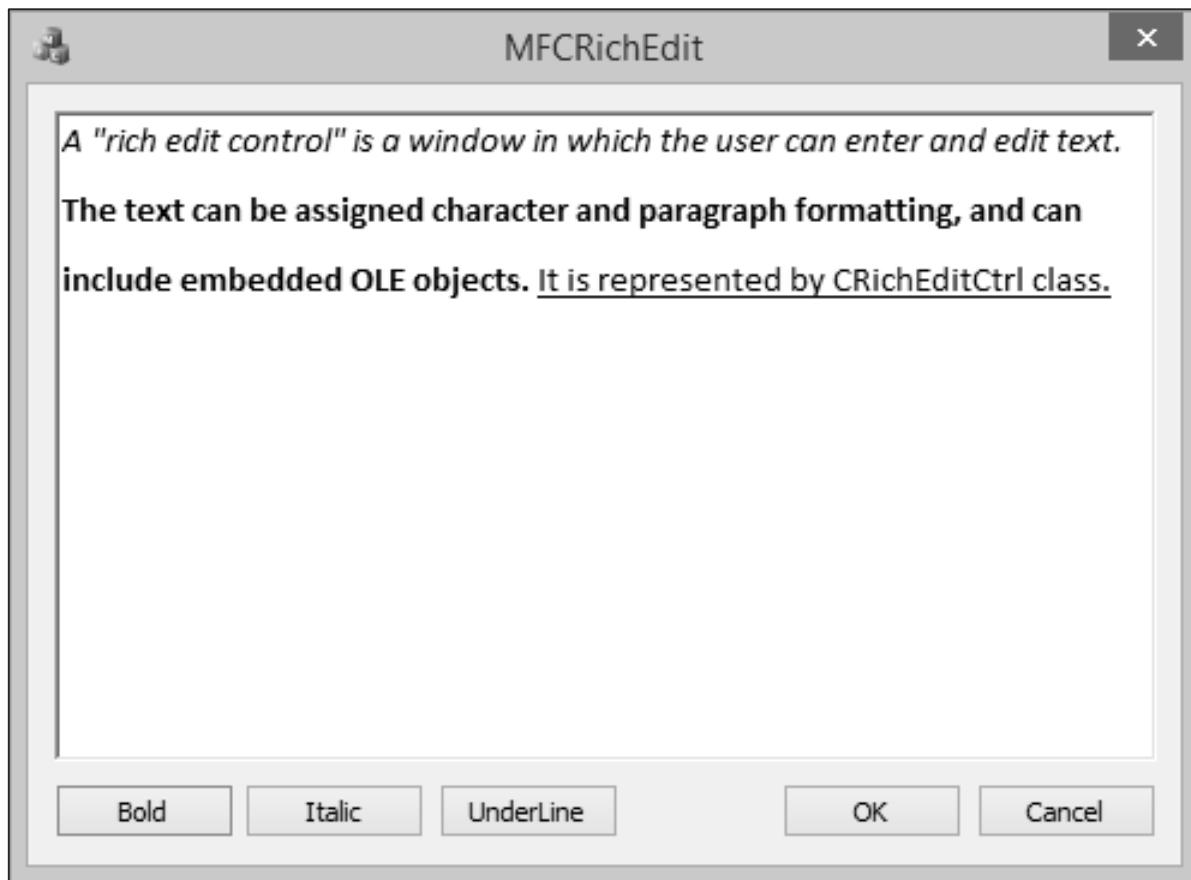
void CMFCRichEditDlg::OnBnClickedButtonUnderline()
{
    // TODO: Add your control notification handler code here
    CHARFORMAT Cfm;

    m_richEdit.GetSelectionCharFormat(Cfm);

    Cfm.cbSize = sizeof(CHARFORMAT);
    Cfm.dwMask = CFM_UNDERLINE;
    Cfm.dwEffects ^= CFE_UNDERLINE;

    m_richEdit.SetSelectionCharFormat(Cfm);
    m_richEdit.SetFocus();
}
```

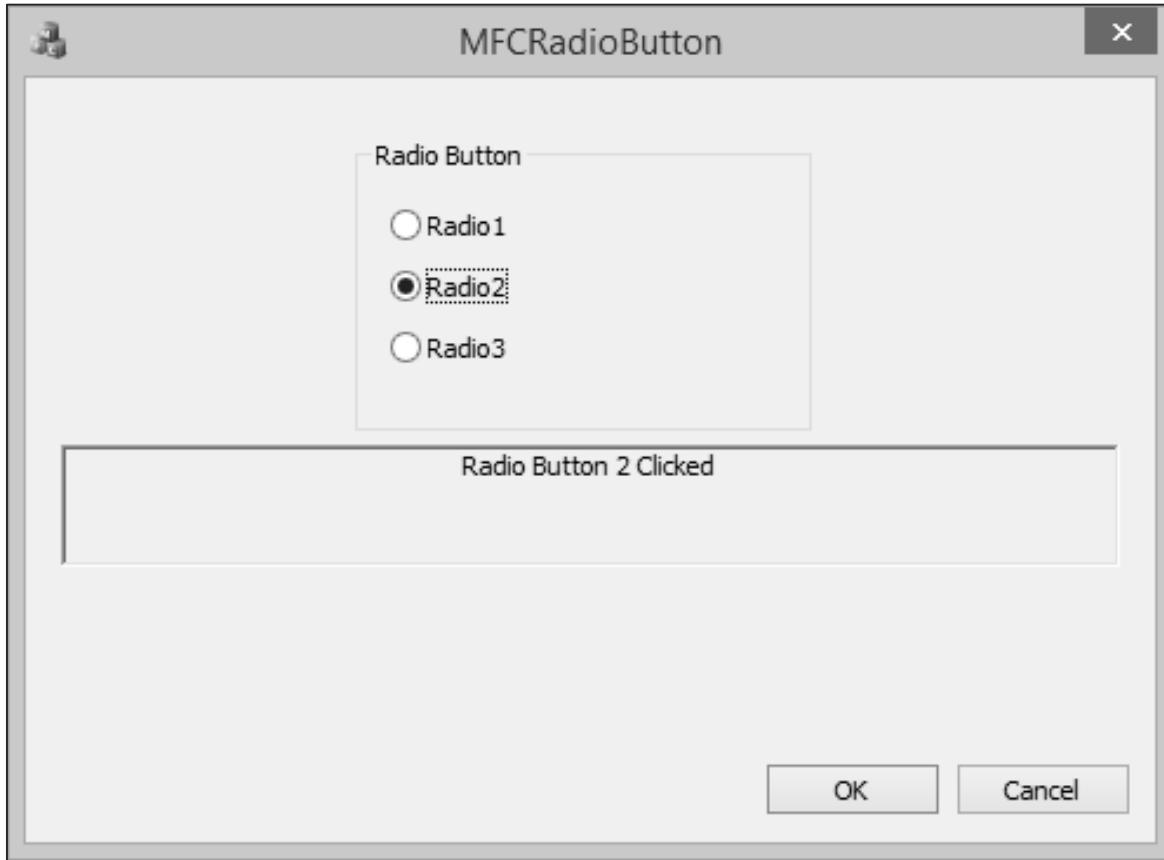
**Step 7:** When the above code is compiled and executed, you will see the following output. Now enter text and change its formatting by selecting the text and then click on any of the three buttons.



## Group Box

A **group box** is a static control used to set a visible or programmatic group of controls. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner.

In the following dialog box, the Group box contains three radio buttons inside.



## Spin Button

A **Spin Button** Control (also known as an up-down control) is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. It is represented by **CSpinButtonCtrl** class.

Here is the list of methods in CSpinButtonCtrl class:

Name	Description
<b>Create</b>	Creates a spin button control and attaches it to a CSpinButtonCtrl object.
<b>CreateEx</b>	Creates a spin button control with the specified Windows extended styles and attaches it to a CSpinButtonCtrl object.
<b>GetAccel</b>	Retrieves acceleration information for a spin button control.
<b>GetBase</b>	Retrieves the current base for a spin button control.
<b>GetBuddy</b>	Retrieves a pointer to the current buddy window.
<b>GetPos</b>	Retrieves the current position of a spin button control.
<b>GetRange</b>	Retrieves the upper and lower limits (range) for a spin button control.
<b>SetAccel</b>	Sets the acceleration for a spin button control.
<b>SetBase</b>	Sets the base for a spin button control.
<b>SetBuddy</b>	Sets the buddy window for a spin button control.
<b>SetPos</b>	Sets the current position for the control.

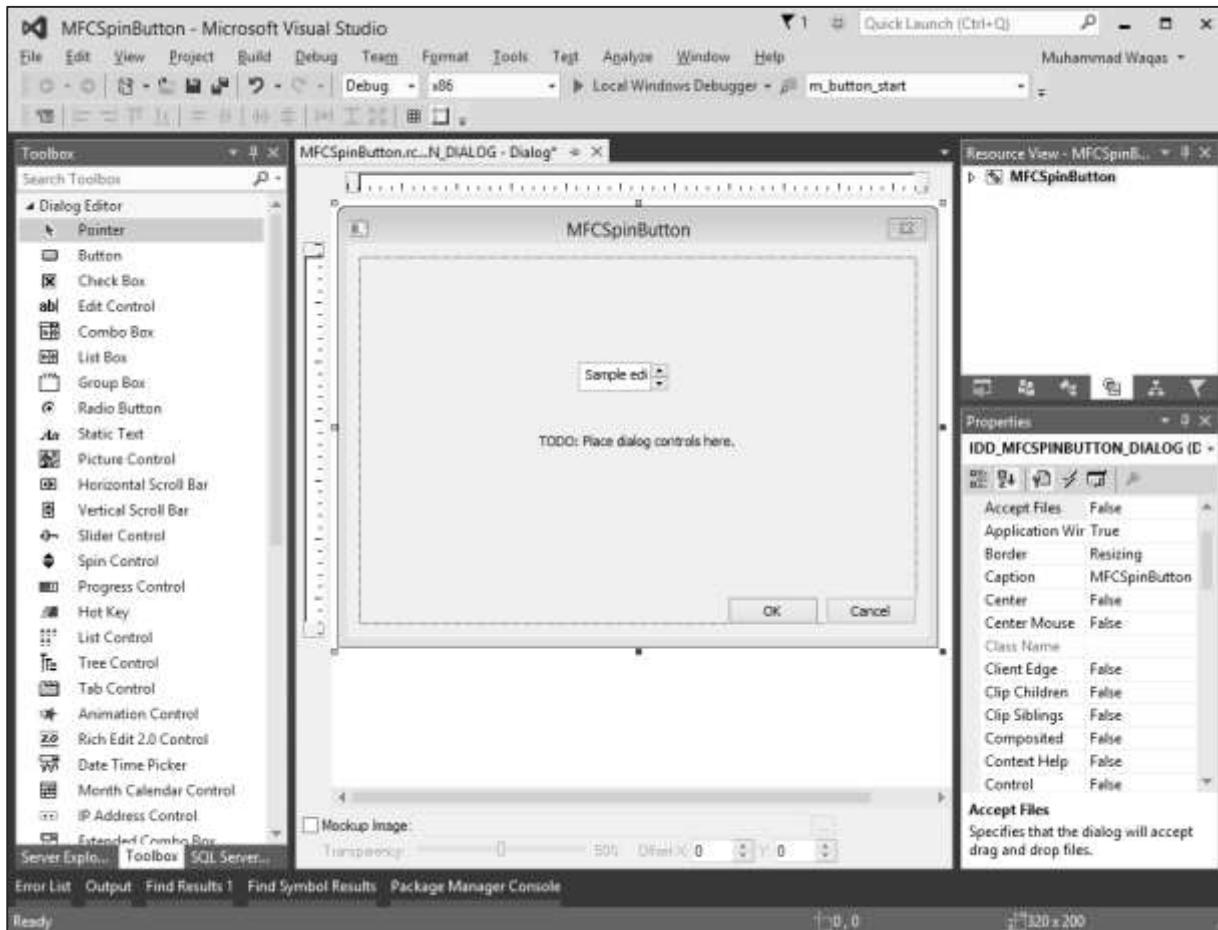
<b>SetRange</b>	Sets the upper and lower limits (range) for a spin button control.
-----------------	--

Here is the list of messages mapping for Spin Button control.

Message	Map entry	Description
<b>BN_CLICKED</b>	ON_BN_CLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is clicked.
<b>BN_DISABLE</b>	ON_BN_DISABLE( <id>, <memberFxn> )	The framework calls this member function when the button is disabled.
<b>BN_DOUBLECLICKED</b>	ON_BN_DOUBLECLICKED( <id>, <memberFxn> )	The framework calls this member function when the button is double clicked.
<b>BN_PAINT</b>	ON_BN_PAINT( <id>, <memberFxn> )	The framework calls this member function when an application makes a request to repaint a button.

Let us look into a simple example of Spin button by creating a new MFC dialog based application.

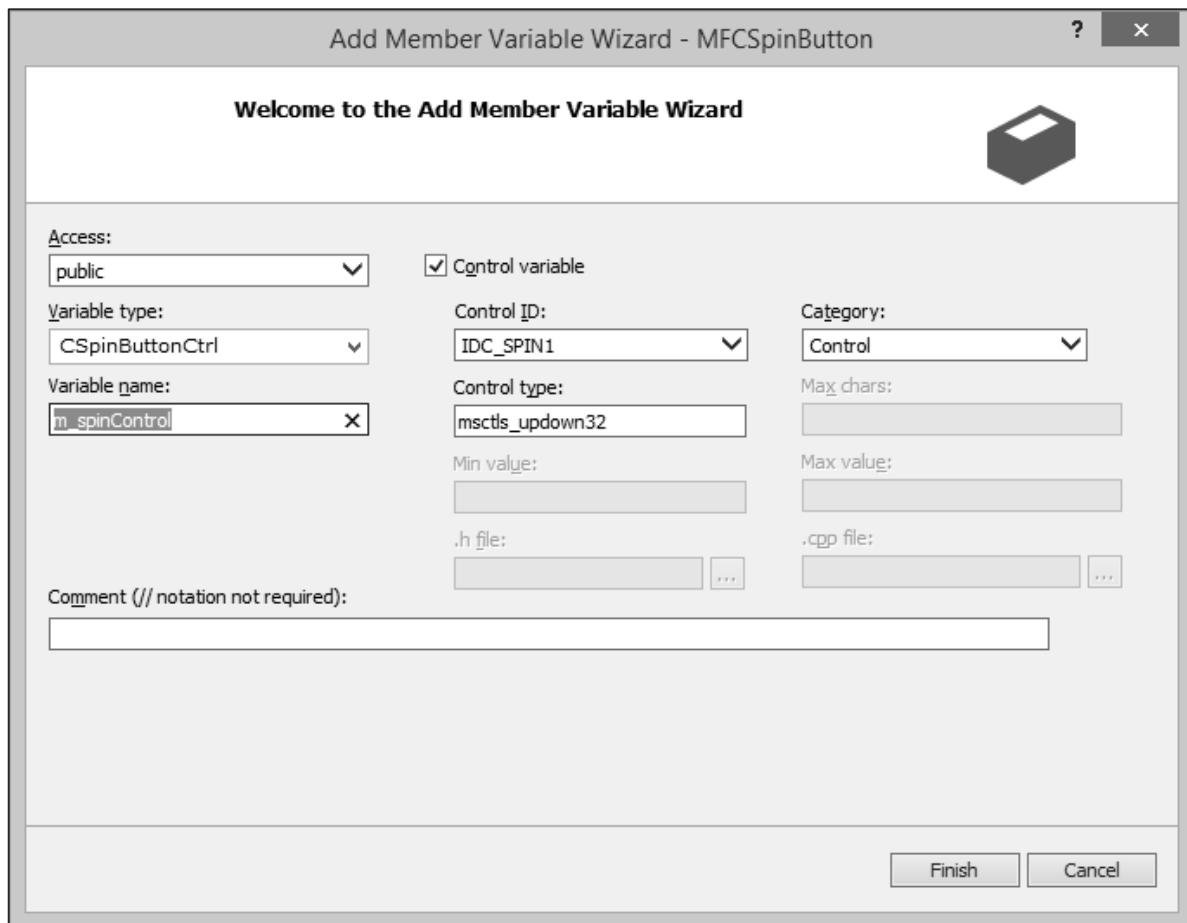
**Step 1:** Add one Spin Control and one Edit control from the Toolbox.



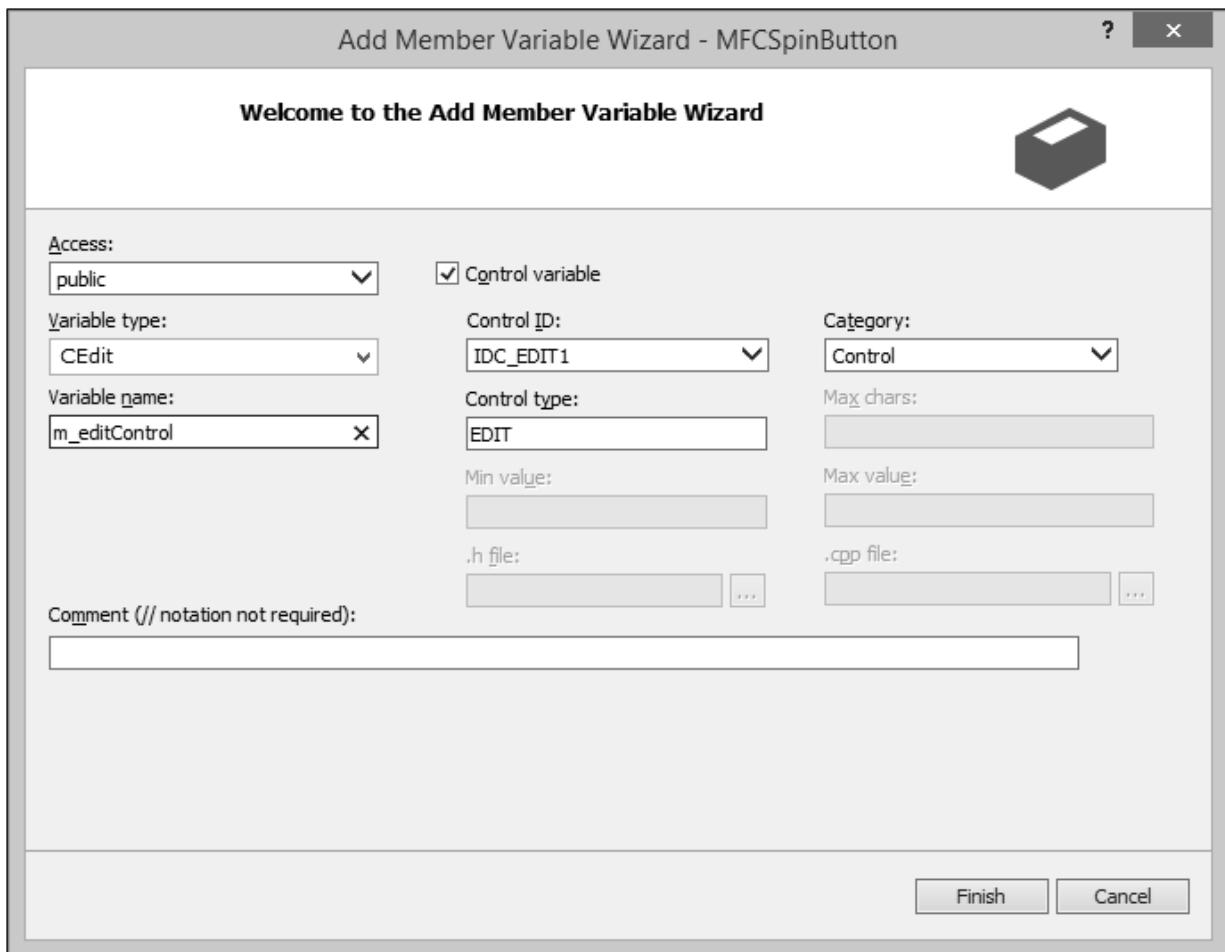
**Step 2:** Go to the Properties of Spin Control and set the values of **Auto Buddy** and **Set Buddy Integer** to True.

## Managing the Updown Control

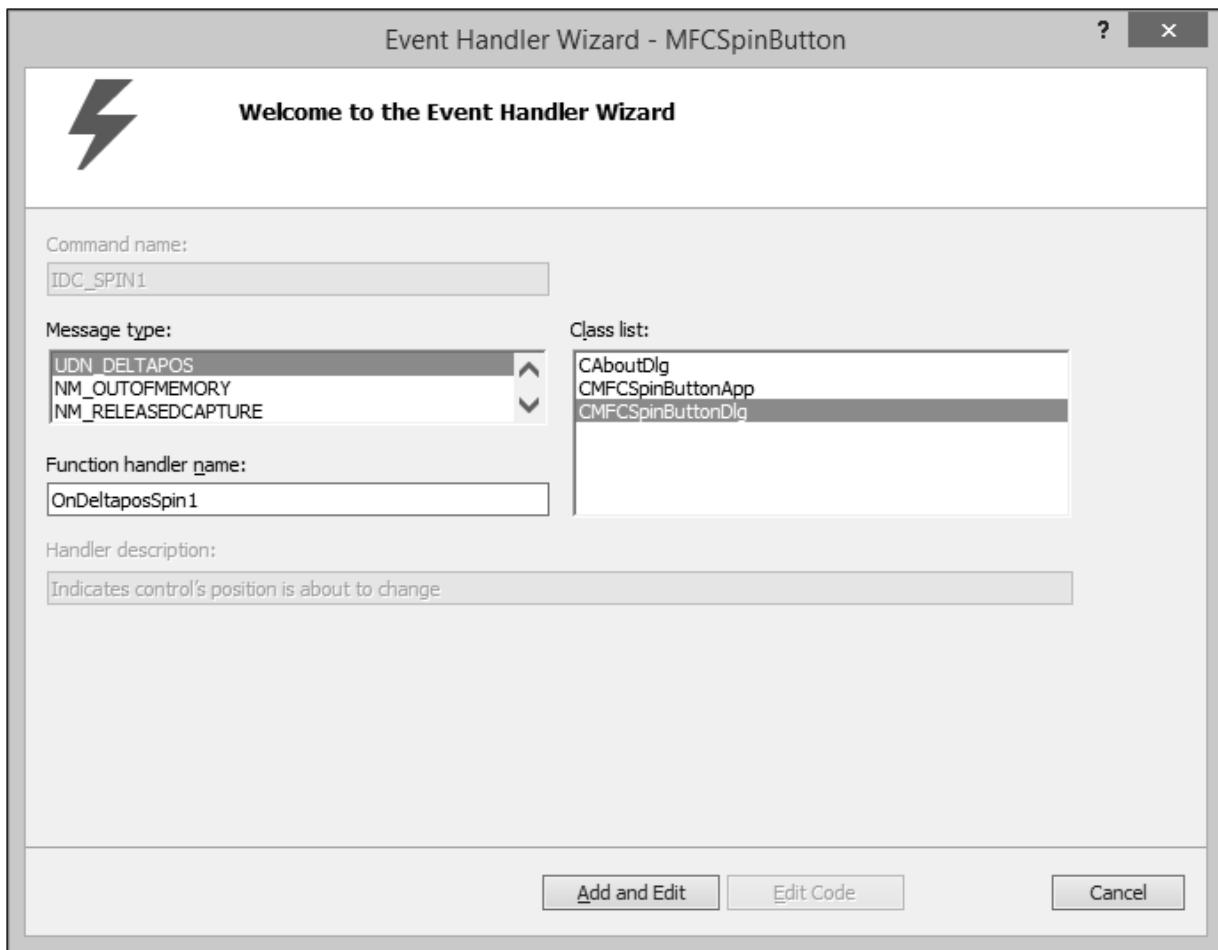
**Step 1:** Add a control variable **m\_spinControl** for spin control with the settings as shown in the following snapshot.



**Step 2:** Add the control variable m\_editControl for Edit control.



**Step 3:** Add the event handler for UDN\_DELTAPOS event for the spin button.



**Step 4:** Update the OnInitDialog() as shown in the following code.

```
BOOL CMFCSpinButtonDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

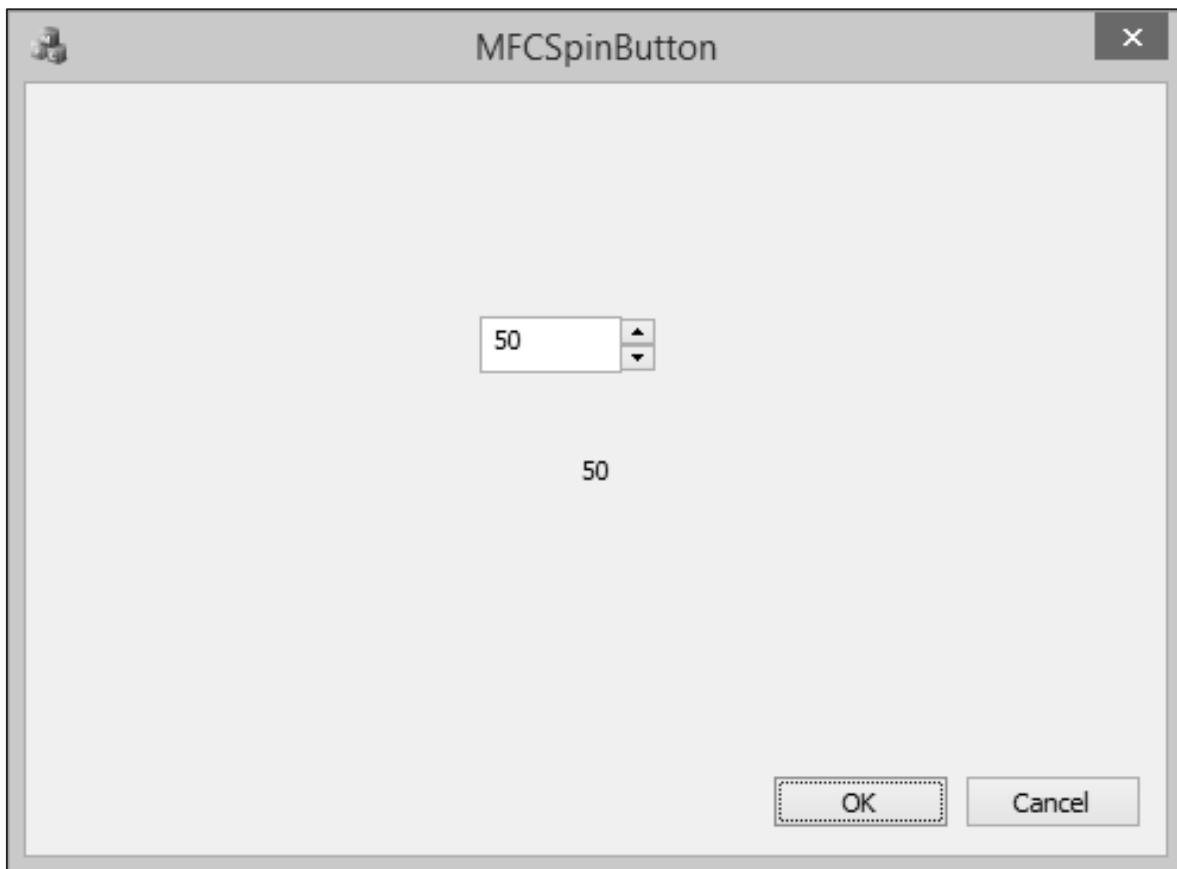
    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon
    // TODO: Add extra initialization here
    m_spinControl.SetRange(0, 100);
    m_spinControl.SetPos(50);
    m_editControl.SetWindowText(L"50");
    return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 5:** Here is the implementation of spin control event.

```
void CMFCSpinButtonDlg::OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
    // TODO: Add your control notification handler code here
    // Declare a pointer to a CSpinButtonCtrl;
    CSpinButtonCtrl *Spinner;
    // Get a pointer to our spin button
    Spinner = reinterpret_cast<CSpinButtonCtrl *>(GetDlgItem(IDC_SPIN1));
    // Found out if it is our spin button that sent the message
    // This conditional statement appears useless but so what?
    if (pNMHDR->hwndFrom == Spinner->m_hWnd)
    {
        // Get the current value of the spin button
        int CurPos = pNMUpDown->iPos;
        // Convert the value to a string

        CString str;
        str.Format(L"%d", CurPos);
        // Display the value into the accompanying edit box
        m_editControl.SetWindowText(str);
    }
    *pResult = 0;
}
```

**Step 6:** When the above code is compiled and executed, you will see the following output.



## Progress Control

A **progress bar control** is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled, from left to right, with the system highlight color as an operation progresses. It is represented by **CProgressCtrl** class.

Here is the list of methods in CProgressCtrl class:

Name	Description
<b>Create</b>	Creates a progress bar control and attaches it to a CProgressCtrl object.
<b>CreateEx</b>	Creates a progress control with the specified Windows extended styles and attaches it to a CProgressCtrl object.
<b>GetBarColor</b>	Gets the color of the progress indicator bar for the current progress bar control.
<b>GetBkColor</b>	Gets the background color of the current progress bar.
<b>GetPos</b>	Gets the current position of the progress bar.
<b>GetRange</b>	Gets the lower and upper limits of the range of the progress bar control.
<b>GetState</b>	Gets the state of the current progress bar control.
<b>GetStep</b>	Retrieves the step increment for the progress bar of the current progress bar control.

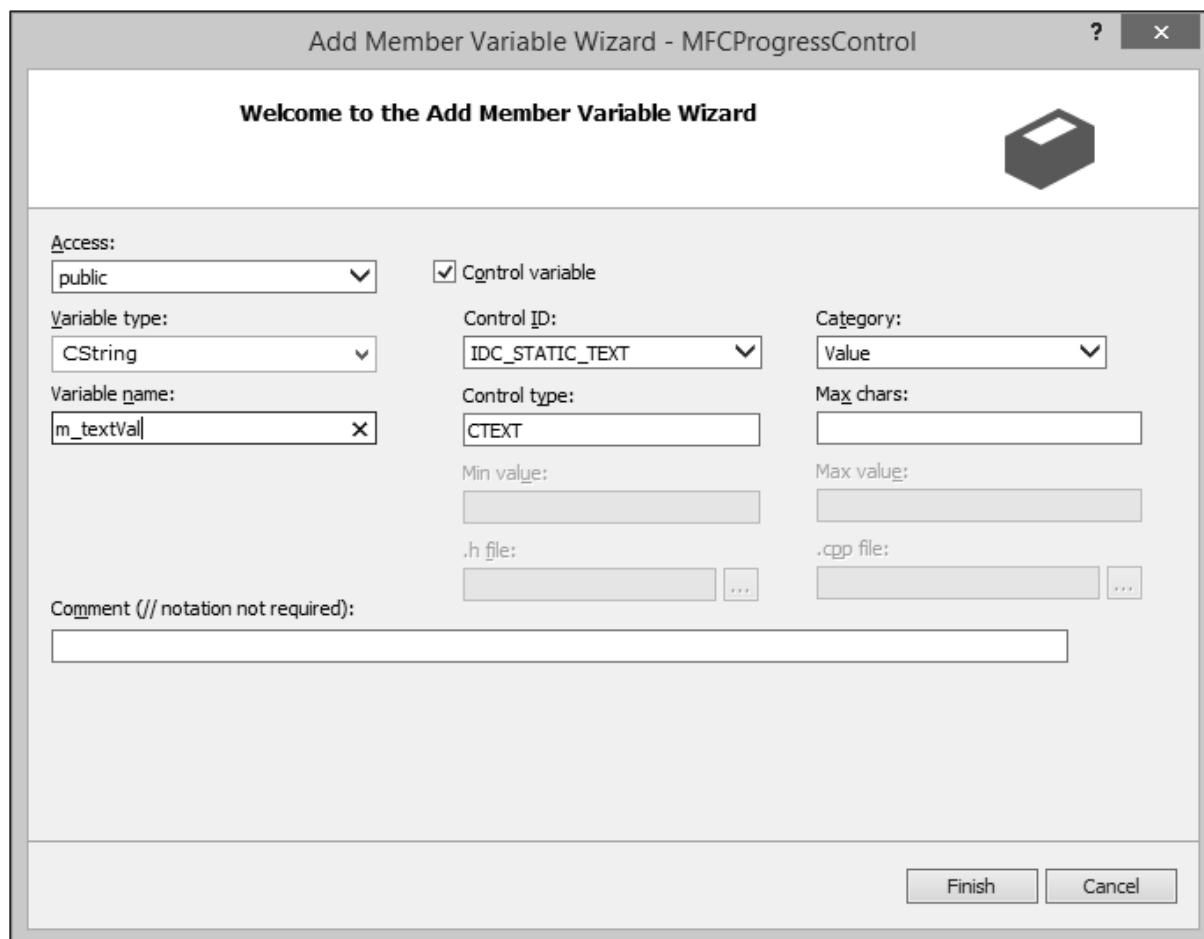
<b>OffsetPos</b>	Advances the current position of a progress bar control by a specified increment and redraws the bar to reflect the new position.
<b>SetBarColor</b>	Sets the color of the progress indicator bar in the current progress bar control.
<b>SetBkColor</b>	Sets the background color for the progress bar.
<b>SetMarquee</b>	Turns marquee mode on or off for the current progress bar control.
<b>SetPos</b>	Sets the current position for a progress bar control and redraws the bar to reflect the new position.
<b>SetRange</b>	Sets the minimum and maximum ranges for a progress bar control and redraws the bar to reflect the new ranges.
<b>SetState</b>	Sets the state of the current progress bar control.
<b>SetStep</b>	Specifies the step increment for a progress bar control.
<b>StepIt</b>	Advances the current position for a progress bar control by the step increment (see SetStep) and redraws the bar to reflect the new position.

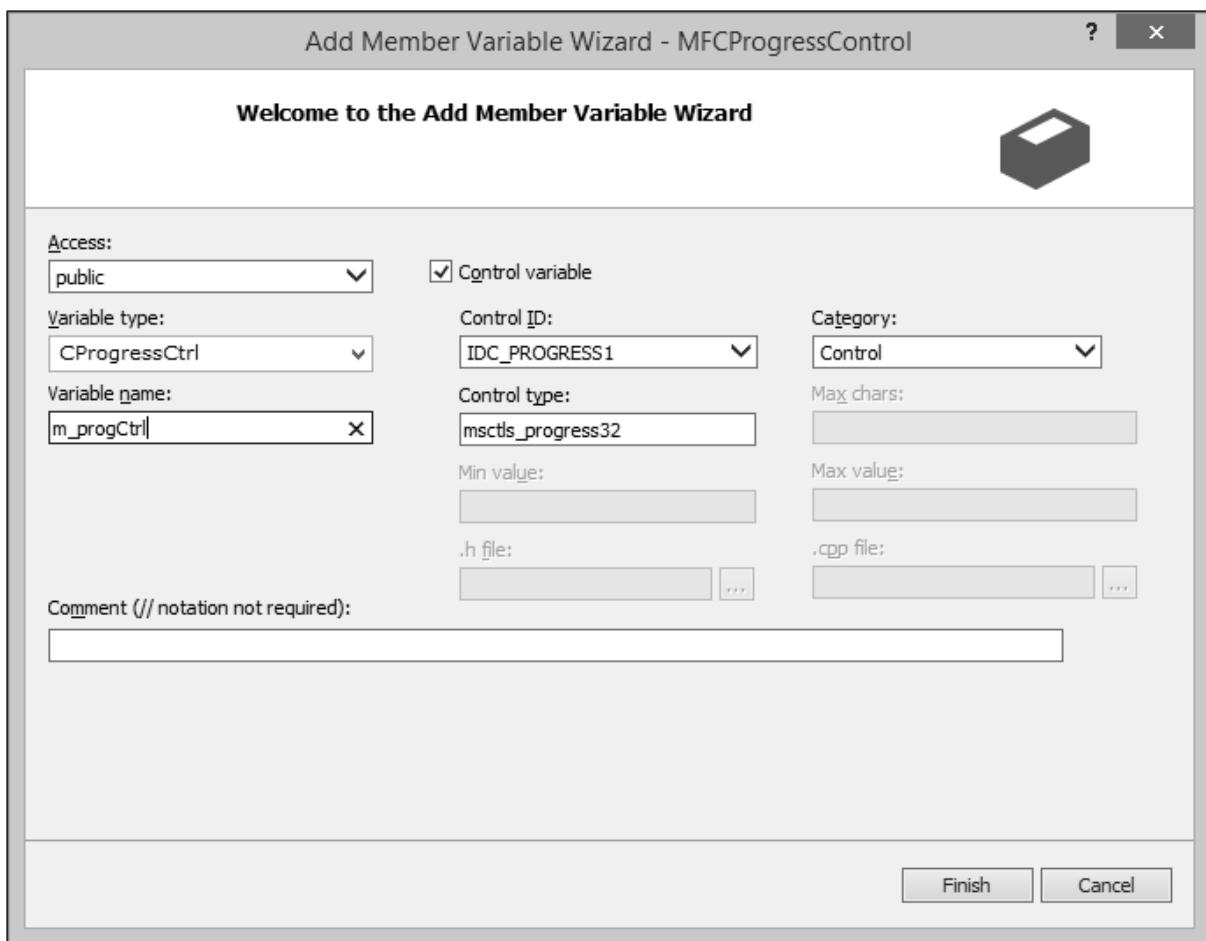
Let us create a new MFC application.

**Step 1:** Change the ID of the Text control to and remove the caption.

**Step 2:** Drag Progress Control from the Toolbox.

**Step 3:** Add value variable for Static Text control.



**Step 4:** Add control variable for the Progress control.**Step 5:** Here is the implementation in OnInitDialog()

```

BOOL CMFCProgressControlDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;

```

```
bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
    pSysMenu->AppendMenu(MF_SEPARATOR);
    pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
}

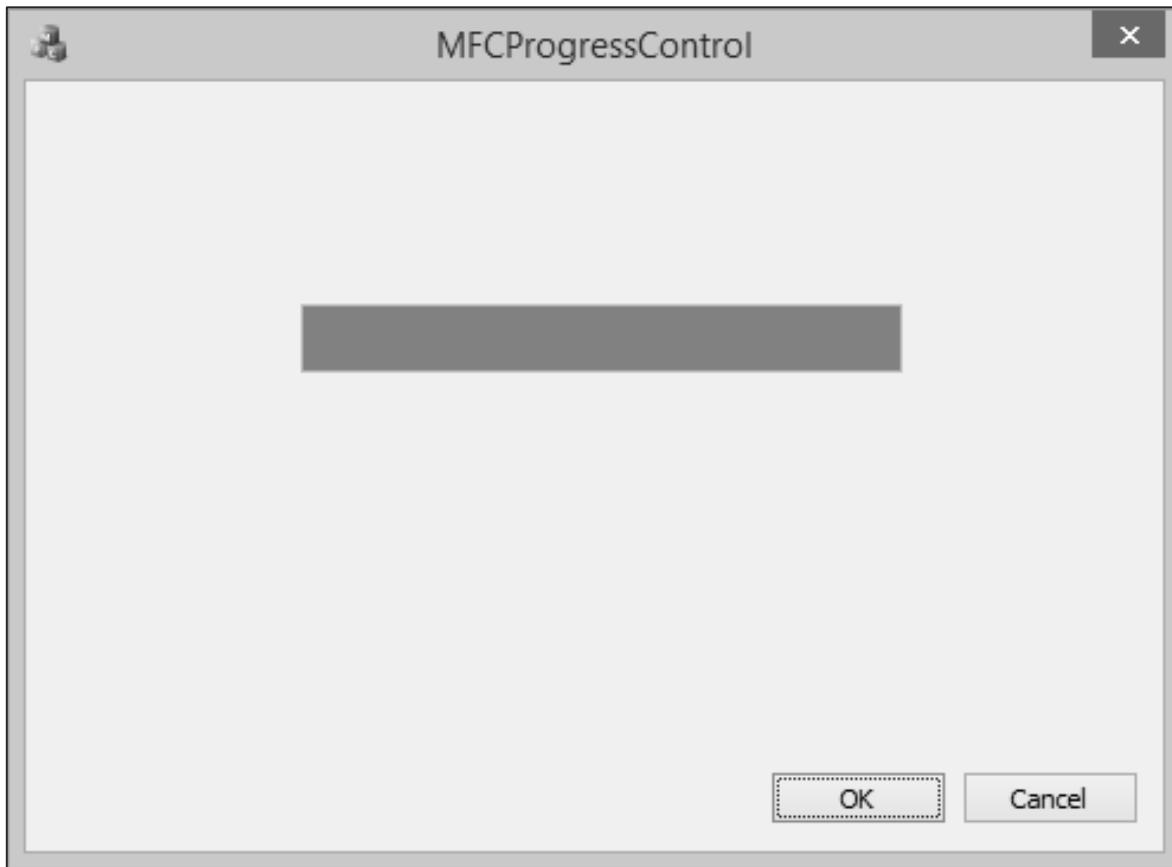
// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

// TODO: Add extra initialization here
m_progCtrl.SetRange(0,100);

for (int i = 0; i <= 100; i++)
{
    m_progCtrl.SetPos(i);
}

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 6:** When the above code is compiled and executed, you will see the following output.



## Progress Bars

---

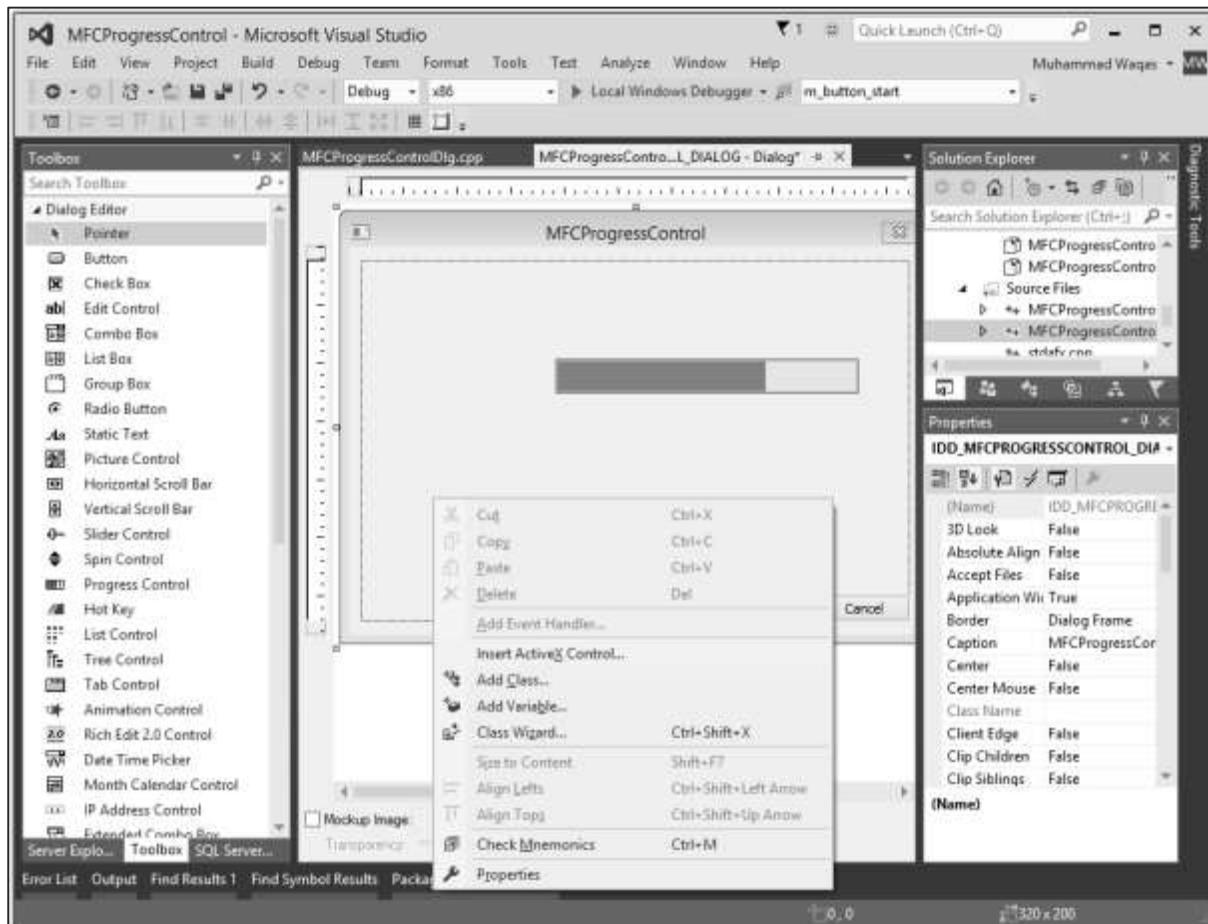
Besides the Progress control, Visual C++ provides two other progress-oriented controls:

- The Microsoft Progress Control Version 5.0
- The Microsoft Progress Control Version 6.0

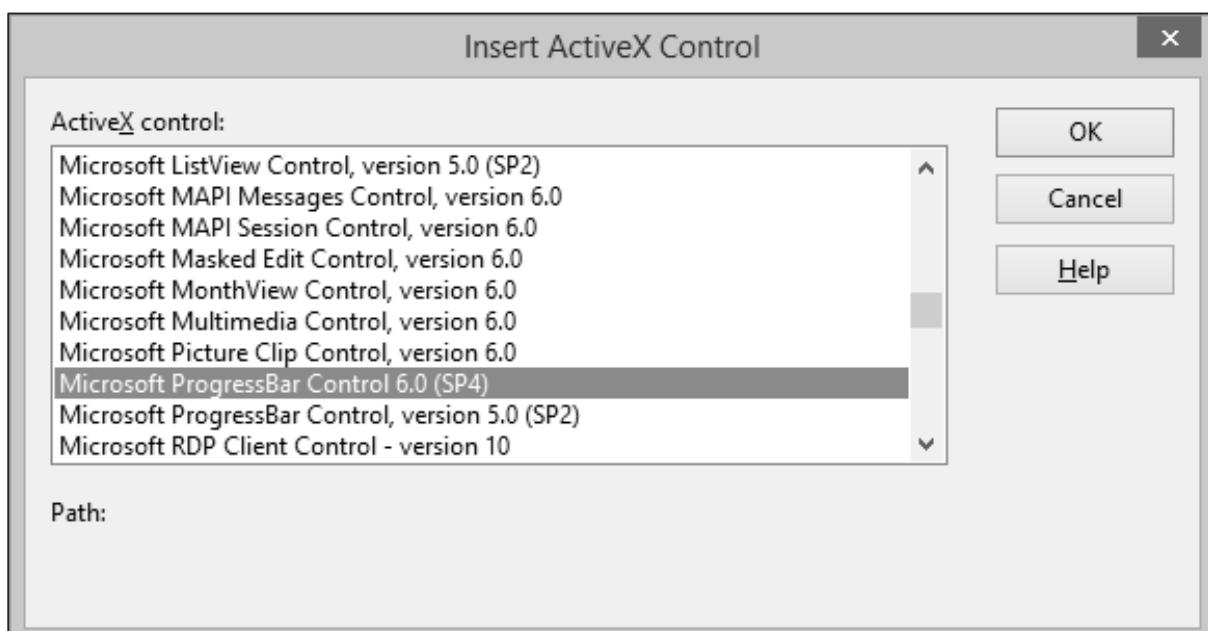
The main difference is in their ability to assume one or two orientations.

Let us look into a simple example.

**Step 1:** Right-click on the dialog in the designer window.



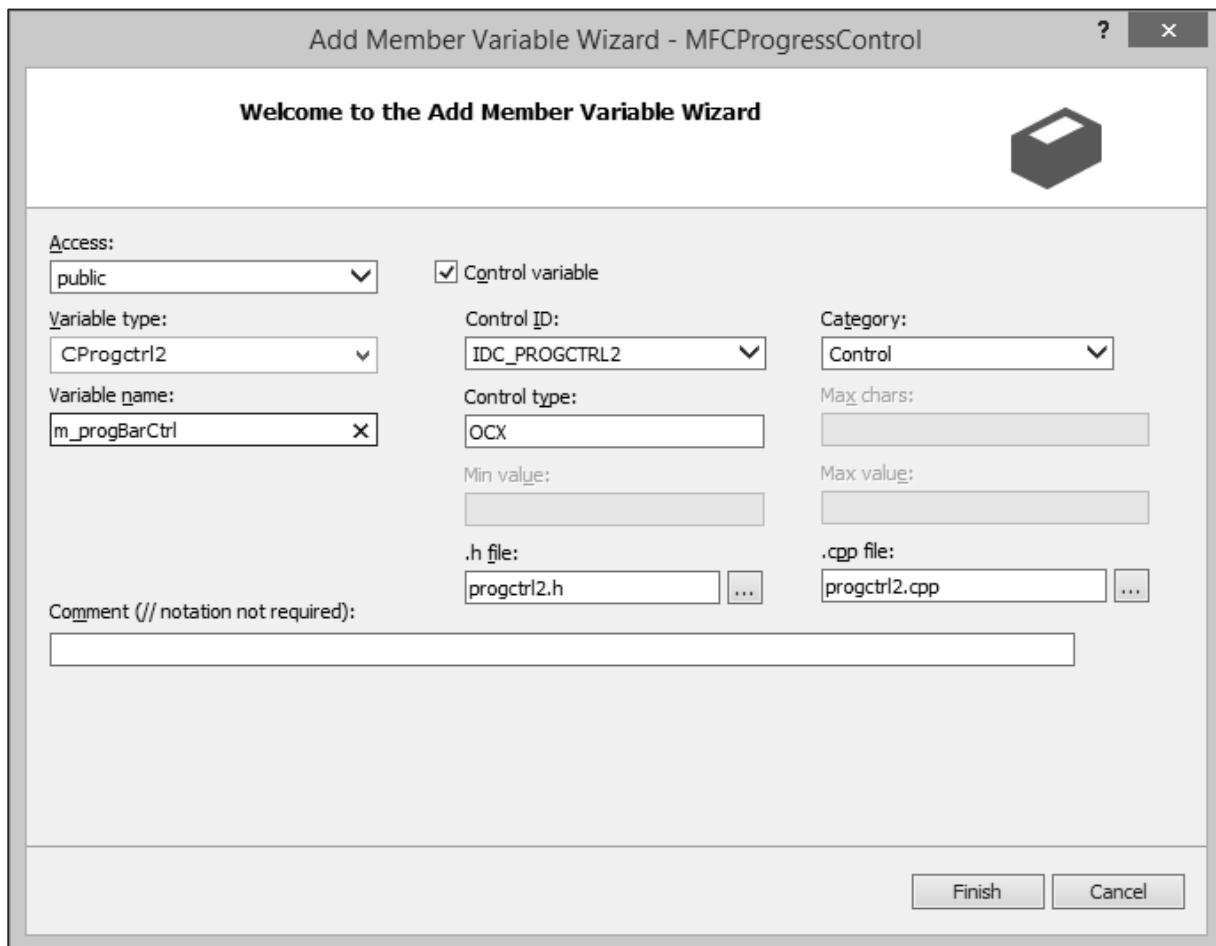
**Step 2:** Select Insert ActiveX Control.



**Step 3:** Select the Microsoft ProgressBar Control 6.0 and click OK.

**Step 4:** Select the progress bar and set its Orientation in the Properties Window to **1 - ccOrientationVertical**.

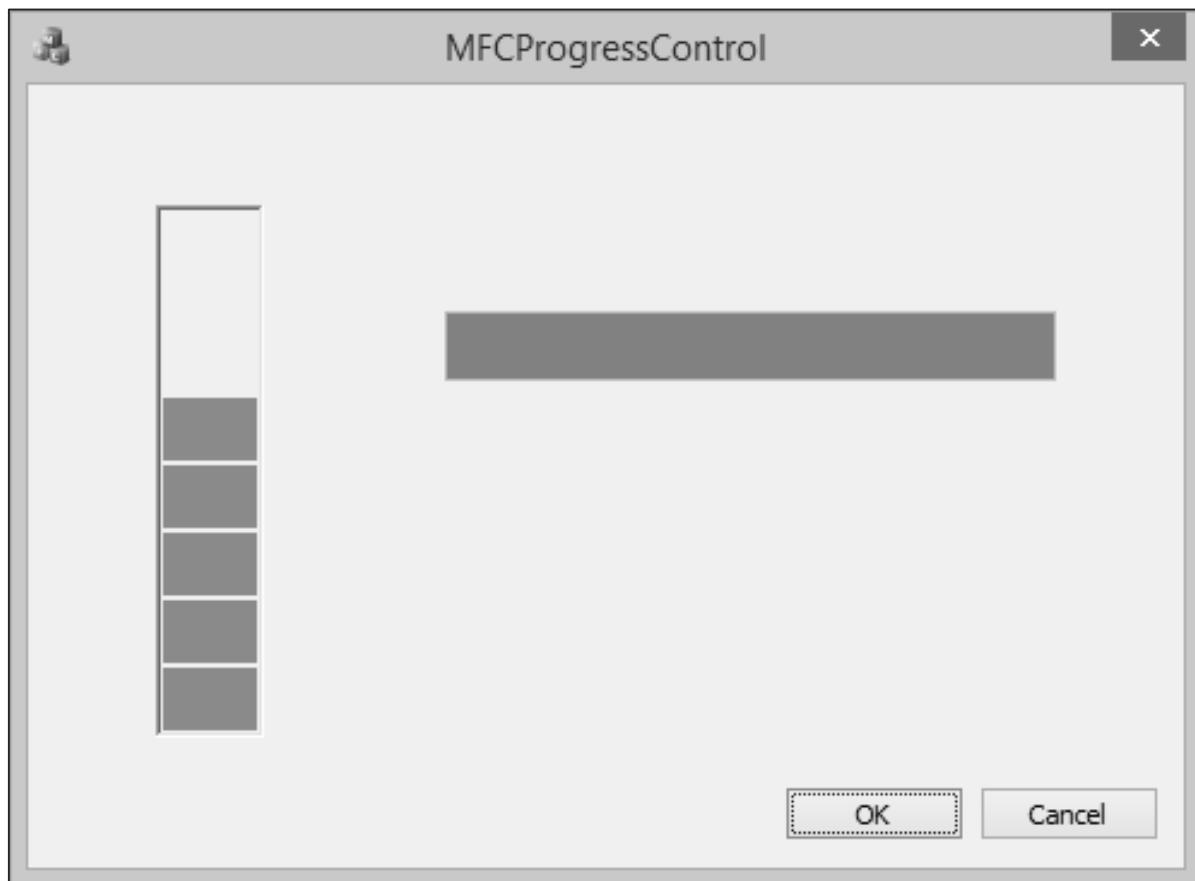
**Step 5:** Add control variable for Progress bar.



**Step 6:** Add the following code in the OnInitDialog()

```
m_progBarCtrl.SetScrollRange(0,100,TRUE);  
m_progBarCtrl.put_Value(53);
```

**Step 7:** Run this application again and you will see the progress bar in Vertical direction as well.



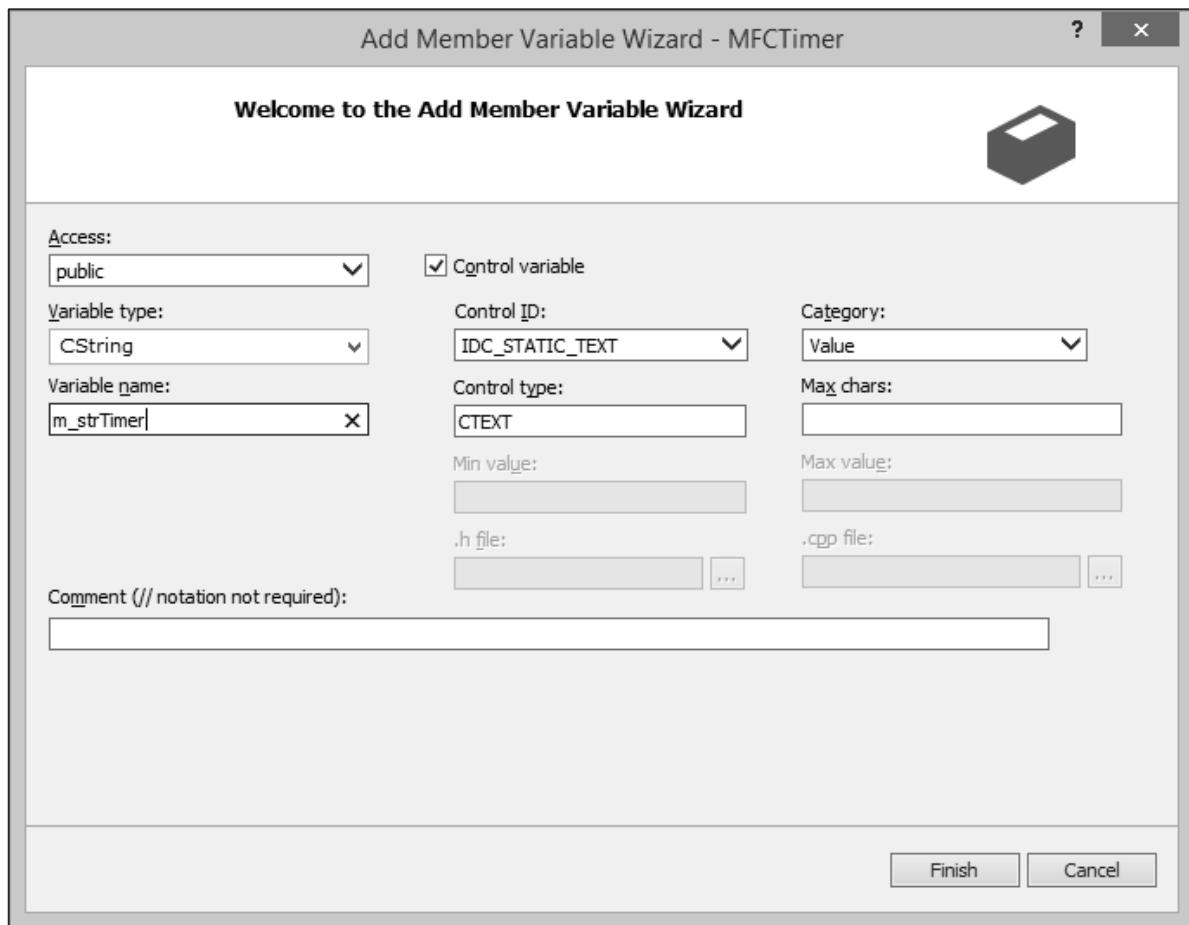
## Timer

A **timer** is a non-spatial object that uses recurring lapses of time from a computer or from your application. To work, every lapse of period, the control sends a message to the operating system. Unlike most other controls, the MFC timer has neither a button to represent it nor a class. To create a timer, you simply call the `CWnd::SetTimer()` method. This function call creates a timer for your application. Like the other controls, a timer uses an identifier.

Let us create a new MFC dialog based application.

**Step 1:** Remove the Caption and set its ID to IDC\_STATIC\_TXT.

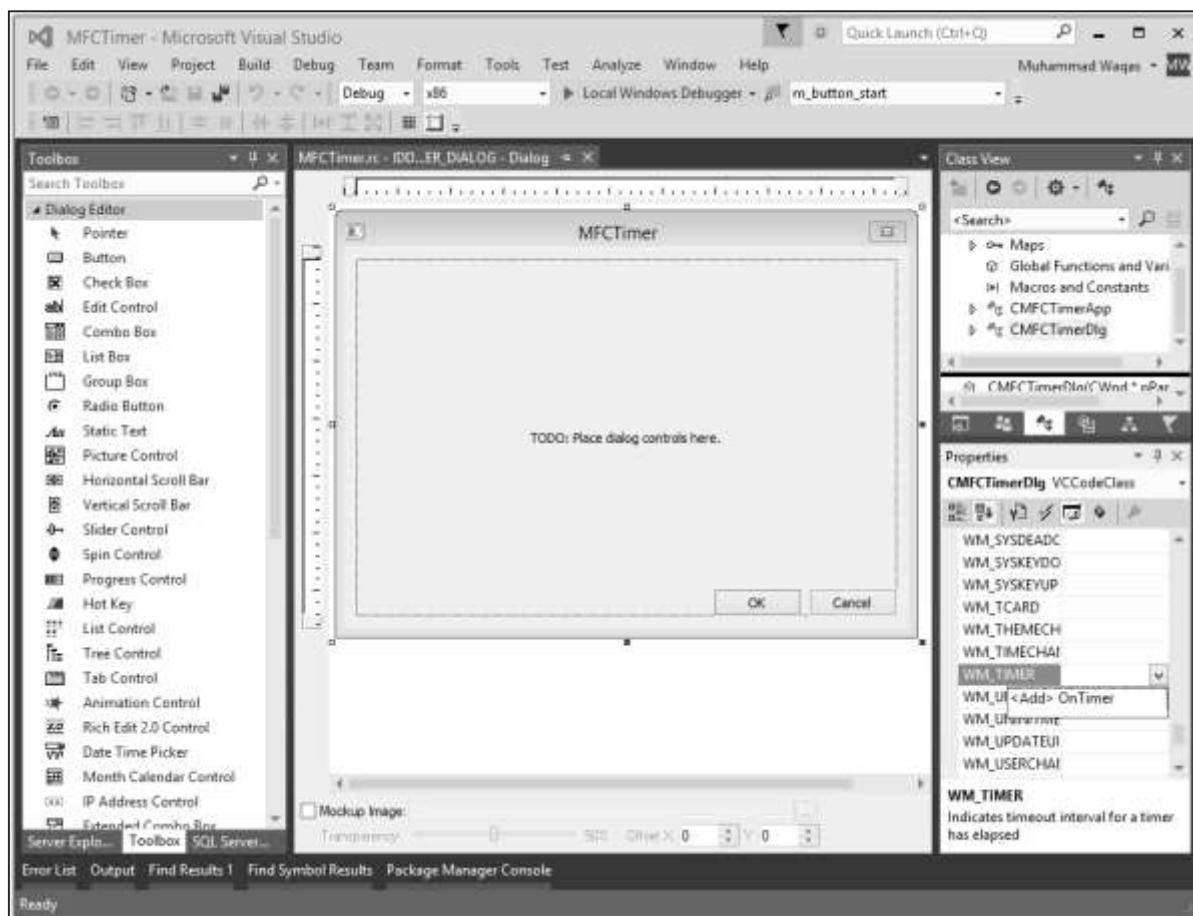
**Step 2:** Add the value variable for text control.



**Step 3:** Go to the class view in solution.

**Step 4:** Click the CMFCTimeDlg class.

**Step 5:** In the Properties window, click the Messages button.



**Step 6:** Click the WM\_TIMER field and click the arrow of its combo box. Select <Add> OnTimer and implement the event.

```
void CMFCTimerDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    CTime CurrentTime = CTime::GetCurrentTime();

    int iHours = CurrentTime.GetHour();
    int iMinutes = CurrentTime.GetMinute();
    int iSeconds = CurrentTime.GetSecond();
    CString strHours, strMinutes, strSeconds;

    if (iHours < 10)
        strHours.Format(_T("0%d"), iHours);
    else
        strHours.Format(_T("%d"), iHours);

    if (iMinutes < 10)
```

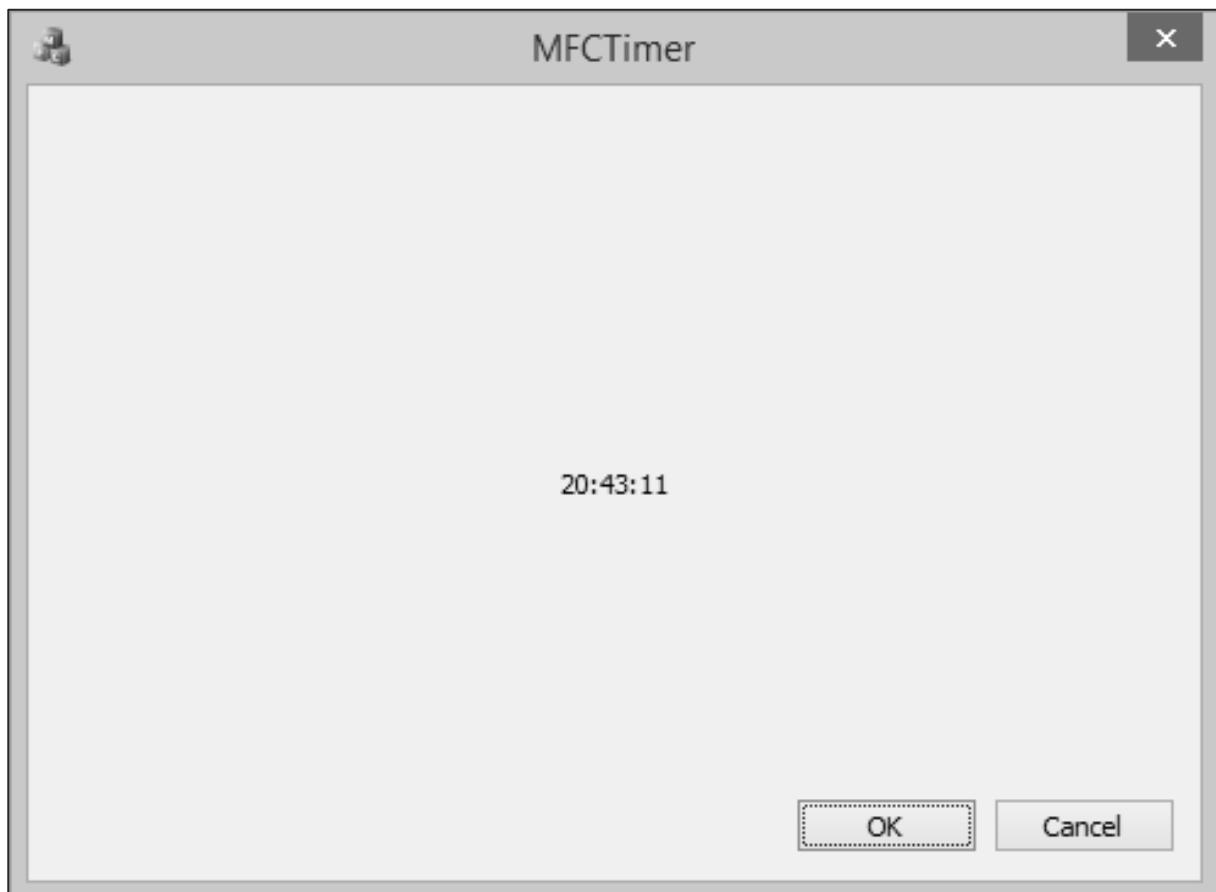
```
    strMinutes.Format(_T("0%d"), iMinutes);
else
    strMinutes.Format(_T("%d"), iMinutes);

if (iSeconds < 10)
    strSeconds.Format(_T("0%d"), iSeconds);
else
    strSeconds.Format(_T("%d"), iSeconds);

m_strTimer.Format(_T("%s:%s:%s"), strHours, strMinutes, strSeconds);

UpdateData(FALSE);
CDialogEx::OnTimer(nIDEvent);
}
```

**Step 7:** When the above code is compiled and executed, you will see the following output.



## Date & Time Picker

The date and time picker control (**CDateTimeCtrl**) implements an intuitive and recognizable method of entering or selecting a specific date. The main interface of the control is similar in functionality to a combo box. However, if the user expands the control, a month calendar control appears (by default), allowing the user to specify a particular date. When a date is chosen, the month calendar control automatically disappears.

Here is the list of methods in CDateTimeCtrl class:

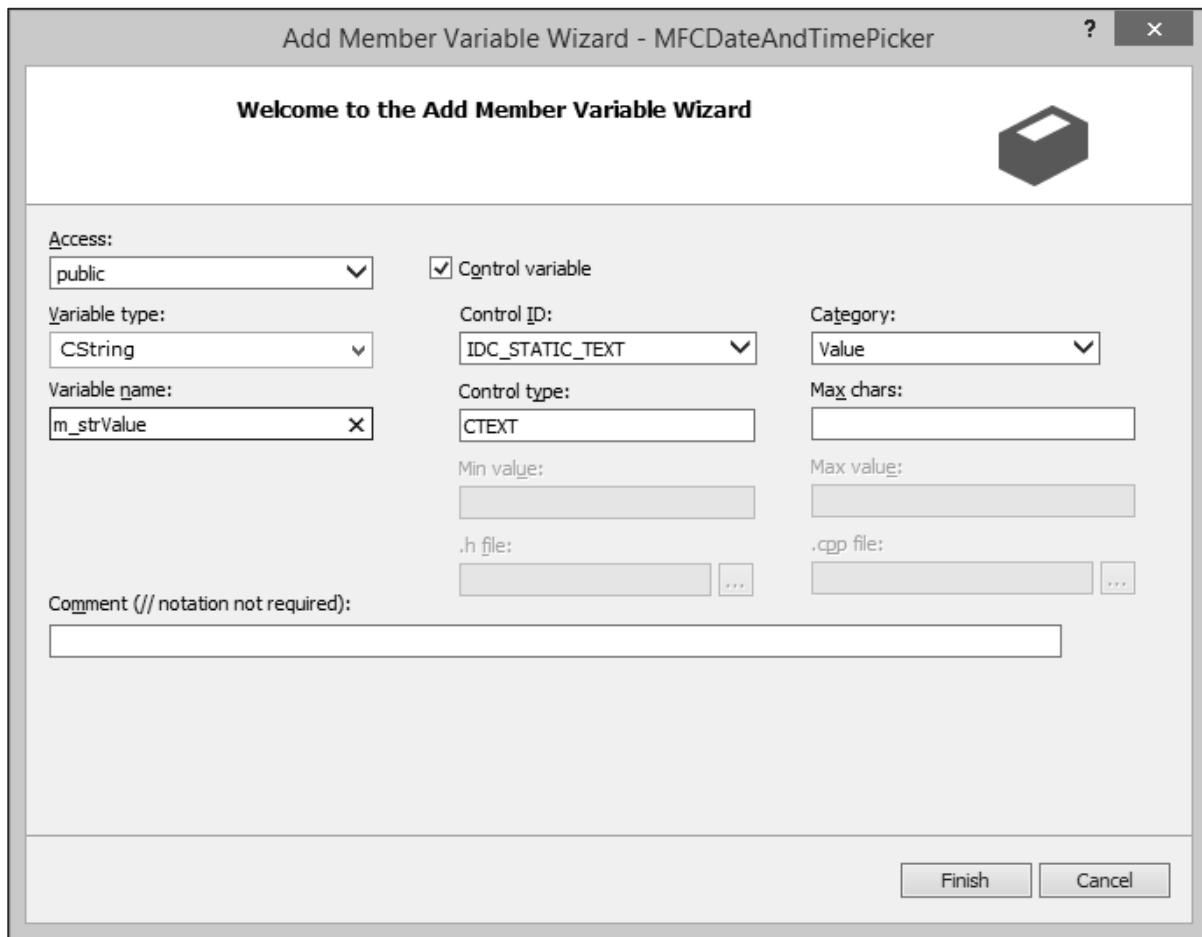
Name	Description
<b>CloseMonthCal</b>	Closes the current date and time picker control.
<b>Create</b>	Creates the date and time picker control and attaches it to the CDateTimeCtrl object.
<b>GetDateTimePickerInfo</b>	Retrieves information about the current date and time picker control.
<b>GetIdealSize</b>	Returns the ideal size of the date and time picker control that is required to display the current date or time.
<b>GetMonthCalColor</b>	Retrieves the color for a given portion of the month calendar within the date and time picker control.
<b>GetMonthCalCtrl</b>	Retrieves the <b>CMonthCalCtrl</b> object associated with the date and time picker control.
<b>GetMonthCalFont</b>	Retrieves the font currently used by the date and time picker control's child month calendar control.
<b>GetMonthCalStyle</b>	Gets the style of the current date and time picker control.
<b>GetRange</b>	Retrieves the current minimum and maximum allowed system times for a date and time picker control.
<b>GetTime</b>	Retrieves the currently selected time from a date and time picker control and puts it in a specified <b>SYSTEMTIME</b> structure.
<b>SetFormat</b>	Sets the display of a date and time picker control in accordance with a given format string.
<b>SetMonthCalColor</b>	Sets the color for a given portion of the month calendar within a date and time picker control.
<b>SetMonthCalFont</b>	Sets the font that the date and time picker control's child month calendar control will use.

<b>SetMonthCalStyle</b>	Sets the style of the current date and time picker control.
<b>SetRange</b>	Sets the minimum and maximum allowed system times for a date and time picker control.
<b>SetTime</b>	Sets the time in a date and time picker control.

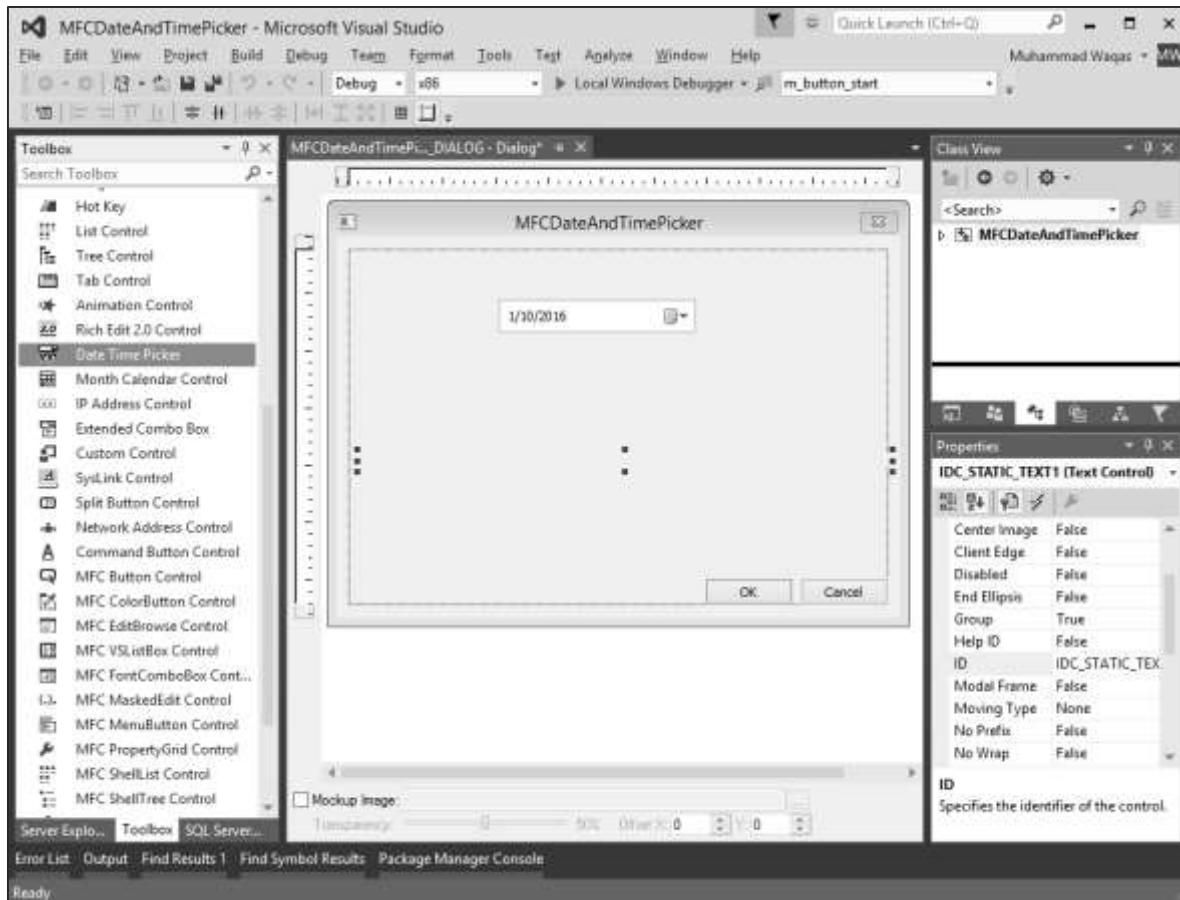
Let us look into a simple example by creating a new MFC application.

**Step 1:** Remove the Caption and set its ID to IDC\_STATIC\_TXT.

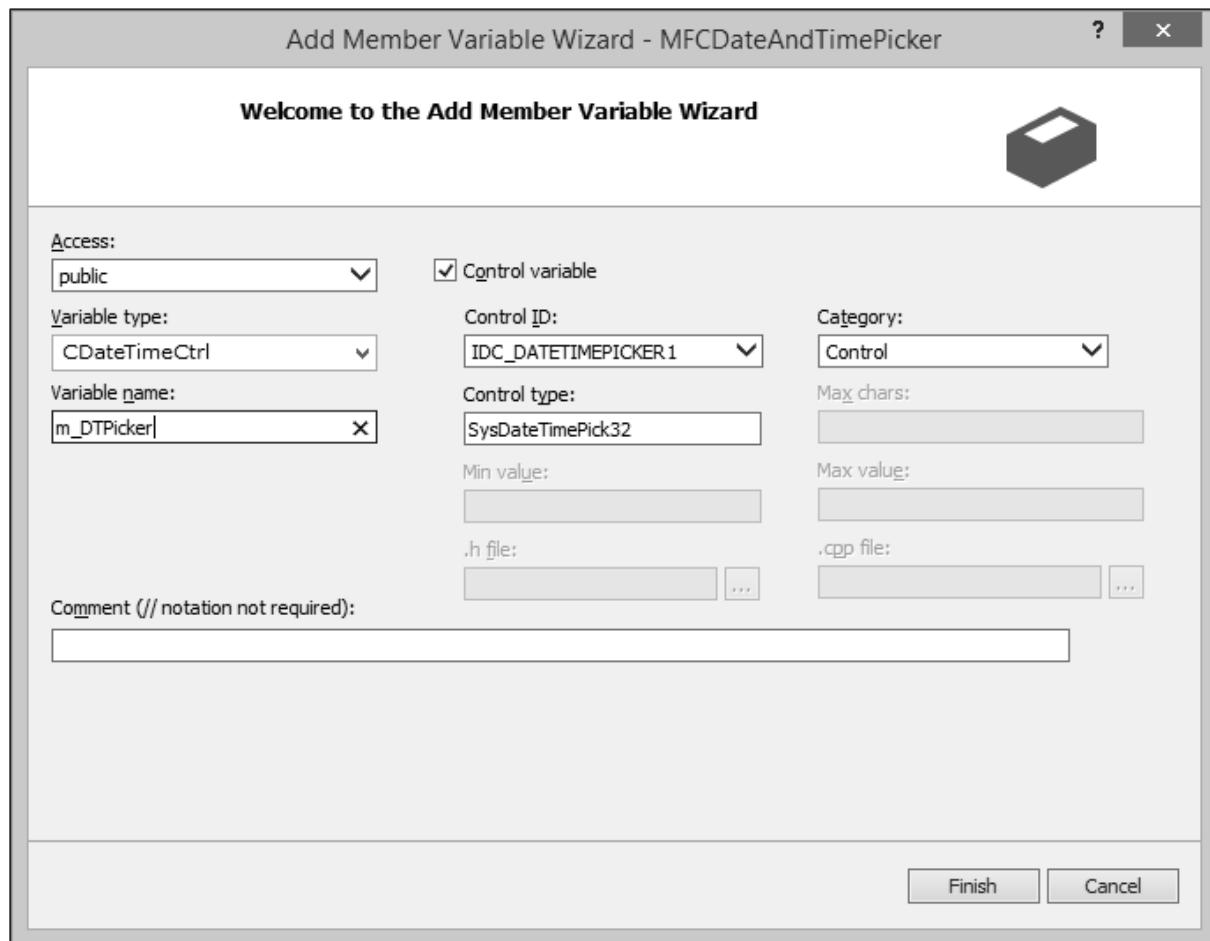
**Step 2:** Add the value variable for text control.

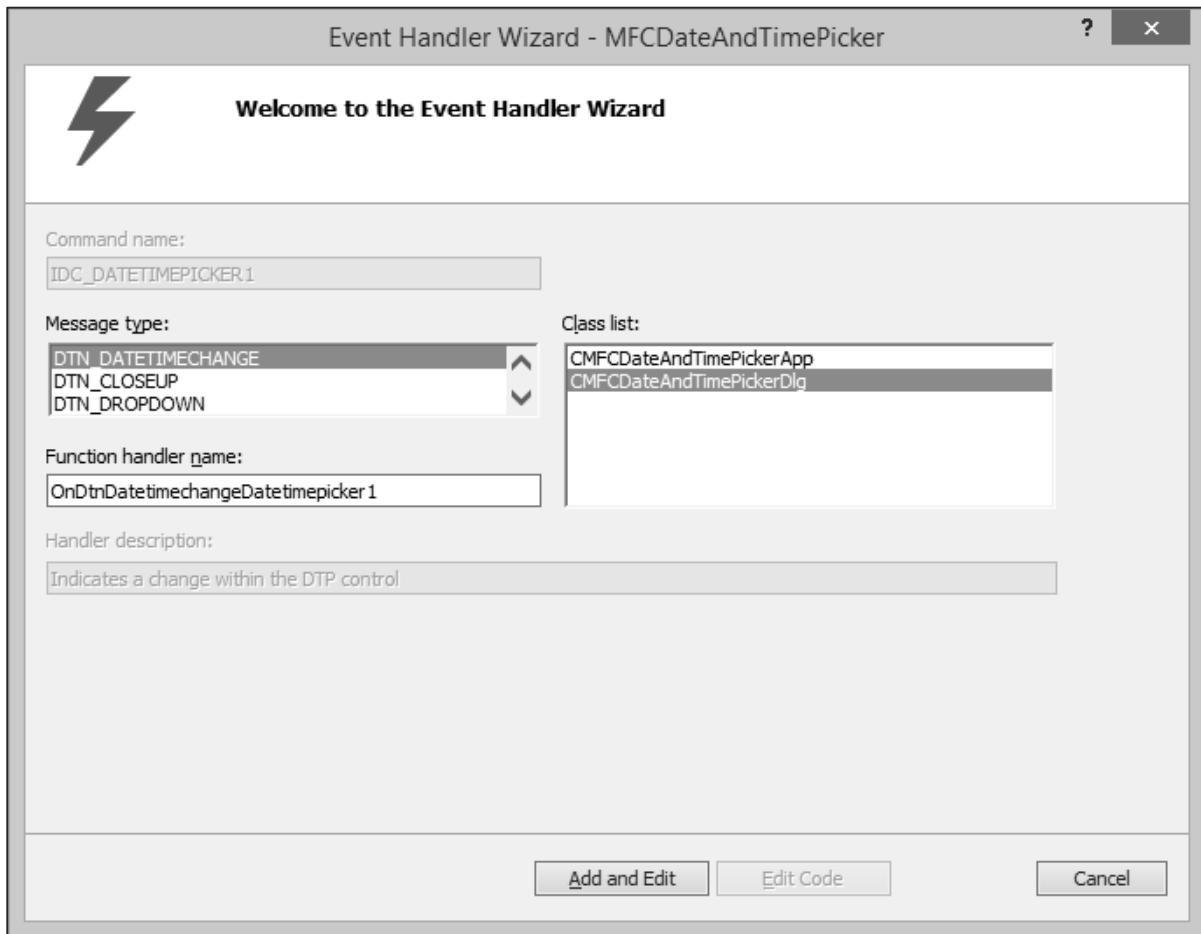


**Step 3:** Drag the Date Time Picker control.



**Step 4:** Add a control variable for Date Time Picker.



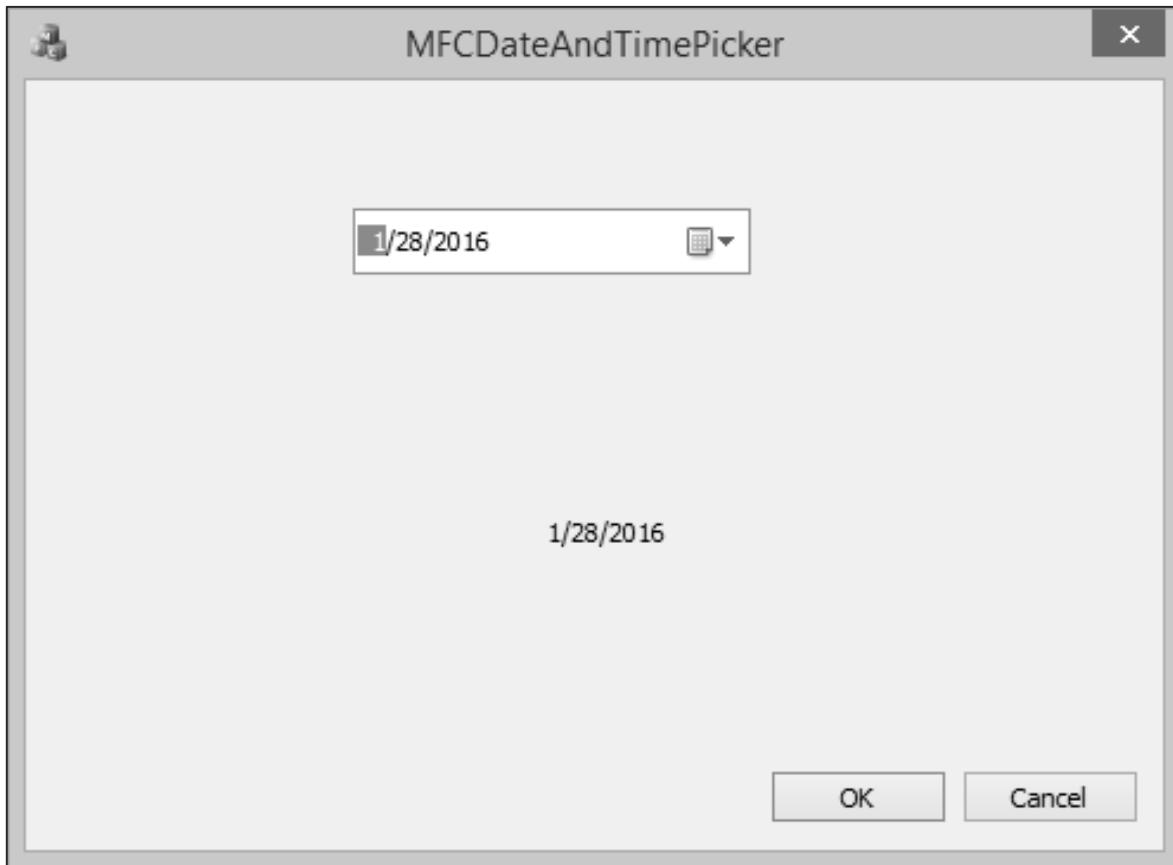
**Step 5:** Add the Event handler for Date Time Picker.**Step 6:** Here is the implementation of event handler.

```
void CMFCDatetimepickerDlg::OnDtnDatetimchangeDatetimepicker1(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMDATEETIMECHANGE pDTChange =
    reinterpret_cast<LPNMDATEETIMECHANGE>(pNMHDR);

    // TODO: Add your control notification handler code here

    GetDlgItemText(IDC_DATEETIMEPICKER1, m_strValue);
    UpdateData(FALSE);
    *pResult = 0;
}
```

**Step 7:** When you run the above application, you see the following output. Select any date, it will display on the Static Text Control.



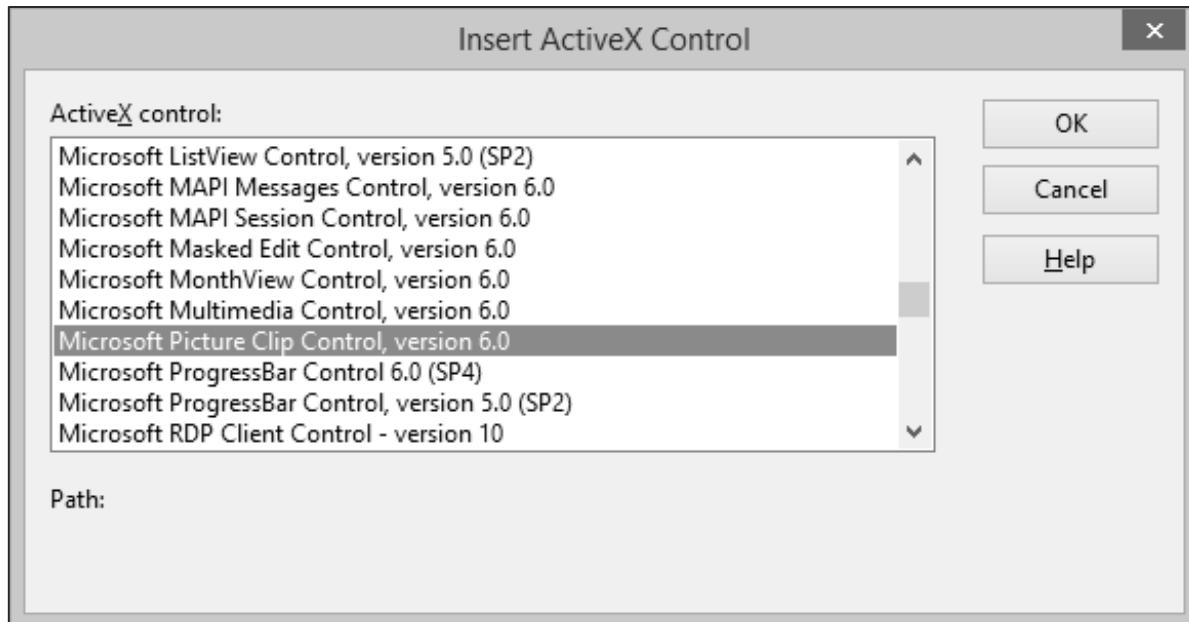
## Picture

---

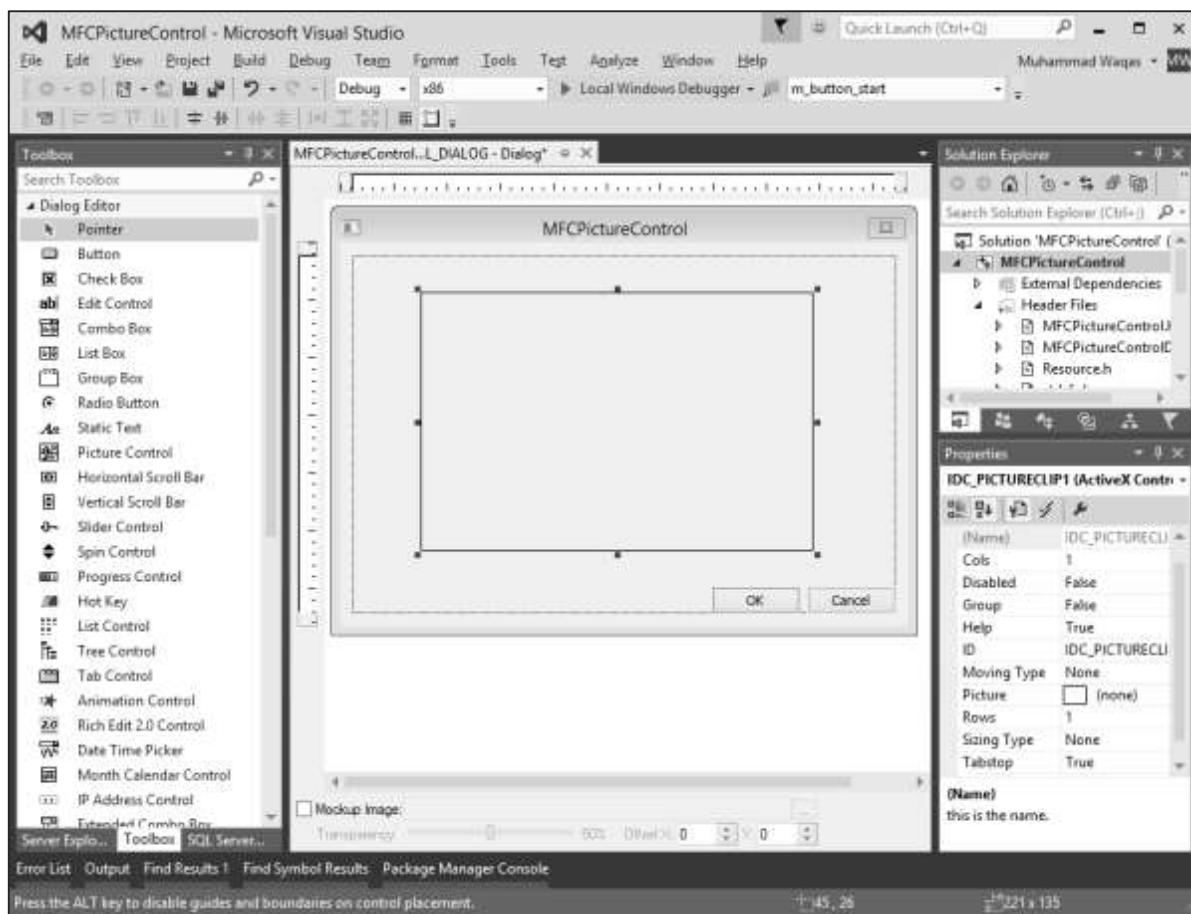
If you need to display a picture for your application, Visual C++ provides a special control for that purpose.

Let us look into a simple example by creating a new MFC dialog based application.

**Step 1:** Right-click on the dialog in the designer window and select Insert ActiveX Control.



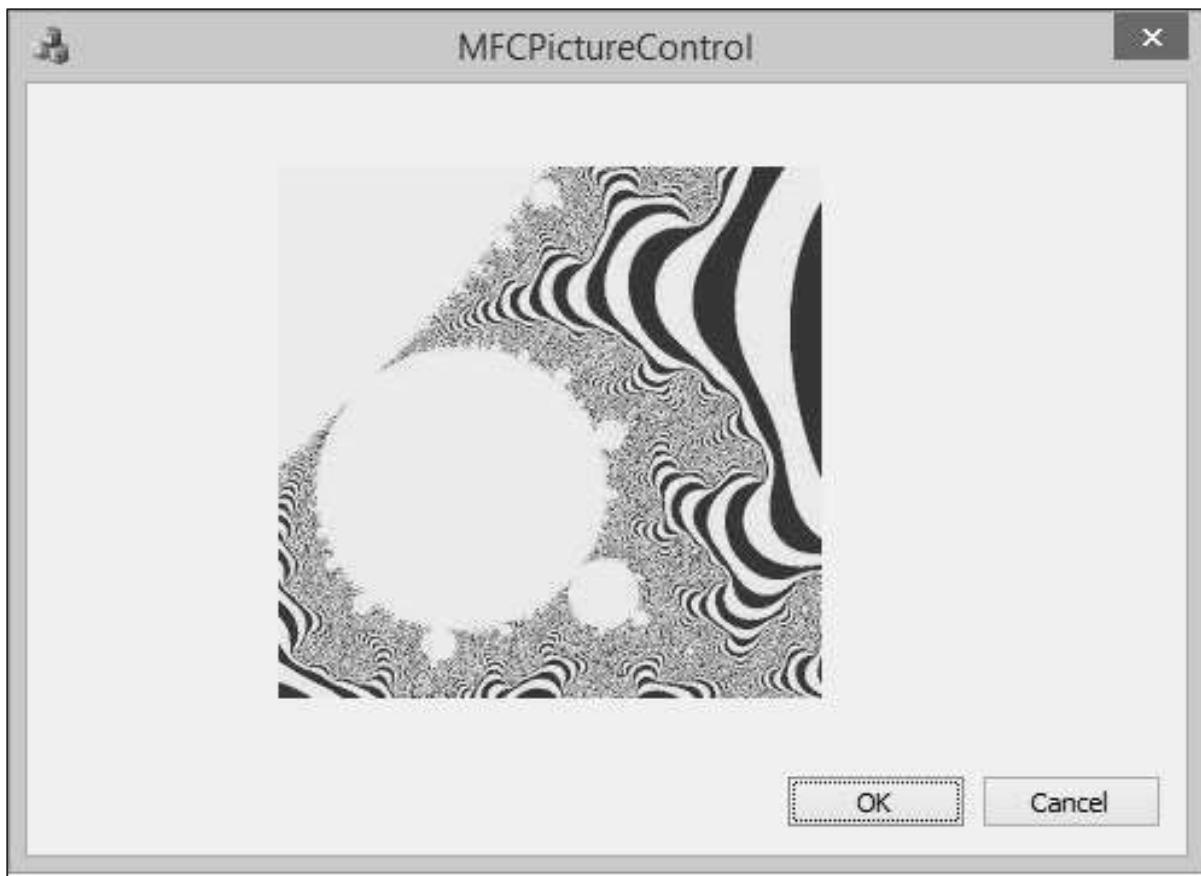
**Step 2:** Select the Microsoft Picture Click Control and click OK.



**Step 3:** Resize the Picture control and in the Properties window, click the Picture field.

**Step 4:** Browse the folder that contains Pictures. Select any picture.

**Step 5:** Run this application and you will see the following output.

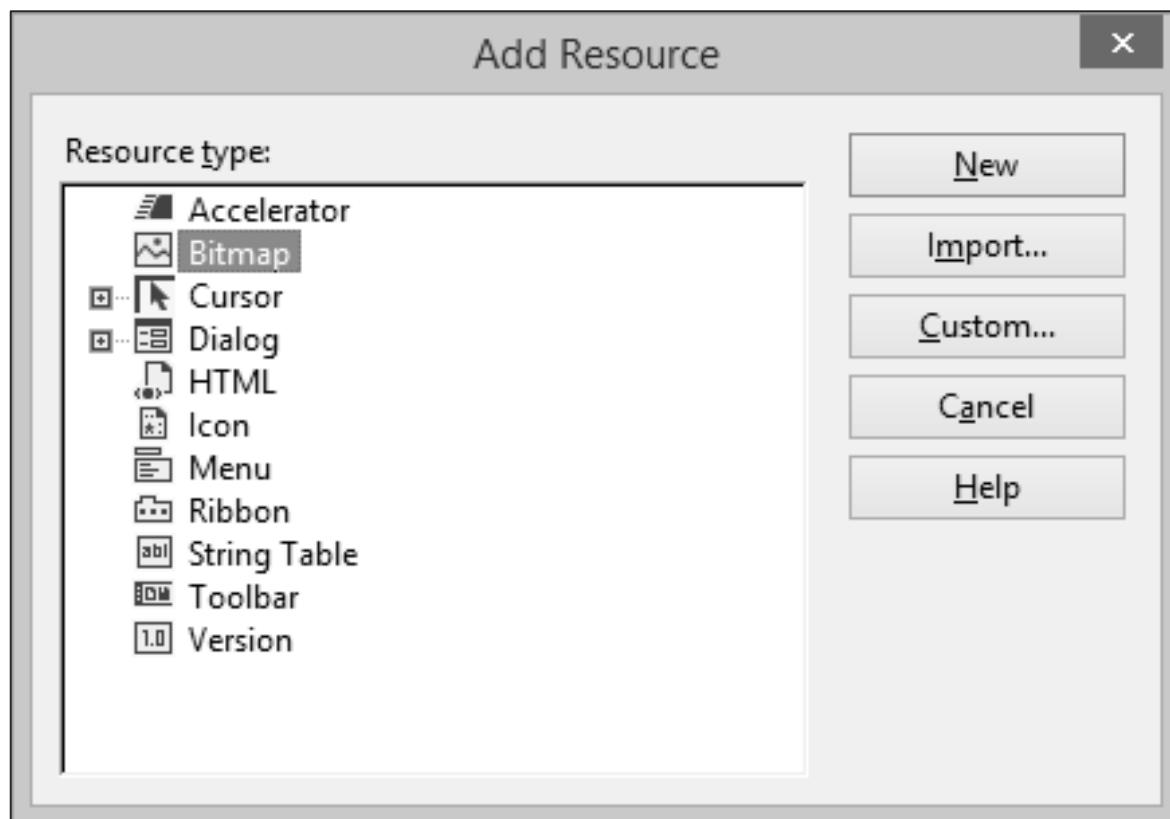


## Image Editor

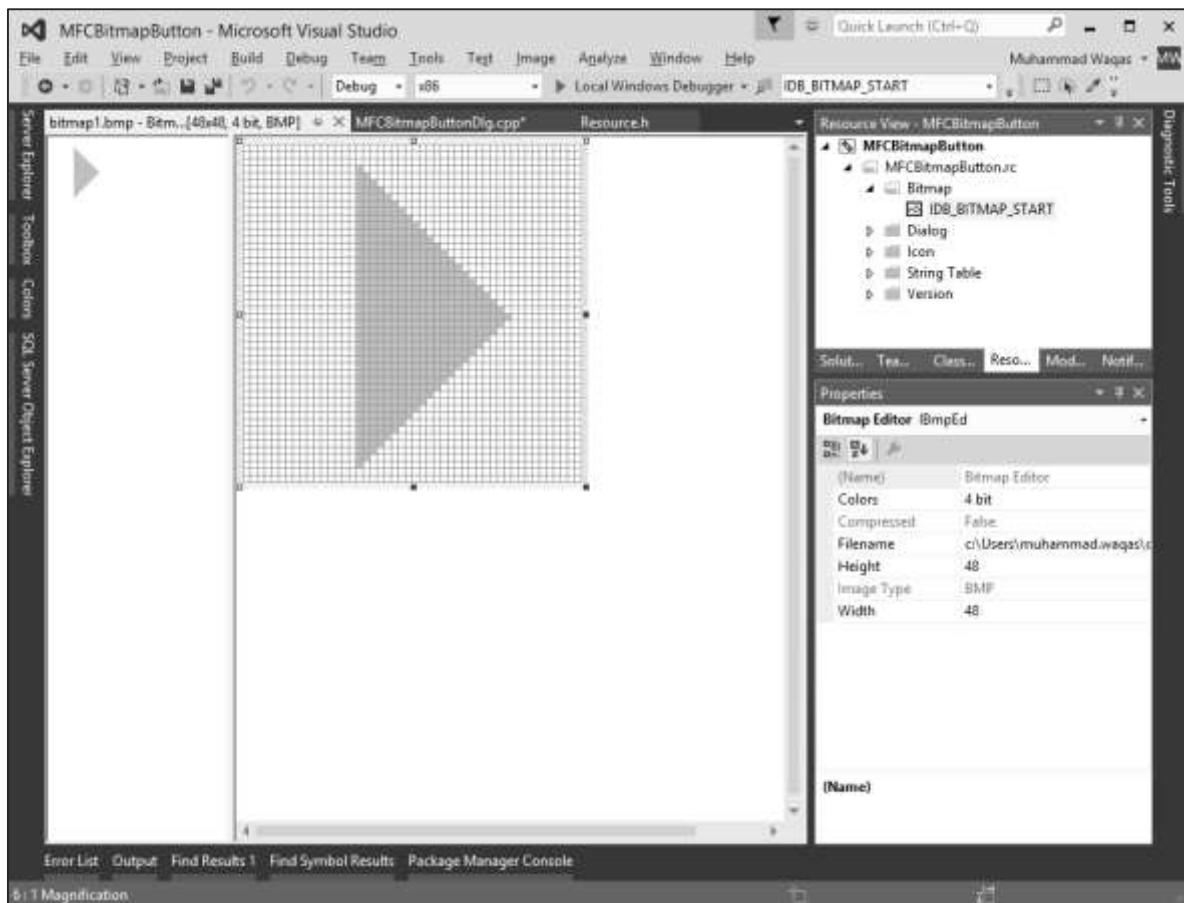
The **Image editor** has an extensive set of tools for creating and editing images, as well as features to help you create toolbar bitmaps. In addition to bitmaps, icons, and cursors, you can edit images in GIF or JPEG format using commands on the Image menu and tools on the Image Editor Toolbar.

Let us look into a simple example by creating a new project.

**Step 1:** Add a Bitmap from Add Resource dialog box.



**Step 2:** Select Bitmap and click New. It will open the Image editor.



**Step 3:** Design your bitmap image in Image editor and change its ID to IDB\_BITMAP\_START as shown above.

**Step 4:** Add a button to your dialog box and also add a control Variable m\_buttonStart for that button.

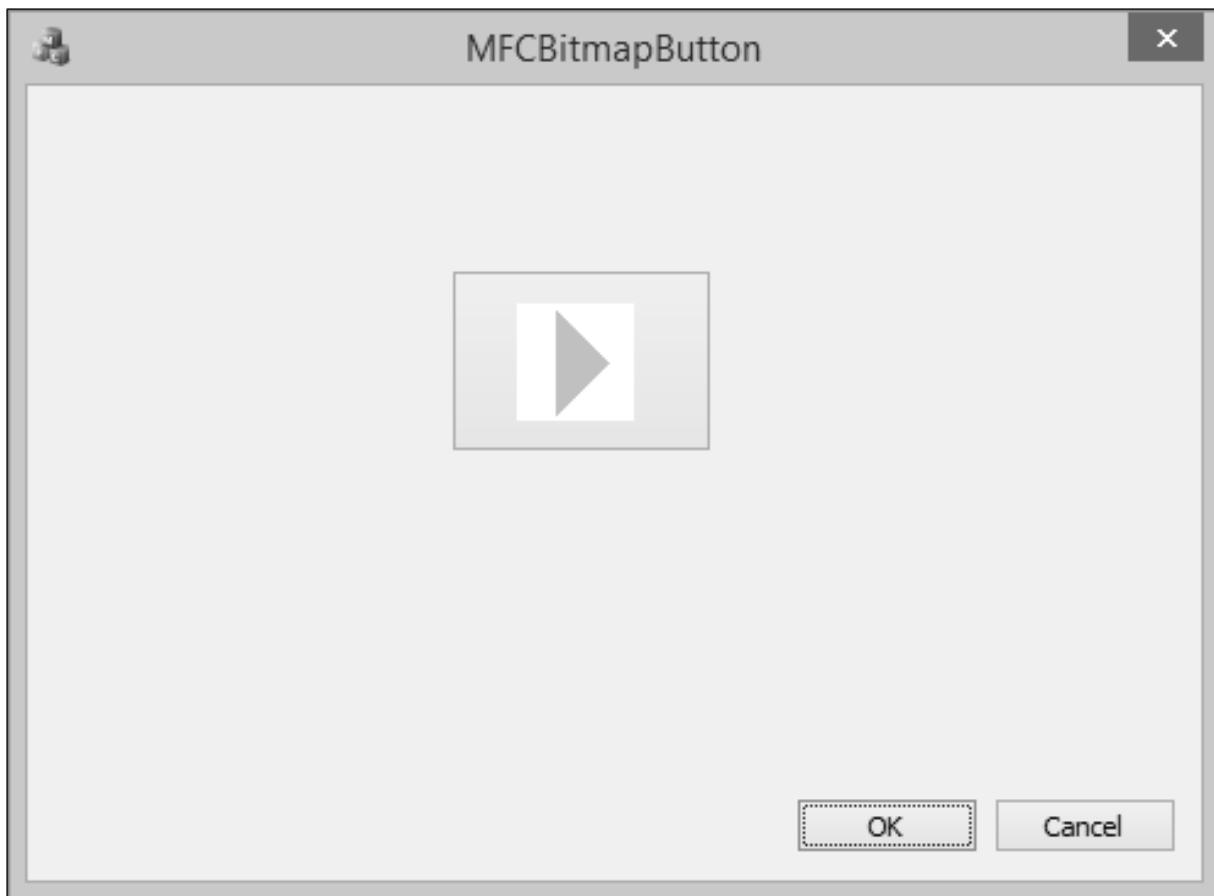
**Step 5:** Add a bitmap variable to your header file. You will now see the following two variables.

```
CBitmap m_bitmapStart;
CButton m_buttonStart;
```

**Step 6:** Modify your OnInitDialog() method as shown in the following code.

```
m_bitmapStart.LoadBitmap(IDB_BITMAP_START);
HBITMAP hBitmap = (HBITMAP)m_bitmapStart.GetSafeHandle();
m_buttonStart.SetBitmap(hBitmap);
```

**Step 7:** When the above code is compiled and executed, you will see the following output.



## Slider Controls

A **Slider Control** (also known as a trackbar) is a window containing a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the control sends notification messages to indicate the change. There are two types of sliders — horizontal and vertical. It is represented by **CSliderCtrl class**.

Here is the list of methods in CSliderCtrl class:

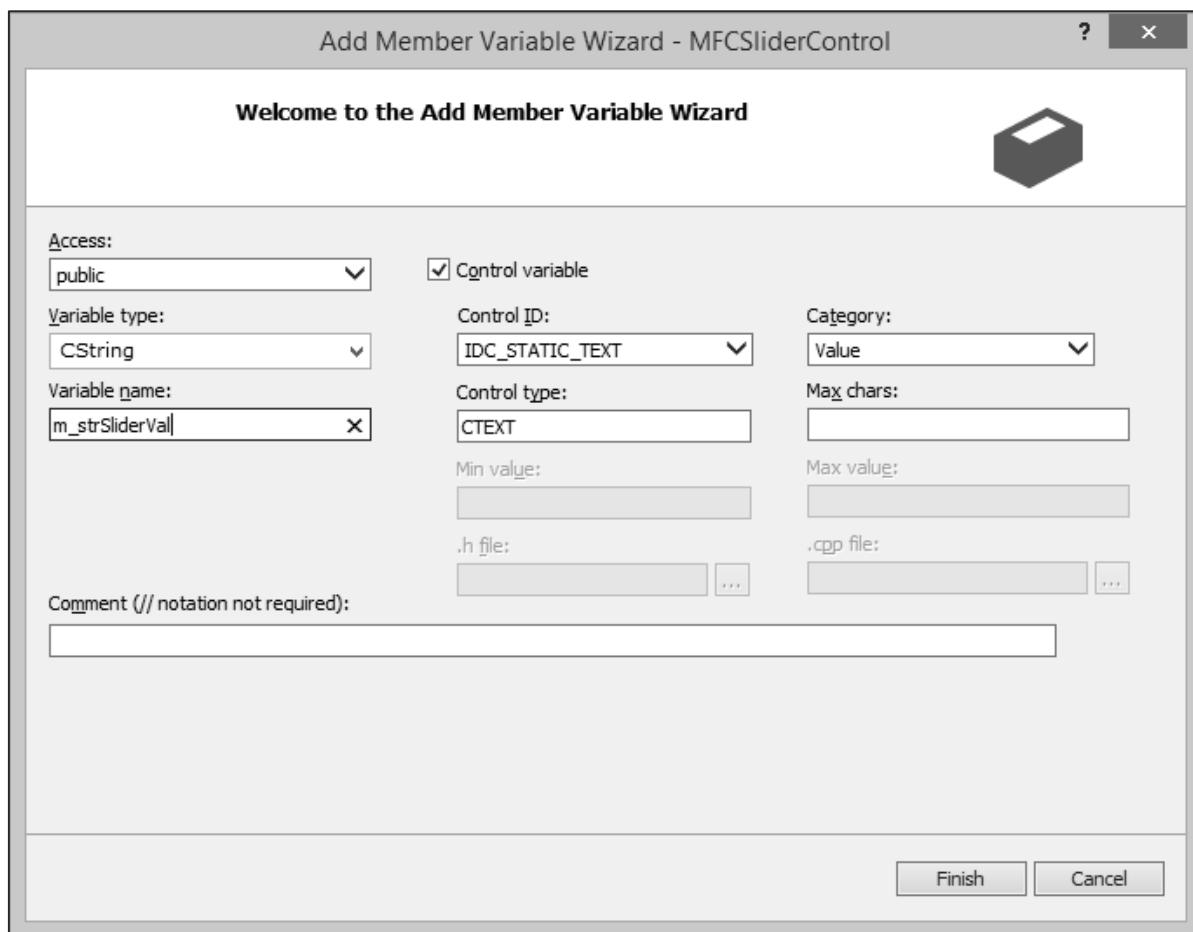
Name	Description
<b>ClearSel</b>	Clears the current selection in a slider control.
<b>ClearTics</b>	Removes the current tick marks from a slider control.
<b>Create</b>	Creates a slider control and attaches it to a CSliderCtrl object.
<b>CreateEx</b>	Creates a slider control with the specified Windows extended styles and attaches it to a CSliderCtrl object.
<b>GetBuddy</b>	Retrieves the handle to a slider control buddy window at a given location.
<b>GetChannelRect</b>	Retrieves the size of the slider control's channel.
<b>GetLineSize</b>	Retrieves the line size of a slider control.
<b>GetNumTics</b>	Retrieves the number of tick marks in a slider control.

<b>GetPageSize</b>	Retrieves the page size of a slider control.
<b>GetPos</b>	Retrieves the current position of the slider.
<b>GetRange</b>	Retrieves the minimum and maximum positions for a slider.
<b>GetRangeMax</b>	Retrieves the maximum position for a slider.
<b>GetRangeMin</b>	Retrieves the minimum position for a slider.
<b>GetSelection</b>	Retrieves the range of the current selection.
<b>GetThumbLength</b>	Retrieves the length of the slider in the current trackbar control.
<b>GetThumbRect</b>	Retrieves the size of the slider control's thumb.
<b>GetTic</b>	Retrieves the position of the specified tick mark.
<b>GetTicArray</b>	Retrieves the array of tick mark positions for a slider control.
<b>GetTicPos</b>	Retrieves the position of the specified tick mark, in client coordinates.
<b>GetToolTips</b>	Retrieves the handle to the tooltip control assigned to the slider control, if any.
<b>SetBuddy</b>	Assigns a window as the buddy window for a slider control.
<b>SetLineSize</b>	Sets the line size of a slider control.
<b>SetPageSize</b>	Sets the page size of a slider control.
<b>SetPos</b>	Sets the current position of the slider.
<b>SetRange</b>	Sets the minimum and maximum positions for a slider.
<b>SetRangeMax</b>	Sets the maximum position for a slider.
<b>SetRangeMin</b>	Sets the minimum position for a slider.
<b>SetSelection</b>	Sets the range of the current selection.
<b>SetThumbLength</b>	Sets the length of the slider in the current trackbar control.
<b>SetTic</b>	Sets the position of the specified tick mark.
<b>SetTicFreq</b>	Sets the frequency of tick marks per slider control increment.
<b>SetTipSide</b>	Positions a tooltip control used by a trackbar control.
<b>SetToolTips</b>	Assigns a tooltip control to a slider control.

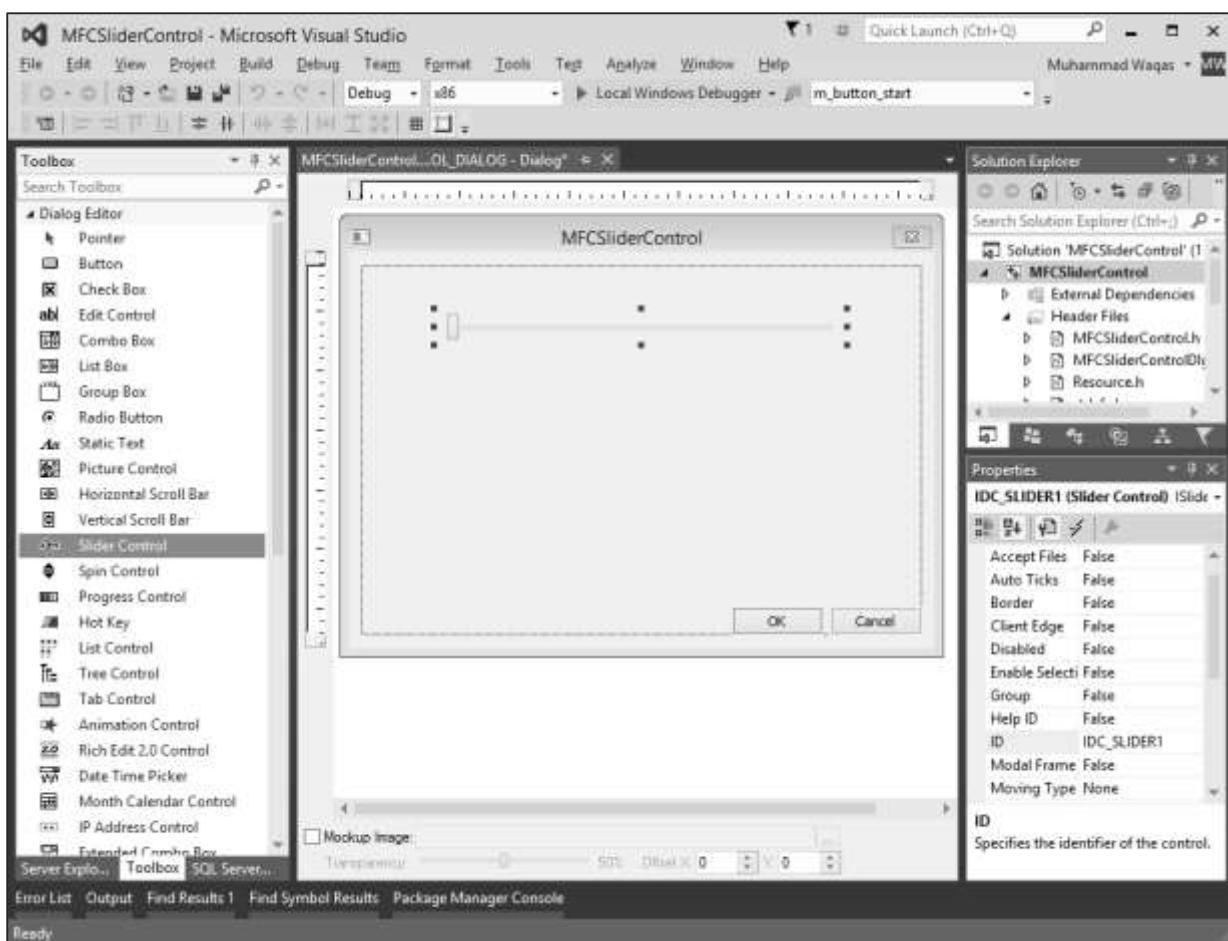
Let us look into a simple example by creating a new MFC dialog based project.

**Step 1:** Once the project is created you will see the TODO line which is the Caption of Text Control. Remove the Caption and set its ID to IDC\_STATIC\_TXT.

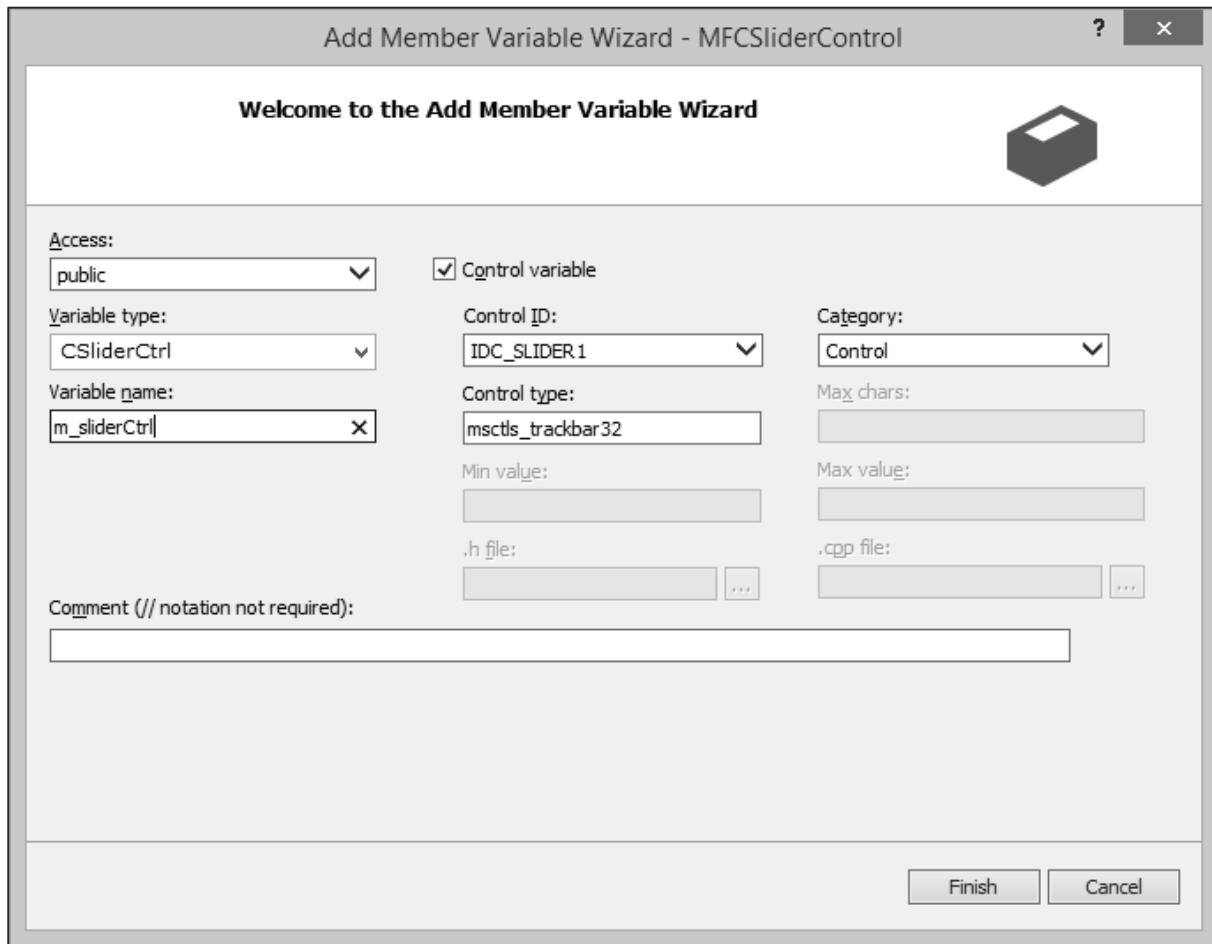
**Step 2:** Add a value variable m\_strSliderVal for the Static Text control.



**Step 3:** Drag the slider control from the Toolbox.



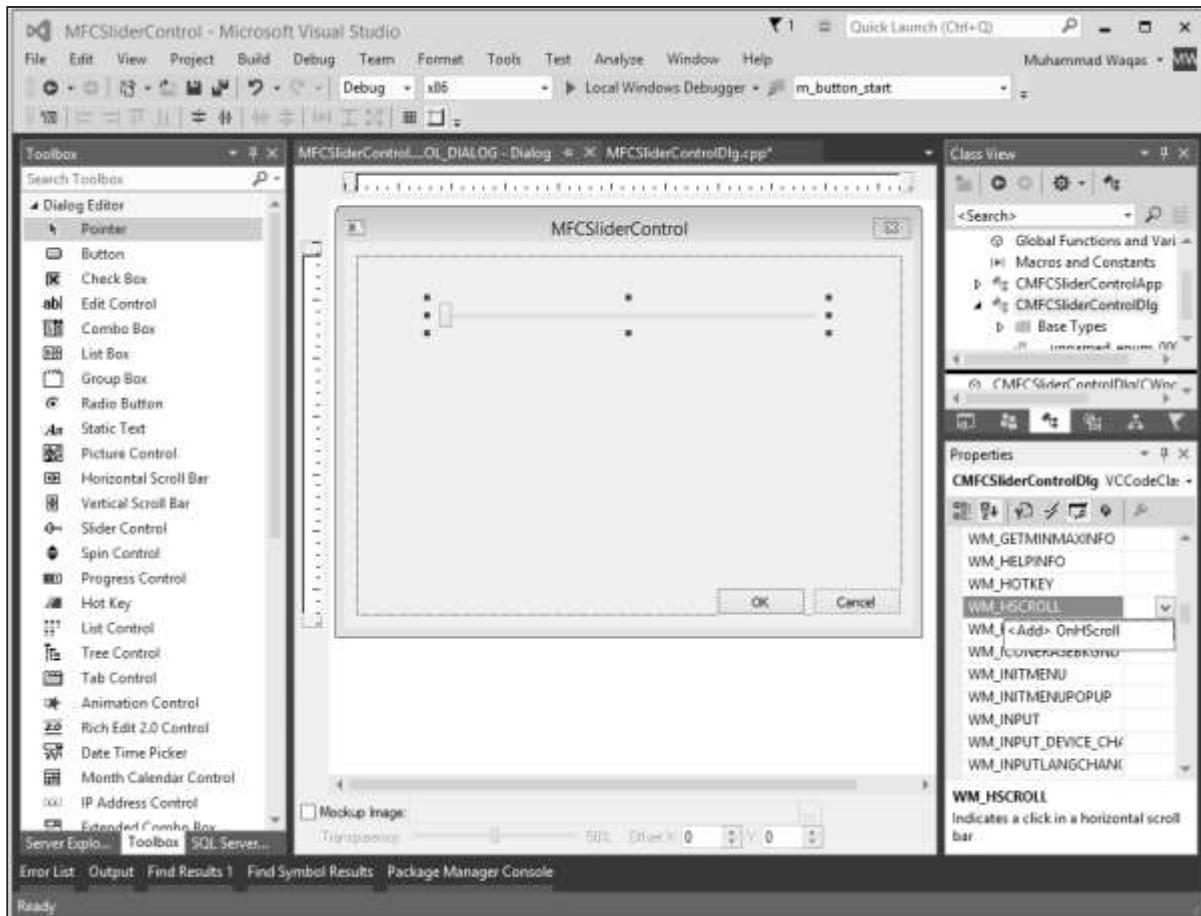
**Step 4:** Add a control variable m\_sliderCtrl for slider.



**Step 5:** Go to the class view in solution.

**Step 6:** Select the CMFCSliderControlDlg class.

**Step 7:** In the Properties window, click Messages.



**Step 8:** Scroll down to "WM\_HSCROLL" and click on the drop-down menu. Click "<Add> OnHScroll".

**Step 9:** Initialize the Slider and Static Text control inside the OnInitDialog() function.

```
BOOL CMFCSliderControlDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    m_sliderCtrl.SetRange(0, 100, TRUE);
}
```

233

```

m_sliderCtrl.SetPos(0);
m_strSliderVal.Format(_T("%d"), 0);

return TRUE; // return TRUE unless you set the focus to a control
}

```

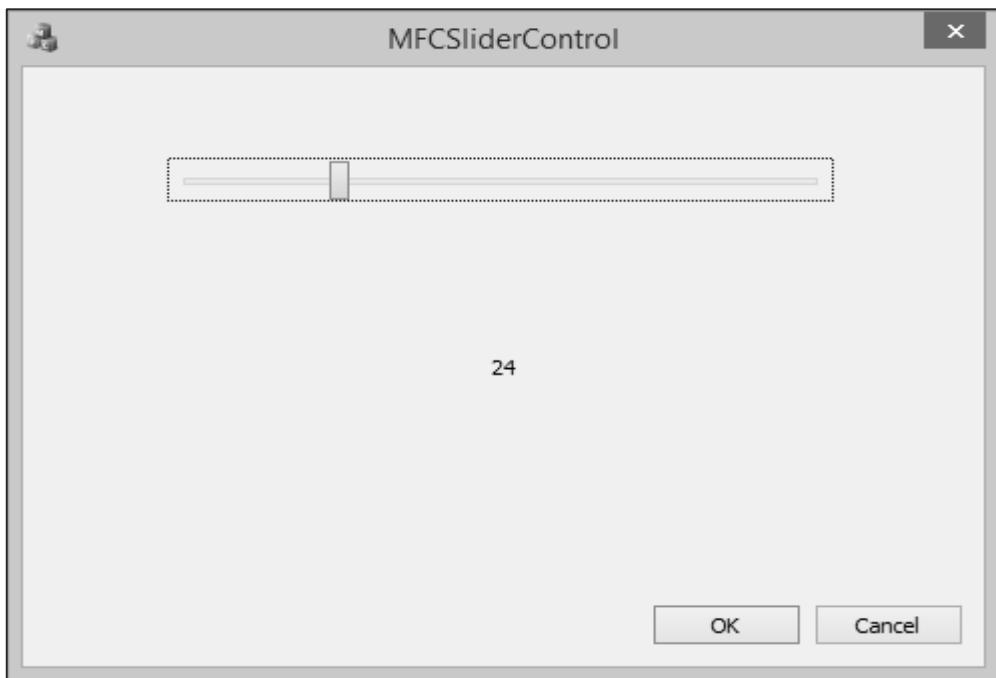
**Step 10:** Add the following code inside the function code block for OnVScroll()

```

void CMFCSliderControlDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default
    if (pScrollBar == (CScrollBar *)&m_sliderCtrl)
    {
        int value = m_sliderCtrl.GetPos();
        m_strSliderVal.Format(_T("%d"), value);
        UpdateData(FALSE);
    }
    else {
        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
    }
}

```

**Step 11:** When the above code is compiled and executed, you will see the following output.



## Scrollbars

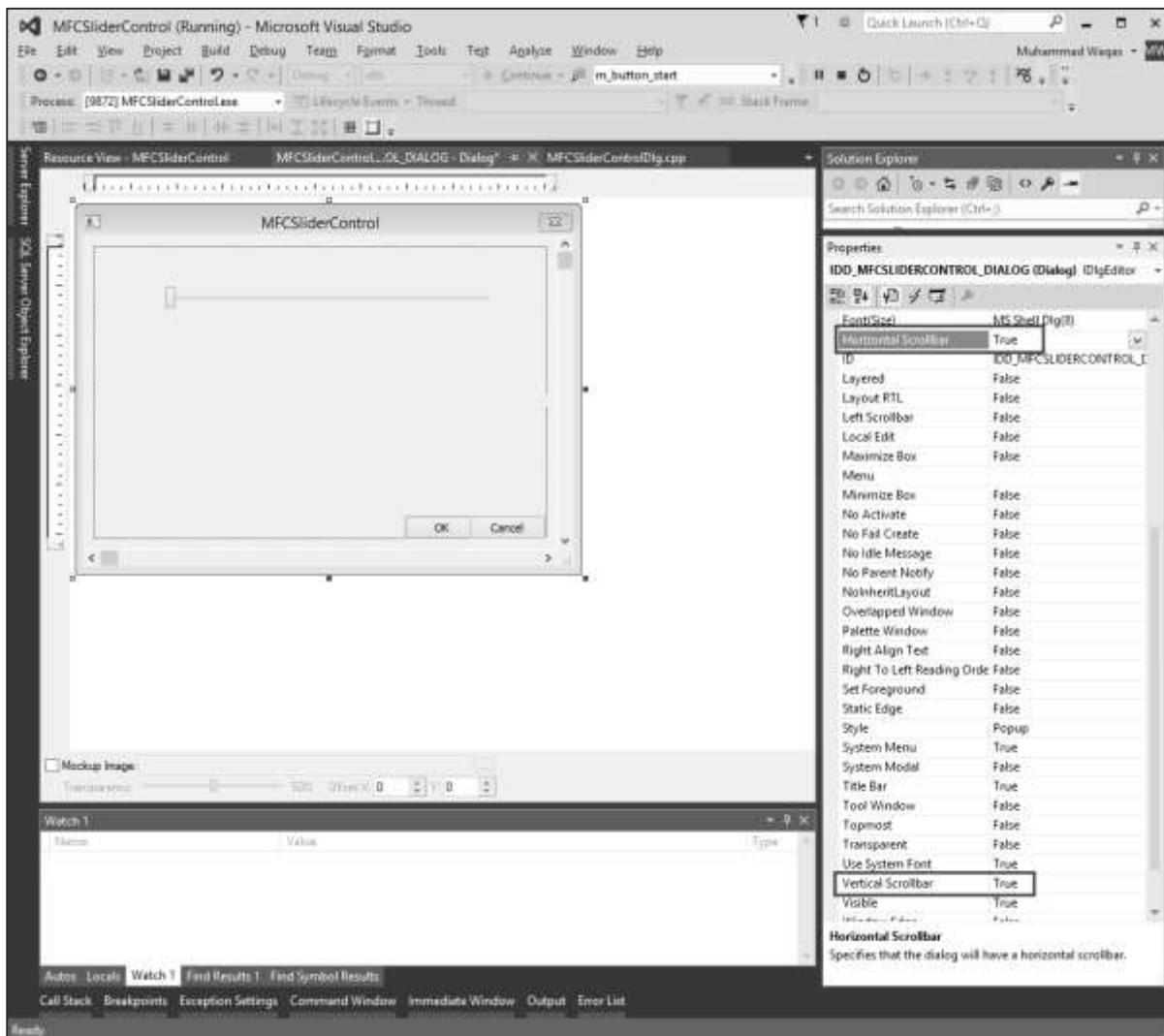
A **scrollbar** is a graphical control element with which continuous text, pictures or anything else can be scrolled in two directions along a control by clicking an arrow. This control can assume one of two directions — horizontal or vertical. It is represented by **CScrollBar** class.

Here is the list of methods in CScrollBar class:

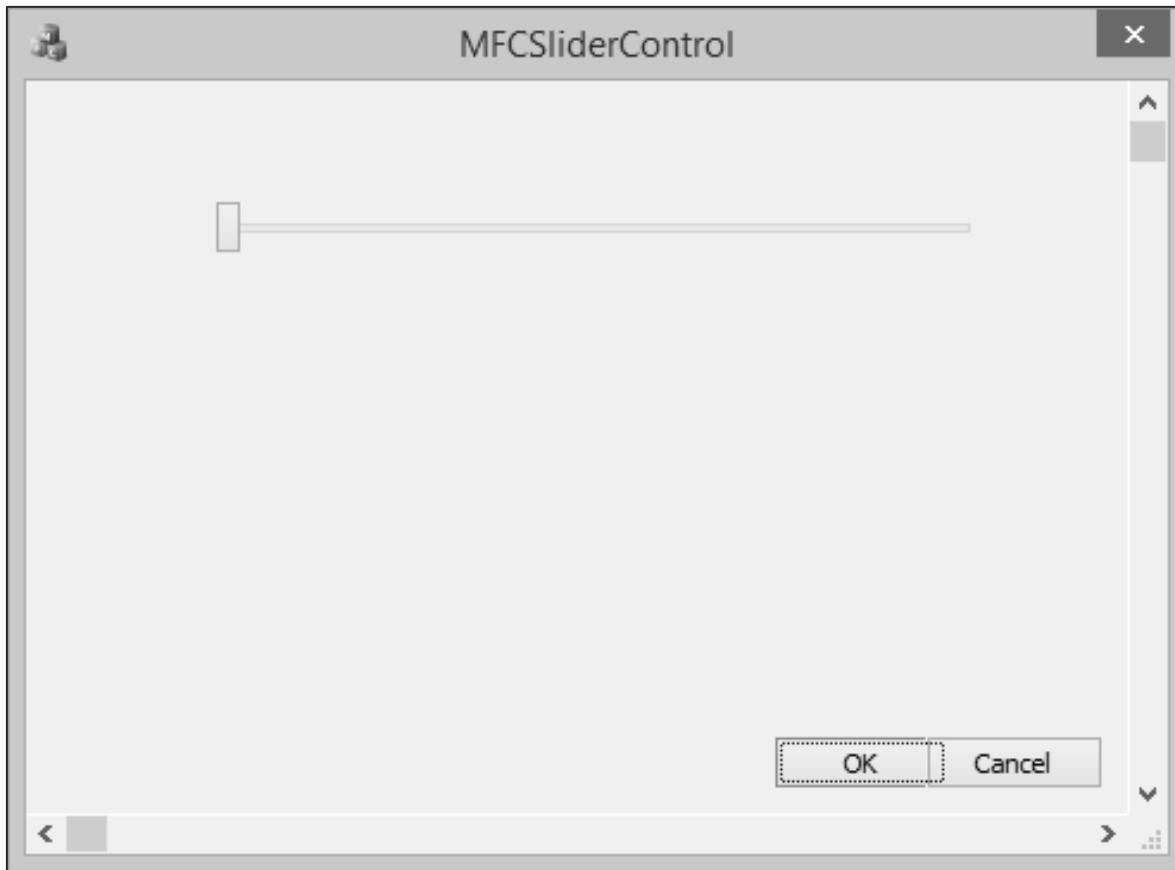
Name	Description
<b>Create</b>	Creates the Windows scroll bar and attaches it to the CScrollBar object.
<b>EnableScrollBar</b>	Enables or disables one or both arrows of a scroll bar.
<b>GetScrollBarInfo</b>	Retrieves information about the scroll bar using a <b>SCROLLBARINFO</b> structure.
<b>GetScrollInfo</b>	Retrieves information about the scroll bar.
<b>GetScrollLimit</b>	Retrieves the limit of the scroll bar
<b>GetScrollPos</b>	Retrieves the current position of a scroll box.
<b>GetScrollRange</b>	Retrieves the current minimum and maximum scroll-bar positions for the given scroll bar.
<b>SetScrollInfo</b>	Sets information about the scroll bar.
<b>SetScrollPos</b>	Sets the current position of a scroll box.
<b>SetScrollRange</b>	Sets minimum and maximum position values for the given scroll bar.
<b>ShowScrollBar</b>	Shows or hides a scroll bar.

Let us look into a simple example of Scrollbar.

**Step 1:** To add either horizontal or vertical scrollbar, you need to set the following highlighted properties of the dialog box to True.



**Step 2:** When you run the above application, you will see that both horizontal and vertical scrollbars have been added.



## Tree Control

A **Tree View Control** is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it. By clicking an item, the user can expand and collapse the associated list of subitems. It is represented by **CTreeCtrl** class.

Here is the list of methods in CTreeCtrl class:

Name	Description
<b>Create</b>	Creates a tree view control and attaches it to a CTreeCtrl object.
<b>CreateDragImage</b>	Creates a dragging bitmap for the specified tree view item.
<b>CreateEx</b>	Creates a tree control with the specified Windows extended styles and attaches it to a CTreeCtrl object.
<b>DeleteAllItems</b>	Deletes all items in a tree view control.
<b>DeleteItem</b>	Deletes a new item in a tree view control.
<b>EditLabel</b>	Edits a specified tree view item in-place.

<b>EndEditLabelInOW</b>	Cancels the edit operation on the label of a tree-view item in the current tree-view control.
<b>EnsureVisible</b>	Ensures that a tree view item is visible in its tree view control.
<b>Expand</b>	Expands, or collapses, the child items of the specified tree view item.
<b>GetBkColor</b>	Retrieves the current background color of the control.
<b>GetCheck</b>	Retrieves the check state of a tree control item.
<b>GetChildItem</b>	Retrieves the child of a specified tree view item.
<b>GetCount</b>	Retrieves the number of tree items associated with a tree view control.
<b>GetDropHiligh tItem</b>	Retrieves the target of a drag-and-drop operation.
<b>GetEditControl</b>	Retrieves the handle of the edit control used to edit the specified tree view item.
<b>GetExtendedS tyle</b>	Retrieves the extended styles that the current tree-view control is using.
<b>GetFirstVisible Item</b>	Retrieves the first visible item of the specified tree view item.
<b>GetImageList</b>	Retrieves the handle of the image list associated with a tree view control.
<b>GetIndent</b>	Retrieves the offset (in pixels) of a tree view item from its parent.
<b>GetInsertMark Color</b>	Retrieves the color used to draw the insertion mark for the tree view.
<b>GetItem</b>	Retrieves the attributes of a specified tree view item.
<b>GetItemData</b>	Returns the 32-bit application-specific value associated with an item.
<b>GetItemExpan dedImageInd ex</b>	Retrieves the index of the image to display when the specified item of the current tree-view control is in the expanded state.
<b>GetItemHeigh t</b>	Retrieves the current height of the tree view items.
<b>GetItemImage</b>	Retrieves the images associated with an item.
<b>GetItemPartR ect</b>	Retrieves the bounding rectangle for a specified part of a specified item in the current tree-view control.
<b>GetItemRect</b>	Retrieves the bounding rectangle of a tree view item.
<b>GetItemState</b>	Returns the state of an item.
<b>GetItemState Ex</b>	Retrieves the extended state of the specified item in the current tree-view control.
<b>GetItemText</b>	Returns the text of an item.
<b>GetLastVisible Item</b>	Retrieves the last expanded item in the current tree-view control.
<b>GetLineColor</b>	Retrieves the current line color for the tree view control.
<b>GetNextItem</b>	Retrieves the next tree view item that matches a specified relationship.
<b>GetNextSiblin gItem</b>	Retrieves the next sibling of the specified tree view item.
<b>GetNextVisibl eItem</b>	Retrieves the next visible item of the specified tree view item.

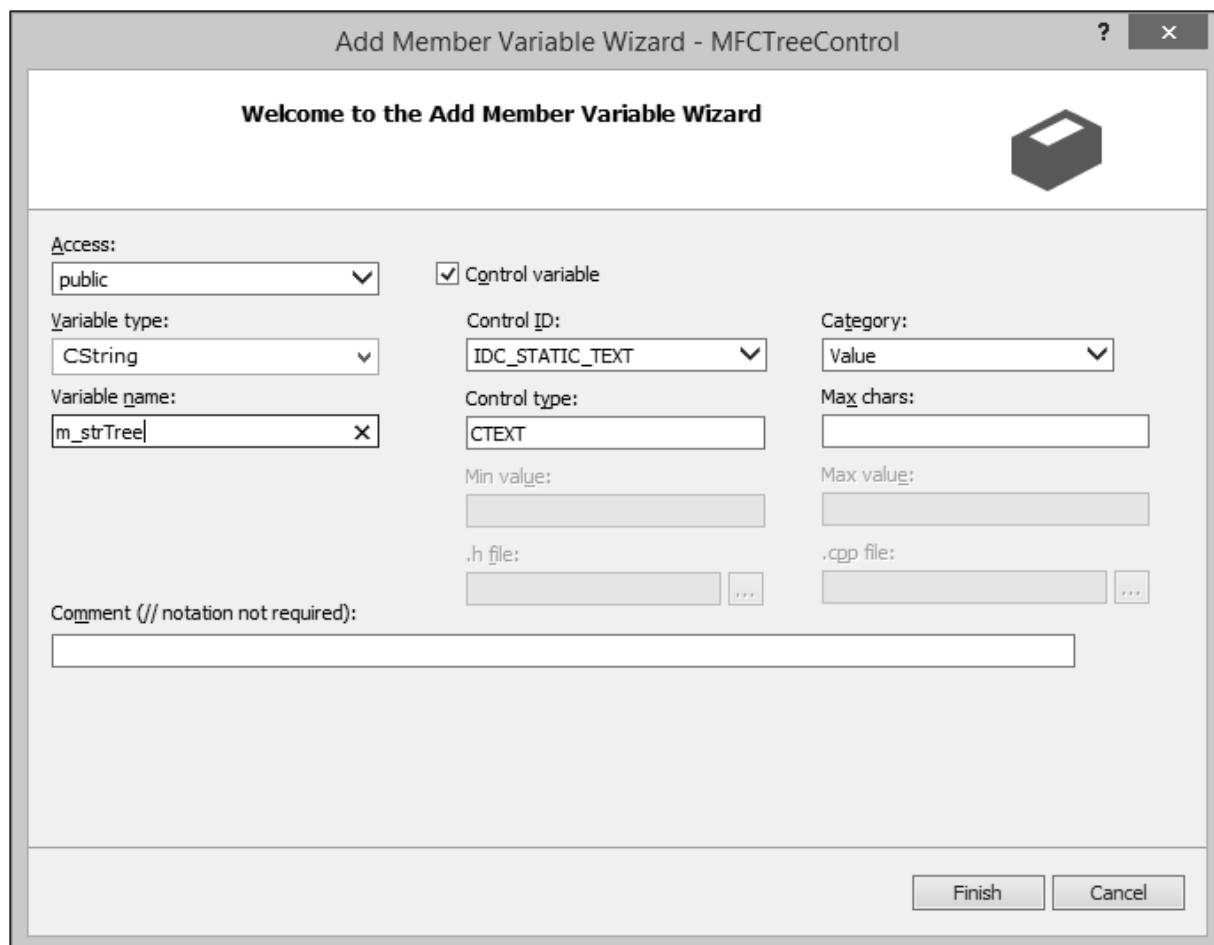
<b>GetParentItem</b>	Retrieves the parent of the specified tree view item.
<b>GetPrevSiblingItem</b>	Retrieves the previous sibling of the specified tree view item.
<b>GetPrevVisibleItem</b>	Retrieves the previous visible item of the specified tree view item.
<b>GetRootItem</b>	Retrieves the root of the specified tree view item.
<b>GetScrollTime</b>	Retrieves the maximum scroll time for the tree view control.
<b>GetSelectedCount</b>	Retrieves the number of selected items in the current tree-view control.
<b>GetSelectedItem</b>	Retrieves the currently selected tree view item.
<b>GetTextColor</b>	Retrieves the current text color of the control.
<b>GetToolTips</b>	Retrieves the handle to the child ToolTip control used by a tree view control.
<b>GetVisibleCount</b>	Retrieves the number of visible tree items associated with a tree view control.
<b>HitTest</b>	Returns the current position of the cursor related to the CTreeCtrl object.
<b>InsertItem</b>	Inserts a new item in a tree view control.
<b>ItemHasChildren</b>	Returns nonzero if the specified item has child items.
<b>MapAccIdToItem</b>	Maps the specified accessibility identifier to the handle to a tree-view item in the current tree-view control.
<b>MapItemToAccID</b>	Maps the specified handle to a tree-view item in the current tree-view control to an accessibility identifier.
<b>Select</b>	Selects, scrolls into view, or redraws a specified tree view item.
<b>SelectDropTarget</b>	Redraws the tree item as the target of a drag-and-drop operation.
<b>SelectItem</b>	Selects a specified tree view item.
<b>SelectSetFirstVisible</b>	Selects a specified tree view item as the first visible item.
<b>SetAutoscrollInfo</b>	Sets the autoscroll rate of the current tree-view control.
<b>SetBkColor</b>	Sets the background color of the control.
<b>SetCheck</b>	Sets the check state of a tree control item.
<b>SetExtendedStyle</b>	Sets the extended styles for the current tree-view control.
<b>SetImageList</b>	Sets the handle of the image list associated with a tree view control.
<b>SetIndent</b>	Sets the offset (in pixels) of a tree view item from its parent.
<b>SetInsertMark</b>	Sets the insertion mark in a tree view control.
<b>SetInsertMarkColor</b>	Sets the color used to draw the insertion mark for the tree view.
<b>SetItem</b>	Sets the attributes of a specified tree view item.
<b>SetItemData</b>	Sets the 32-bit application-specific value associated with an item.
<b>SetItemExpandedImageIndex</b>	Sets the index of the image to display when the specified item of the current tree-view control is in the expanded state.

<b>SetItemHeight</b>	Sets the height of the tree view items.
<b>SetItemImage</b>	Associates images with an item.
<b>SetItemState</b>	Sets the state of an item.
<b>SetItemStateEx</b>	Sets the extended state of the specified item in the current tree-view control.
<b>SetItemText</b>	Sets the text of an item.
<b>SetLineColor</b>	Sets the current line color for the tree view control.
<b>SetScrollTime</b>	Sets the maximum scroll time for the tree view control.
<b>SetTextColor</b>	Sets the text color of the control.
<b>SetToolTips</b>	Sets a tree view control's child ToolTip control.
<b>ShowInfoTip</b>	Displays the infotip for the specified item in the current tree-view control.
<b>SortChildren</b>	Sorts the children of a given parent item.
<b>SortChildrenCB</b>	Sorts the children of a given parent item using an application-defined sort function.

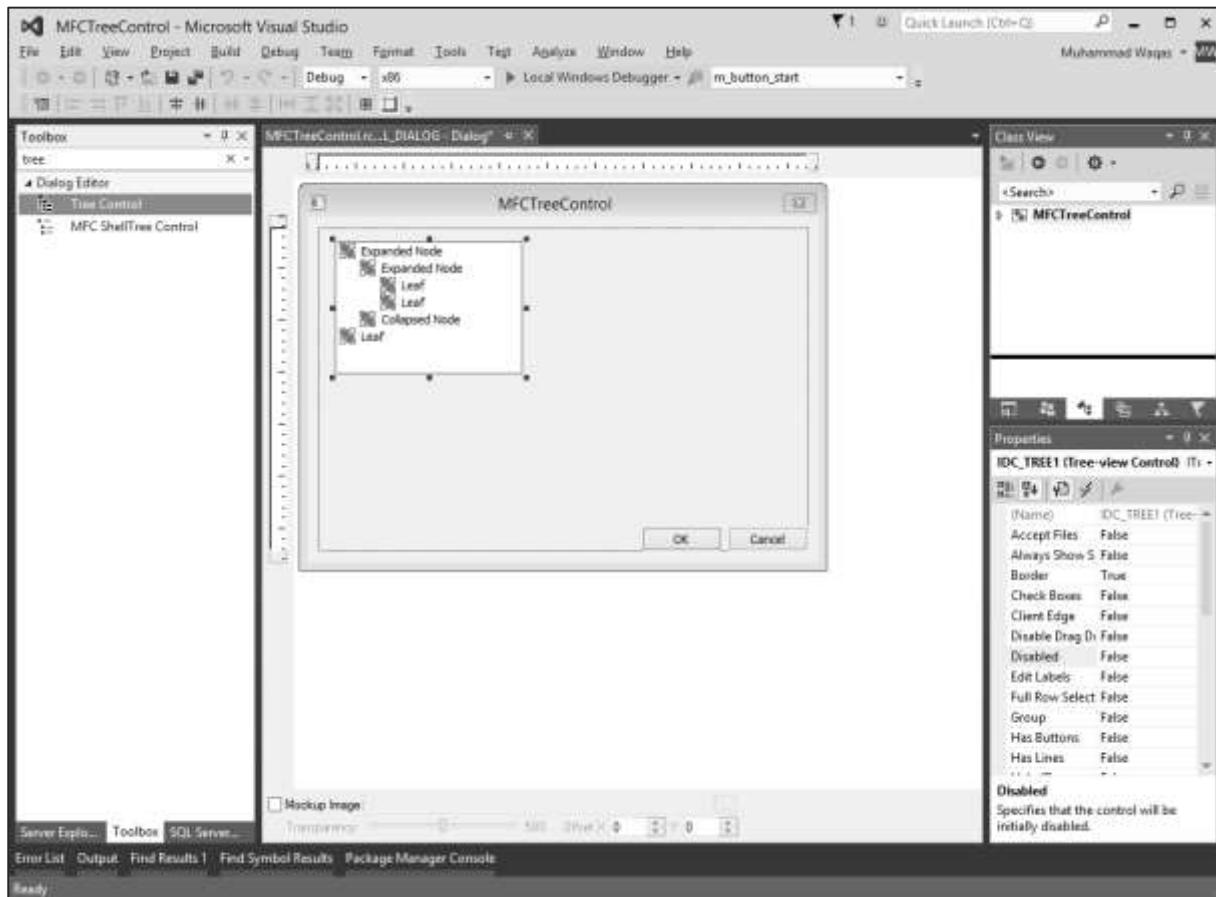
Let us look into a simple example by creating a new MFC dialog based project.

**Step 1:** Once the project is created you will see the TODO line, which is the Caption of Text Control. Remove the Caption and set its ID to IDC\_STATIC\_TXT.

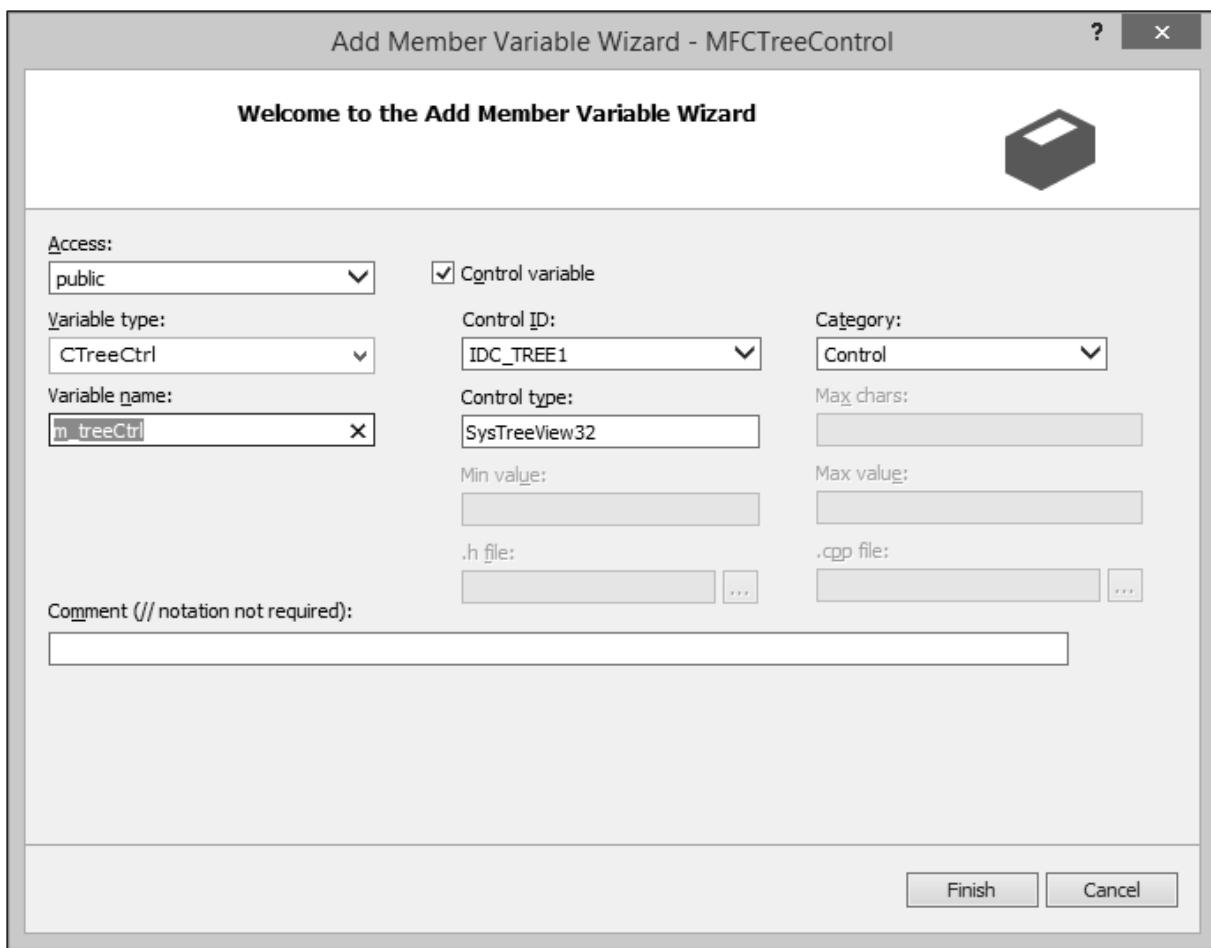
**Step 2:** Add a value variable m\_strTree for the Static Text control.



**Step 3:** From the Controls toolbox, drag the Tree Control.



**Step 4:** On the dialog box, click the Tree Control to select it. On the Properties window, set the Has Buttons, the Has Lines, the Lines At Root, the Client Edge and the Modal Frame properties to True.

**Step 5:** Add a control variable m\_treeCtrl for Tee Control.**Step 6:** Here is initialization of tree control in OnInitDialog()

```
BOOL CMFCTreeControlDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

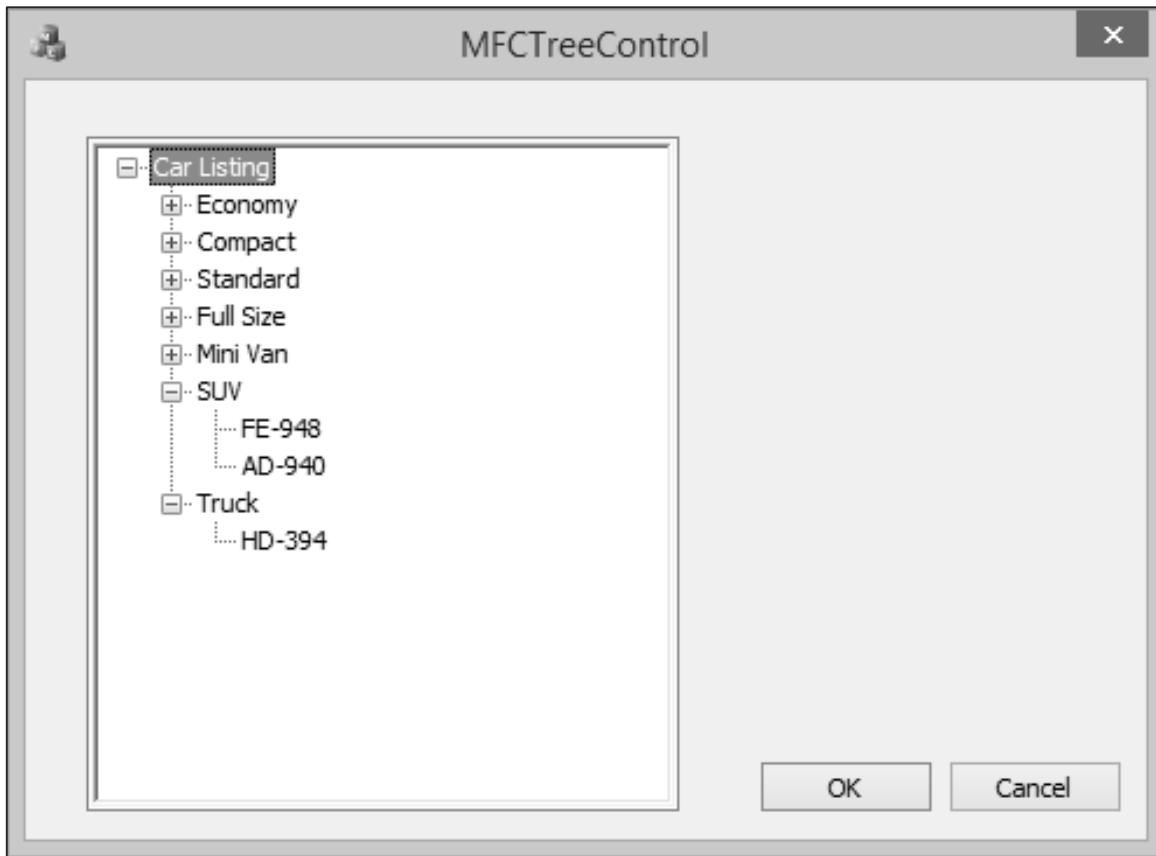
    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    HTREEITEM hItem, hCar;
    hItem = m_treeCtrl.InsertItem(L"Car Listing", TVI_ROOT);
    hCar = m_treeCtrl.InsertItem(L"Economy", hItem);
    m_treeCtrl.InsertItem(L"BH-733", hCar);
}
```

```
m_treeCtrl.InsertItem(L"SD-397", hCar);
m_treeCtrl.InsertItem(L"JU-538", hCar);
m_treeCtrl.InsertItem(L"DI-285", hCar);
m_treeCtrl.InsertItem(L"AK-830", hCar);
hCar = m_treeCtrl.InsertItem(L"Compact", hItem);
m_treeCtrl.InsertItem(L"HG-490", hCar);
m_treeCtrl.InsertItem(L"PE-473", hCar);
hCar = m_treeCtrl.InsertItem(L"Standard", hItem);
m_treeCtrl.InsertItem(L"SO-398", hCar);
m_treeCtrl.InsertItem(L"DF-438", hCar);
m_treeCtrl.InsertItem(L"IS-833", hCar);
hCar = m_treeCtrl.InsertItem(L"Full Size", hItem);
m_treeCtrl.InsertItem(L"PD-304", hCar);
hCar = m_treeCtrl.InsertItem(L"Mini Van", hItem);
m_treeCtrl.InsertItem(L"ID-497", hCar);
m_treeCtrl.InsertItem(L"RU-304", hCar);
m_treeCtrl.InsertItem(L"DK-905", hCar);
hCar = m_treeCtrl.InsertItem(L"SUV", hItem);
m_treeCtrl.InsertItem(L"FE-948", hCar);
m_treeCtrl.InsertItem(L"AD-940", hCar);
hCar = m_treeCtrl.InsertItem(L"Truck", hItem);
m_treeCtrl.InsertItem(L"HD-394", hCar);

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 7:** When the above code is compiled and executed, you will see the following output.



## List Control

Encapsulates the functionality of a List View Control, which displays a collection of items each consisting of an icon (from an image list) and a label. It is represented by **CListCtrl** class. A list control consists of using one of four views to display a list of items.

- Icons
- Small Icons
- List
- Report

Here is the list of methods in **CListCtrl** class.

Name	Description
<b>ApproximateViewRect</b>	Determines the width and height required to display the items of a list view control.
<b>Arrange</b>	Aligns items on a grid.
<b>CancelEditLabel</b>	Cancels item text editing operation.
<b>Create</b>	Creates a list control and attaches it to a CListCtrl object.
<b>CreateDragImage</b>	Creates a drag image list for a specified item.
<b>CreateEx</b>	Creates a list control with the specified Windows extended styles and attaches it to a CListCtrl object.

<b>DeleteAllItems</b>	Deletes all items from the control.
<b>DeleteColumn</b>	Deletes a column from the list view control.
<b>DeleteItem</b>	Deletes an item from the control.
<b>DrawItem</b>	Called when a visual aspect of an owner-draw control changes.
<b>EditLabel</b>	Begins in-place editing of an item's text.
<b>EnableGroupView</b>	Enables or disables whether the items in a list view control display as a group.
<b>EnsureVisible</b>	Ensures that an item is visible.
<b>FindItem</b>	Searches for a list view item having specified characteristics.
<b>GetBkColor</b>	Retrieves the background color of a list view control.
<b>GetBkImage</b>	Retrieves the current background image of a list view control.
<b>GetCallbackMask</b>	Retrieves the callback mask for a list view control.
<b>GetCheck</b>	Retrieves the current display status of the state image associated with an item.
<b>GetColumn</b>	Retrieves the attributes of a control's column.
<b>GetColumnOrderArray</b>	Retrieves the column order (left to right) of a list view control.
<b>GetColumnWidth</b>	Retrieves the width of a column in report view or list view.
<b>GetCountPerPage</b>	Calculates the number of items that can fit vertically in a list view control.
<b>GetEditControl</b>	Retrieves the handle of the edit control used to edit an item's text.
<b>GetEmptyText</b>	Retrieves the string to display if the current list-view control is empty.
<b>GetExtendedStyle</b>	Retrieves the current extended styles of a list view control.
<b>GetFirstSelectedItemPosition</b>	Retrieves the position of the first selected list view item in a list view control.
<b>GetFocusedGroup</b>	Retrieves the group that has the keyboard focus in the current list-view control.
<b>GetGroupCount</b>	Retrieves the number of groups in the current list-view control.
<b>GetGroupInfo</b>	Gets the information for a specified group of the list view control.
<b>GetGroupInfoByIndex</b>	Retrieves information about a specified group in the current list-view control.
<b>GetGroupMetrics</b>	Retrieves the metrics of a group.
<b>GetGroupRect</b>	Retrieves the bounding rectangle for a specified group in the current list-view control.
<b>GetGroupState</b>	Retrieves the state for a specified group in the current list-view control.
<b>GetHeaderCtrl</b>	Retrieves the header control of a list view control.
<b>GetHotCursor</b>	Retrieves the cursor used when hot tracking is enabled for a list view control.
<b>GetHotItem</b>	Retrieves the list view item currently under the cursor.
<b>GetHoverTime</b>	Retrieves the current hover time of a list view control.

<b>GetImageList</b>	Retrieves the handle of an image list used for drawing list view items.
<b>GetInsertMark</b>	Retrieves the current position of the insertion mark.
<b>GetInsertMarkColor</b>	Retrieves the current color of the insertion mark.
<b>GetInsertMarkRect</b>	Retrieves the rectangle that bounds the insertion point.
<b>GetItem</b>	Retrieves a list view item's attributes.
<b>GetItemCount</b>	Retrieves the number of items in a list view control.
<b>GetItemData</b>	Retrieves the application-specific value associated with an item.
<b>GetItemIndexRect</b>	Retrieves the bounding rectangle for all or part of a subitem in the current list-view control.
<b>GetPosition</b>	Retrieves the position of a list view item.
<b>GetItemRect</b>	Retrieves the bounding rectangle for an item.
<b>GetItemSpacing</b>	Calculates the spacing between items in the current list-view control.
<b>GetItemState</b>	Retrieves the state of a list view item.
<b>GetItemText</b>	Retrieves the text of a list view item or subitem.
<b>GetNextItem</b>	Searches for a list view item with specified properties and with specified relationship to a given item.
<b>GetNextItemIndex</b>	Retrieves the index of the item in the current list-view control that has a specified set of properties.
<b>GetNextSelectedItem</b>	Retrieves the index of a list view item position, and the position of the next selected list view item for iterating.
<b>GetNumberOfWorkAreas</b>	Retrieves the current number of working areas for a list view control.
<b>GetOrigin</b>	Retrieves the current view origin for a list view control.
<b>GetOutlineColor</b>	Retrieves the color of the border of a list view control.
<b>GetSelectedColumn</b>	Retrieves the index of the currently selected column in the list control.
<b>GetSelectedCount</b>	Retrieves the number of selected items in the list view control.
<b>GetSelectionMark</b>	Retrieves the selection mark of a list view control.
<b>GetStringWidth</b>	Determines the minimum column width necessary to display all of a given string.
<b>GetSubItemRect</b>	Retrieves the bounding rectangle of an item in a list view control.
<b>GetTextBkColor</b>	Retrieves the text background color of a list view control.
<b>GetTextColor</b>	Retrieves the text color of a list view control.
<b>GetTileInfo</b>	Retrieves information about a tile in a list view control.
<b>GetTileViewInfo</b>	Retrieves information about a list view control in tile view.
<b>GetToolTips</b>	Retrieves the tooltip control that the list view control uses to display tooltips.
<b>GetTopIndex</b>	Retrieves the index of the topmost visible item.
<b>GetView</b>	Gets the view of the list view control.
<b>GetViewRect</b>	Retrieves the bounding rectangle of all items in the list view control.

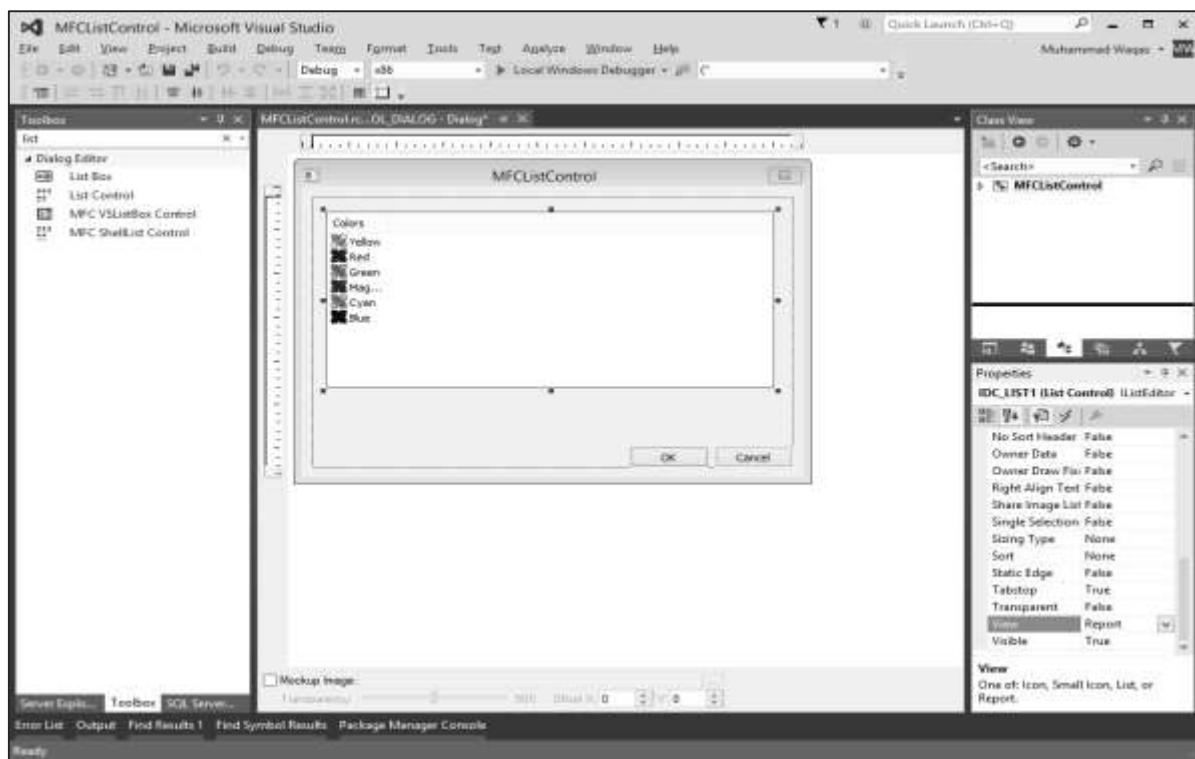
<b>GetWorkAreas</b>	Retrieves the current working areas of a list view control.
<b>HasGroup</b>	Determines if the list view control has the specified group.
<b>HitTest</b>	Determines which list view item is at a specified position.
<b>InsertColumn</b>	Inserts a new column in a list view control.
<b>InsertGroup</b>	Inserts a group into the list view control.
<b>InsertGroupSorted</b>	Inserts the specified group into an ordered list of groups.
<b>InsertItem</b>	Inserts a new item in a list view control.
<b>InsertMarkHitTest</b>	Retrieves the insertion point closest to a specified point.
<b>IsGroupViewEnabled</b>	Determines whether group view is enabled for a list view control.
<b>IsItemVisible</b>	Indicates whether a specified item in the current list-view control is visible.
<b>MapIDToIndex</b>	Maps the unique ID of an item in the current list-view control to an index.
<b>MapIndexToID</b>	Maps the index of an item in the current list-view control to a unique ID.
<b>MoveGroup</b>	Moves the specified group.
<b>MoveItemToGroup</b>	Moves the specified group to the specified zero based index of the list view control.
<b>RedrawItems</b>	Forces a list view control to repaint a range of items.
<b>RemoveAllGroups</b>	Removes all groups from a list view control.
<b>RemoveGroup</b>	Removes the specified group from the list view control.
<b>Scroll</b>	Scrolls the content of a list view control.
<b>SetBkColor</b>	Sets the background color of the list view control.
<b>SetBkImage</b>	Sets the current background image of a list view control.
<b>SetCallbackMask</b>	Sets the callback mask for a list view control.
<b>SetCheck</b>	Sets the current display status of the state image associated with an item.
<b>SetColumn</b>	Sets the attributes of a list view column.
<b>SetColumnOrderArray</b>	Sets the column order (left to right) of a list view control.
<b>SetColumnWidth</b>	Changes the width of a column in report view or list view.
<b>SetExtendedStyle</b>	Sets the current extended styles of a list view control.
<b>SetGroupInfo</b>	Sets the information for the specified group of a list view control.
<b>SetGroupMetrics</b>	Sets the group metrics of a list view control.
<b>SetHotCursor</b>	Sets the cursor used when hot tracking is enabled for a list view control.
<b>SetHotItem</b>	Sets the current hot item of a list view control.
<b>SetHoverTime</b>	Sets the current hover time of a list view control.
<b>SetIconSpacing</b>	Sets the spacing between icons in a list view control.
<b>SetImageList</b>	Assigns an image list to a list view control.
<b>SetInfoTip</b>	Sets the tooltip text.
<b>SetInsertMark</b>	Sets the insertion point to the defined position.

<b>SetInsertMarkColor</b>	Sets the color of the insertion point.
<b>SetItem</b>	Sets some or all of a list view item's attributes.
<b>SetItemCount</b>	Prepares a list view control for adding a large number of items.
<b>SetItemCountEx</b>	Sets the item count for a virtual list view control.
<b>SetItemData</b>	Sets the item's application-specific value.
<b>SetItemIndexState</b>	Sets the state of an item in the current list-view control.
<b>SetItemPosition</b>	Moves an item to a specified position in a list view control.
<b>SetItemState</b>	Changes the state of an item in a list view control.
<b>SetItemText</b>	Changes the text of a list view item or subitem.
<b>SetOutlineColor</b>	Sets the color of the border of a list view control.
<b>SetSelectedColumn</b>	Sets the selected column of the list view control.
<b>SetSelectionMark</b>	Sets the selection mark of a list view control.
<b>SetTextBkColor</b>	Sets the background color of text in a list view control.
<b>SetTextColor</b>	Sets the text color of a list view control.
<b>SetTileInfo</b>	Sets the information for a tile of the list view control.
<b>SetTileViewInfo</b>	Sets information that a list view control uses in tile view.
<b>SetToolTips</b>	Sets the tooltip control that the list view control will use to display tooltips.
<b>SetView</b>	Sets the view of the list view control.
<b>SetWorkAreas</b>	Sets the area where icons can be displayed in a list view control.
<b>SortGroups</b>	Sorts the groups of a list view control with a user-defined function.
<b>SortItems</b>	Sorts list view items using an application-defined comparison function.
<b>SortItemsEx</b>	Sorts list view items using an application-defined comparison function.
<b>SubItemHitTest</b>	Determines which list view item, if any, is at a given position.
<b>Update</b>	Forces the control to repaint a specified item.

Let us look into a simple example by creating a new MFC dialog based application.

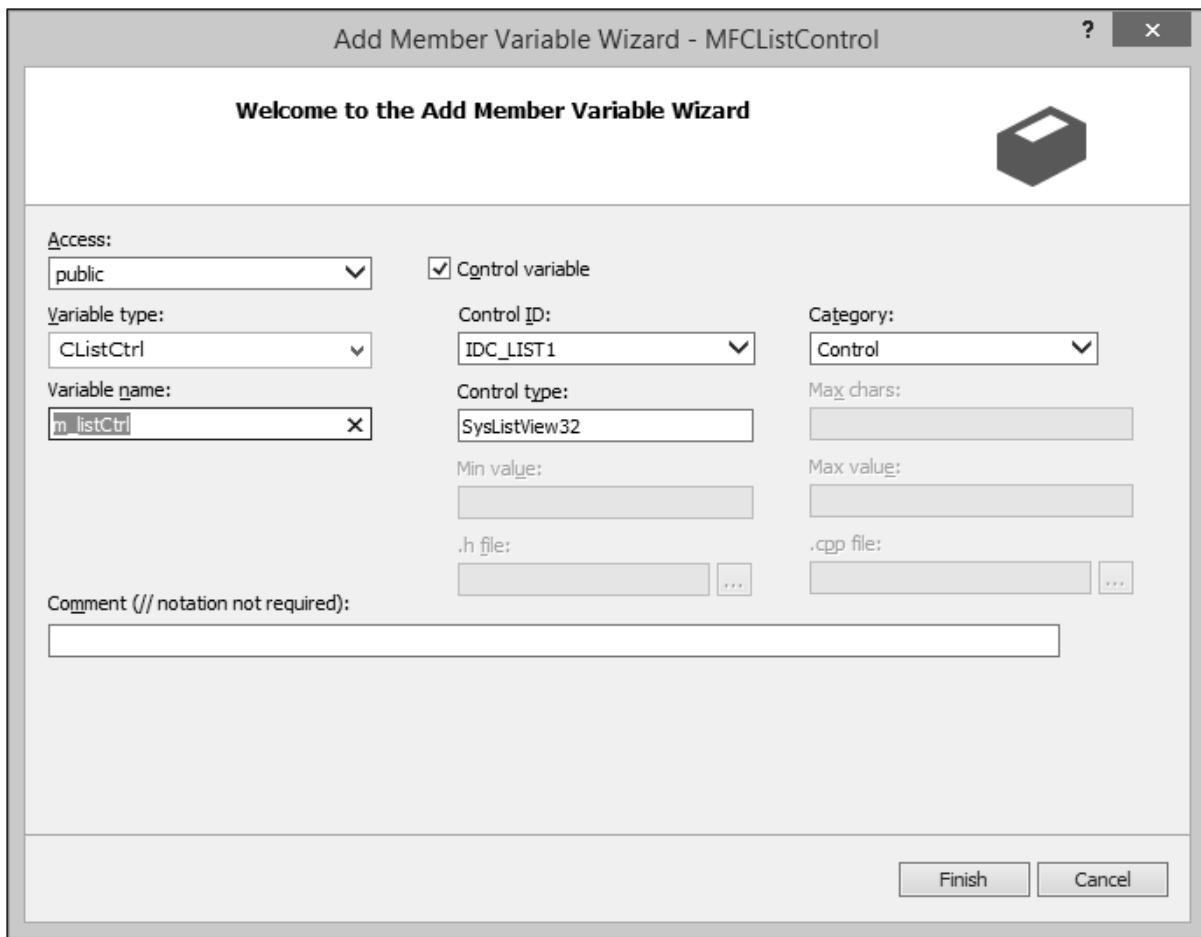
**Step 1:** Delete the TODO line and drag one List Control.

**Step 2:** In the Properties Window, you will see the different options in View dropdown list.



**Step 3:** Select the Report from the View field.

**Step 4:** Add control variable m\_listCtrl for List Control.



### Step 5: Initialize the List Control in OnInitDialog()

```
BOOL CMFCLISTCONTROLDLG::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    // Ask Mfc to create/insert a column
    m_listCtrl.InsertColumn(
        0,                      // Rank/order of item
        L"ID",                  // Caption for this header
        LVCFMT_LEFT,             // Relative position of items under header
        100);                   // Width of items under header
}
```

```
m_listCtrl.InsertColumn(1, L"Name", LVCFMT_CENTER, 80);
m_listCtrl.InsertColumn(2, L"Age", LVCFMT_LEFT, 100);
m_listCtrl.InsertColumn(3, L"Address", LVCFMT_LEFT, 80);

int nItem;

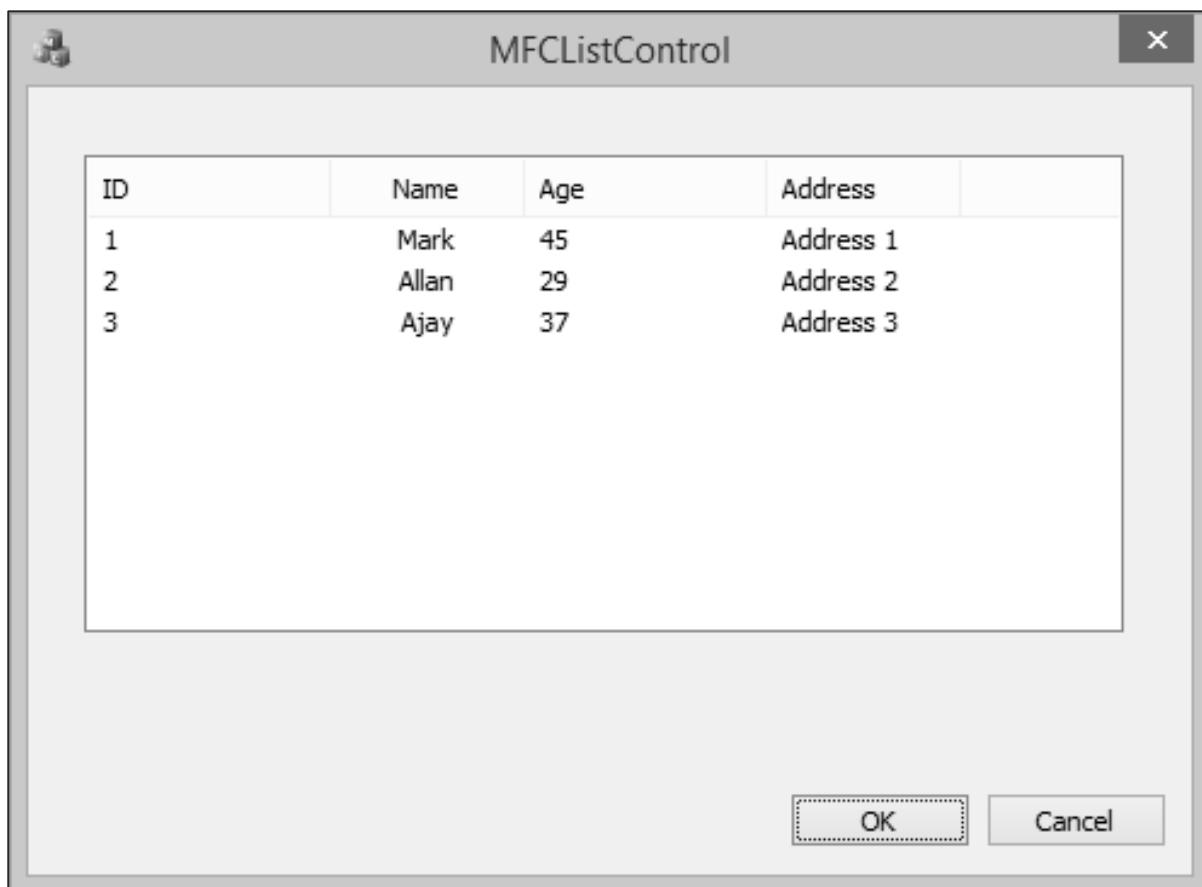
nItem = m_listCtrl.InsertItem(0, L"1");
m_listCtrl.SetItemText(nItem, 1, L"Mark");
m_listCtrl.SetItemText(nItem, 2, L"45");
m_listCtrl.SetItemText(nItem, 3, L"Address 1");

nItem = m_listCtrl.InsertItem(0, L"2");
m_listCtrl.SetItemText(nItem, 1, L"Allan");
m_listCtrl.SetItemText(nItem, 2, L"29");
m_listCtrl.SetItemText(nItem, 3, L"Address 2");

nItem = m_listCtrl.InsertItem(0, L"3");
m_listCtrl.SetItemText(nItem, 1, L"Ajay");
m_listCtrl.SetItemText(nItem, 2, L"37");
m_listCtrl.SetItemText(nItem, 3, L"Address 3");

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 6:** When the above code is compiled and executed, you will see the following output.



# 12. MFC - Messages and Events

An application is made of various objects. Most of the time, more than one application is running on the computer and the operating system is constantly asked to perform some assignments. Because there can be so many requests presented unpredictably, the operating system leaves it up to the objects to specify what they want, when they want it, and what behavior or result they expect.

## Overview

---

- The Microsoft Windows operating system cannot predict what kinds of requests one object would need to be taken care of and what type of assignment another object would need.
- To manage all these assignments and requests, the objects send messages.
- Each object has the responsibility to decide what message to send and when.
- In order to send a message, a control must create an event.
- To make a distinction between the two, a message's name usually starts with WM\_ which stands for Window Message.
- The name of an event usually starts with On which indicates an action.
- The event is the action of sending the message.

## Map of Messages

---

Since Windows is a message-oriented operating system, a large portion of programming for the Windows environment involves message handling. Each time an event such as a keystroke or mouse click occurs, a message is sent to the application, which must then handle the event.

- For the compiler to manage messages, they should be included in the class definition.
- The **DECLARE\_MESSAGE\_MAP** macro should be provided at the end of the class definition as shown in the following code.

```
class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    DECLARE_MESSAGE_MAP()
};
```

- The actual messages should be listed just above the DECLARE\_MESSAGE\_MAP line.
- To implement the messages, you need to create a table of messages that your program is using.
- This table uses two delimiting macros;
- Its starts with a **BEGIN\_MESSAGE\_MAP** and ends with an **END\_MESSAGE\_MAP** macros.
- The BEGIN\_MESSAGE\_MAP macro takes two arguments, the name of your class and the MFC class you derived your class from as shown in the following code.

```
#include <afxwin.h>

class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    DECLARE_MESSAGE_MAP()
};

CMainFrame::CMainFrame()
{
    // Create the window's frame
    Create(NULL, L"MFC Messages Demo", WS_OVERLAPPEDWINDOW,
        CRect(120, 100, 700, 480), NULL);
}

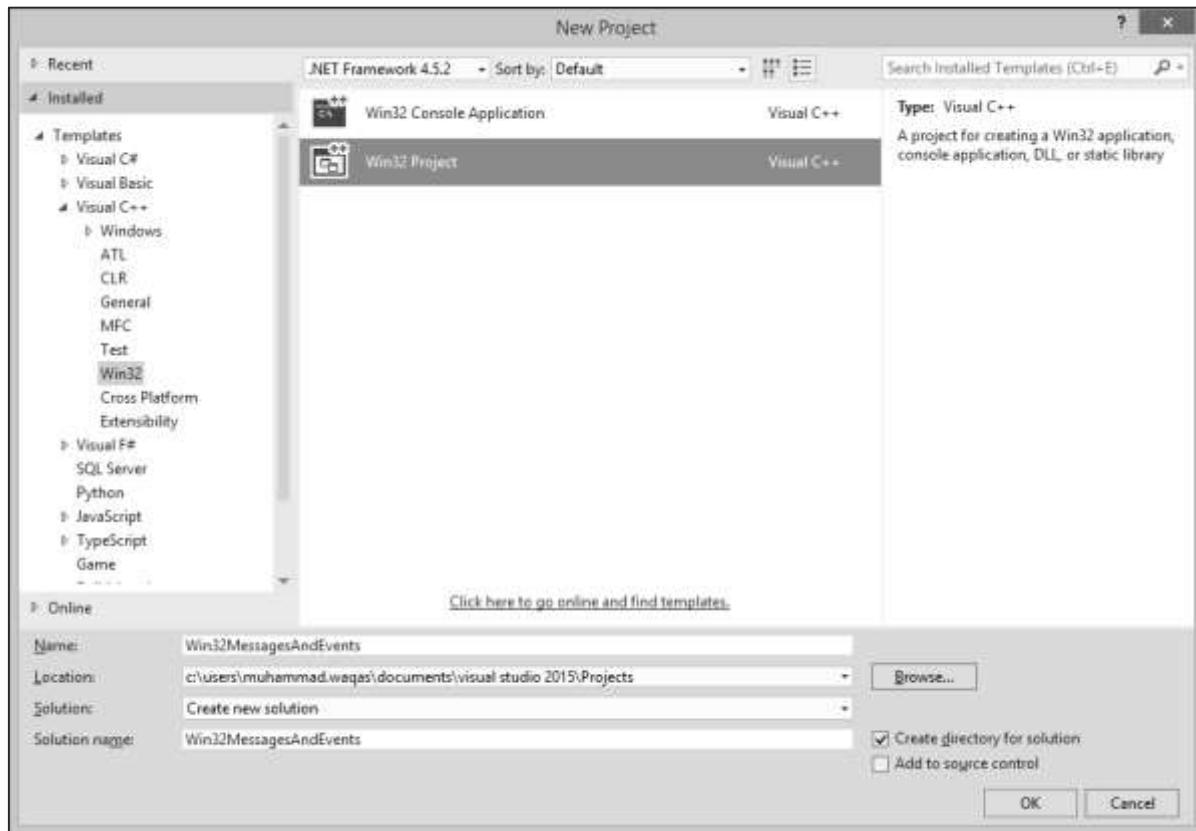
class CMessagesApp : public CWinApp
{
public:
    BOOL InitInstance();
};

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()

BOOL CMessagesApp::InitInstance()
{
    m_pMainWnd = new CMainFrame;
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}

CMessagesApp theApp;
```

Let us look into a simple example by creating a new Win32 project.



**Step 1:** To create an MFC project, right-click on the project and select Properties.

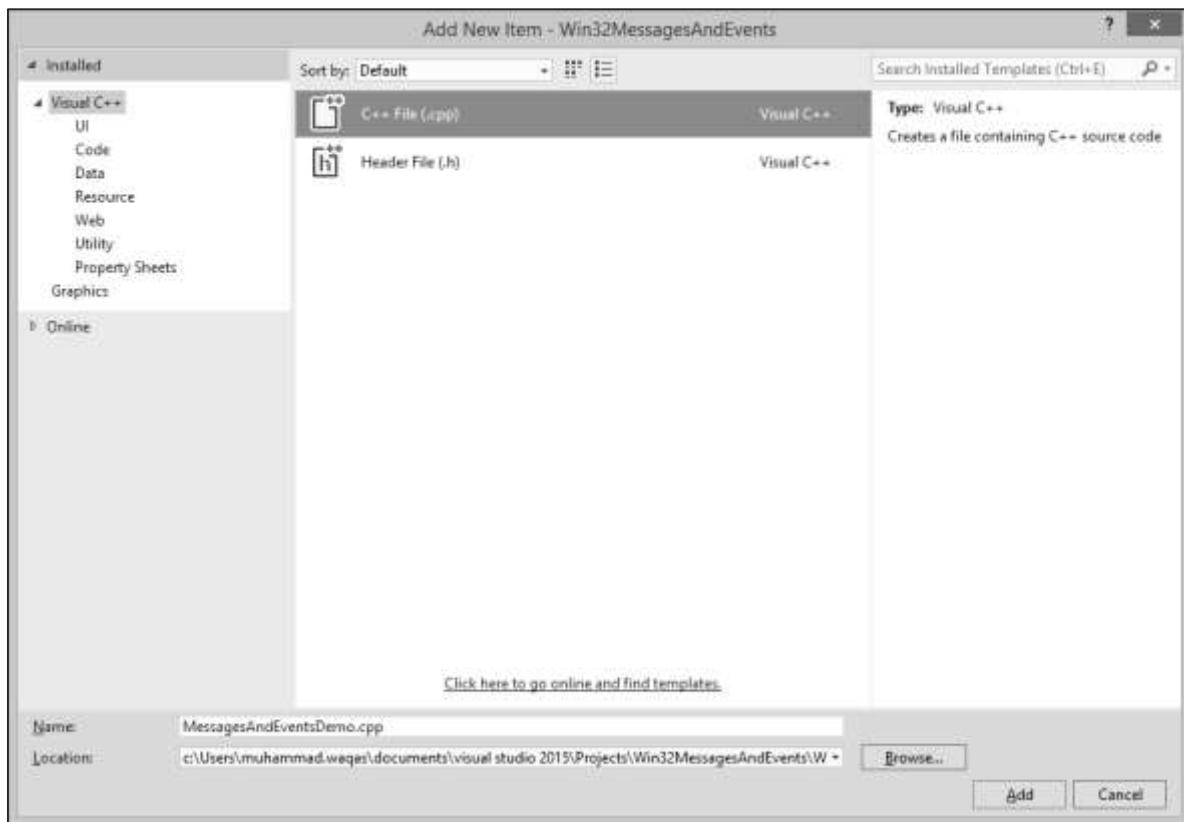
**Step 2:** In the left section, click Configuration Properties -> General.

**Step 3:** Select the 'Use MFC in Shared DLL' option in Project Defaults section and click OK.

**Step 4:** We need to add a new source file.

**Step 5:** Right-click on your Project and select Add -> New Item.

**Step 6:** In the Templates section, click C++ File (.cpp).



**Step 7:** Click Add to Continue.

**Step 8:** Now, add the following code in the \*.cpp file.

```
#include <afxwin.h>

class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    DECLARE_MESSAGE_MAP()
};

CMainFrame::CMainFrame()
{
    // Create the window's frame
    Create(NULL, L"MFC Messages Demo", WS_OVERLAPPEDWINDOW,
        CRect(120, 100, 700, 480), NULL);
}

class CMessagesApp : public CWinApp
{
public:
```

```

    BOOL InitInstance();
};

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
END_MESSAGE_MAP()
BOOL CMessagesApp::InitInstance()
{
    m_pMainWnd = new CMainFrame;
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
CMessagesApp theApp;

```

## Windows Messages

There are different types of Windows messages like creating a window, showing a window etc. Here are some of the commonly used windows messages.

Message	Map entry	Description
<b>WM_ACTIVATE</b>	ON_WM_ACTIVATE()	The framework calls this member function when a CWnd object is being activated or deactivated.
<b>WM_ACTIVATEAPP</b>	ON_WM_ACTIVATEAPP()	The framework calls this member function to all top-level windows of the task being activated and for all top-level windows of the task being deactivated.
<b>WM_APPCOMMAND</b>	ON_WM_APPCOMMAND()	The framework calls this member function when the user generates an application command event
<b>WM_CANCELMODE</b>	ON_WM_CANCELMODE()	The framework calls this member function to inform CWnd to cancel any internal mode.
<b>WM_CHILDACTIVATE</b>	ON_WM_CHILDACTIVATE()	If the CWnd object is a multiple document interface (MDI) child window, OnChildActivate is called by the framework when the user clicks the window's title bar or when the window is activated, moved, or sized.
<b>WM_CLIPBOARDUPDATE</b>	ON_WM_CLIPBOARDUPDATE()	The framework calls this member function when the contents of the clipboard have changed.
<b>WM_CLOSE</b>	ON_WM_CLOSE()	The framework calls this member function as a signal that the CWnd or an application is to terminate.
<b>WM_CONTEXTMENU</b>	ON_WM_CONTEXTMENU()	Called by the framework when the user has clicked the right mouse button (right-clicked) in the window.

<b>WM_COPYDATA</b>	ON_WM_COPYDATA()	This member function is called by the framework to copy data from one application to another.
<b>WM_CREATE</b>	ON_WM_CREATE()	The framework calls this member function when an application requests that the Windows window be created by calling the Create or CreateEx member function.
<b>WM_CTLCOLOR</b>	ON_WM_CTLCOLOR()	The framework calls this member function when a child control is about to be drawn.
<b>WM_DELETEITEM</b>	ON_WM_DELETEITEM()	The framework calls this member function to inform the owner of an owner-draw list box or combo box that the list box or combo box is destroyed or that items have been removed
<b>WM_DESTROY</b>	ON_WM_DESTROY()	The framework calls this member function to inform the CWnd object that it is being destroyed.
<b>WM_DRAWITEM</b>	ON_WM_DRAWITEM()	The framework calls this member function for the owner of an owner-draw button control, combo-box control, list-box control, or menu when a visual aspect of the control or menu has changed.
<b>WM_DROPFILES</b>	ON_WM_DROPFILES()	The framework calls this member function when the user releases the left mouse button over a window that has registered itself as the recipient of dropped files.
<b>WM_ENABLE</b>	ON_WM_ENABLE()	The framework calls this member function when an application changes the enabled state of the CWnd object. Syntax
<b>WM_HELPINFO</b>	ON_WM_HELPINFO()	Handles F1 Help within the application (using the current context).
<b>WM_HOTKEY</b>	ON_WM_HOTKEY()	The framework calls this member function when the user presses a system-wide hot key.
<b>WM_HSCROLL</b>	ON_WM_HSCROLL()	The framework calls this member function when the user clicks a window's horizontal scroll bar.
<b>WM_KEYDOWN</b>	ON_WM_KEYDOWN()	The framework calls this member function when a nonsystem key is pressed.
<b>WM_KEYUP</b>	ON_WM_KEYUP()	The framework calls this member function when a nonsystem key is released.
<b>WM_KILLFOCUS</b>	ON_WM_KILLFOCUS()	The framework calls this member function immediately before losing the input focus.
<b>WM_LBUTTONDOWNDBLCLK</b>	ON_WM_LBUTTONDOWNDBLCLK()	The framework calls this member function when the user double-clicks the left mouse button.
<b>WM_LBUTTONDOWNOWN</b>	ON_WM_LBUTTONDOWNOWN()	The framework calls this member function when the user presses the left mouse button.
<b>WM_LBUTTONUP</b>	ON_WM_LBUTTONUP()	The framework calls this member function when the user releases the left mouse button.

<b>WM_MBUTTONDOWN BLCLK</b>	ON_WM_MBUTTONDOWNDBLCLK()	The framework calls this member function when the user double-clicks the middle mouse button.
<b>WM_MBUTTONDOWN OWN</b>	ON_WM_MBUTTONDOWNDOWN()	The framework calls this member function when the user presses the middle mouse button.
<b>WM_MBUTTONDOWN UP</b>	ON_WM_MBUTTONUP()	The framework calls this member function when the user releases the middle mouse button.
<b>WM_MENUSELECT</b>	ON_WM_MENUSELECT()	If the CWnd object is associated with a menu, OnMenuSelect is called by the framework when the user selects a menu item.
<b>WM_MOUSEACTIVATE</b>	ON_WM_MOUSEACTIVATE()	The framework calls this member function when the cursor is in an inactive window and the user presses a mouse button.
<b>WM_MOUSEHOVER</b>	ON_WM_MOUSEHOVER()	The framework calls this member function when the cursor hovers over the client area of the window for the period of time specified in a prior call to TrackMouseEvent.
<b>WM_MOUSEHWHEEL</b>	ON_WM_MOUSEHWHEEL()	The framework calls this member when the current window is composed by the Desktop Window Manager (DWM), and that window is maximized.
<b>WM_MOUSELEAVE</b>	ON_WM_MOUSELEAVE()	The framework calls this member function when the cursor leaves the client area of the window specified in a prior call to TrackMouseEvent.
<b>WM_MOUSEMOVE</b>	ON_WM_MOUSEMOVE()	The framework calls this member function when the mouse cursor moves.
<b>WM_MOVE</b>	ON_WM_MOVE()	The framework calls this member function after the CWnd object has been moved.
<b>WM_PAINT</b>	ON_WM_PAINT()	The framework calls this member function when Windows or an application makes a request to repaint a portion of an application's window.
<b>WM_SETFOCUS( )</b>	ON_WM_SETFOCUS()	The framework calls this member function after gaining the input focus.
<b>WM_SIZE( )</b>	ON_WM_SIZE()	The framework calls this member function after the window's size has changed.
<b>WM_TIMER</b>	ON_WM_TIMER()	The framework calls this member function after each interval specified in the SetTimer member function used to install a timer.
<b>WM_VSCROLL</b>	ON_WM_VSCROLL()	The framework calls this member function when the user clicks the window's vertical scroll bar.
<b>WM_WINDOWPOSCHANGED</b>	ON_WM_WINDOWPOSCHANGED()	The framework calls this member function when the size, position, or Z-order has changed as a result of a call to the SetWindowPos member function or another window-management function.

Let us look into a simple example of window creation.

**WM\_CREATE:** When an object, called a window, is created, the frame that creates the objects sends a message identified as **ON\_WM\_CREATE**.

**Step 1:** To create ON\_WM\_CREATE, add `afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);` before the `DECLARE_MESSAGE_MAP()` as shown below.

```
class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};
```

**Step 2:** Add the `ON_WM_CREATE()` after the `BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)` and before `END_MESSAGE_MAP()`

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()
```

**Step 3:** Here is the Implementation of `OnCreate()`

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // Call the base class to create the window
    if (CFrameWnd::OnCreate(lpCreateStruct) == 0)
    {
        // If the window was successfully created, let the user know
        MessageBox(L"The window has been created!!!");
        // Since the window was successfully created, return 0
        return 0;
    }
    // Otherwise, return -1
    return -1;
}
```

**Step 4:** Now your \*.cpp file will look like as shown in the following code.

```
#include <afxwin.h>

class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};

CMainFrame::CMainFrame()
{
    // Create the window's frame
    Create(NULL, L"MFC Messages Demo", WS_OVERLAPPEDWINDOW,
        CRect(120, 100, 700, 480), NULL);
}

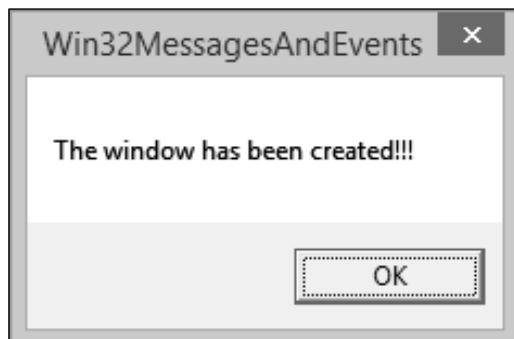
class CMessagesApp : public CWinApp
{
public:
    BOOL InitInstance();
};

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()

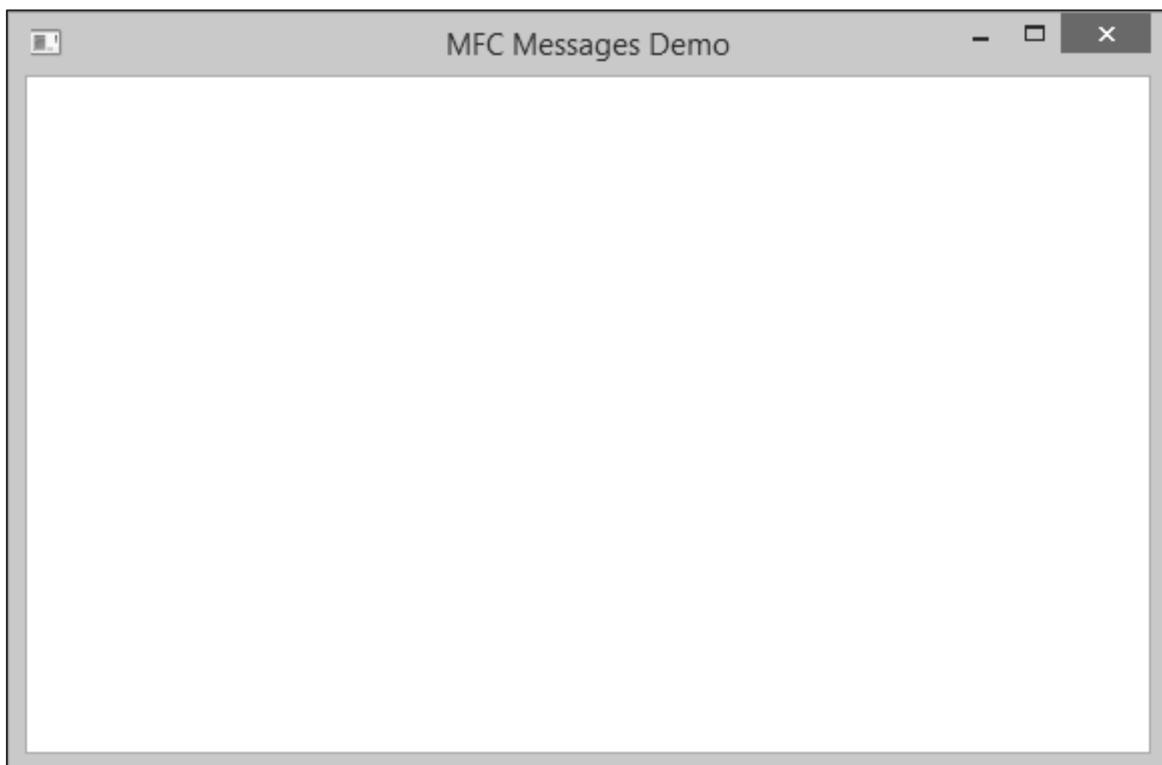
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // Call the base class to create the window
    if (CFrameWnd::OnCreate(lpCreateStruct) == 0)
    {
        // If the window was successfully created, let the user know
        MessageBox(L"The window has been created!!!");
        // Since the window was successfully created, return 0
        return 0;
    }
    // Otherwise, return -1
    return -1;
}
```

```
BOOL CMessagesApp::InitInstance()
{
    m_pMainWnd = new CMainFrame;
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
CMessagesApp theApp;
```

**Step 5:** When the above code is compiled and executed, you will see the following output.



**Step 6:** When you click OK, it will display the main window.



## Command Messages

One of the main features of a graphical application is to present Windows controls and resources that allow the user to interact with the machine. Examples of controls that we will learn are buttons, list boxes, combo boxes, etc.

One type of resource we introduced in the previous lesson is the menu. Such controls and resources can initiate their own messages when the user clicks them. A message that emanates from a Windows control or a resource is called a command message.

Let us look into a simple example of Command messages.

To provide your application the ability to create a new document, the CWinApp class provides the OnFileNew() method.

```
afx_msg void OnFileNew();

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_COMMAND(ID_FILE_NEW, CMainFrame::OnFileNew)
END_MESSAGE_MAP()
```

Here is the method definition:

```
void CMainFrame::OnFileNew()
{
    // Create New file
}
```

## Keyboard Messages

A **keyboard** is a hardware object attached to the computer. By default, it is used to enter recognizable symbols, letters, and other characters on a control. Each key on the keyboard displays a symbol, a letter, or a combination of those, to give an indication of what the key could be used for. The user typically presses a key, which sends a signal to a program.

Each key has a code that the operating system can recognize. This code is known as the **virtual key code**. The virtual key codes are as follows:

Constant/value	Description
<b>VK_LBUTTON</b>	Left mouse button
<b>VK_RBUTTON</b>	Right mouse button
<b>VK_CANCEL</b>	Control-break processing
<b>VK_MBUTTON</b>	Middle mouse button (three-button mouse)
<b>VK_BACK</b>	BACKSPACE key
<b>VK_TAB</b>	TAB key
<b>VK_CLEAR</b>	CLEAR key

<b>VK_RETURN</b>	ENTER key
<b>VK_SHIFT</b>	SHIFT key
<b>VK_CONTROL</b>	CTRL key
<b>VK_MENU</b>	ALT key
<b>VK_PAUSE</b>	PAUSE key
<b>VK_CAPITAL</b>	CAPS LOCK key
<b>VK_ESCAPE</b>	ESC key
<b>VK_SPACE</b>	SPACEBAR
<b>VK_PRIOR</b>	PAGE UP key
<b>VK_NEXT</b>	PAGE DOWN key
<b>VK_END</b>	END key
<b>VK_HOME</b>	HOME key
<b>VK_LEFT</b>	LEFT ARROW key
<b>VK_UP</b>	UP ARROW key
<b>VK_RIGHT</b>	RIGHT ARROW key
<b>VK_DOWN</b>	DOWN ARROW key
<b>VK_SELECT</b>	SELECT key
<b>VK_PRINT</b>	PRINT key
<b>VK_EXECUTE</b>	EXECUTE key
<b>VK_SNAPSHOT</b>	PRINT SCREEN key
<b>VK_INSERT</b>	INS key
<b>VK_DELETE</b>	DEL key
<b>VK_NUMPAD0</b>	Numeric keypad 0 key
<b>VK_NUMPAD1</b>	Numeric keypad 1 key
<b>VK_NUMPAD2</b>	Numeric keypad 2 key
<b>VK_NUMPAD3</b>	Numeric keypad 3 key
<b>VK_NUMPAD4</b>	Numeric keypad 4 key
<b>VK_NUMPAD5</b>	Numeric keypad 5 key
<b>VK_NUMPAD6</b>	Numeric keypad 6 key
<b>VK_NUMPAD7</b>	Numeric keypad 7 key
<b>VK_NUMPAD8</b>	Numeric keypad 8 key
<b>VK_NUMPAD9</b>	Numeric keypad 9 key
<b>VK_MULTIPLY</b>	Multiply key
<b>VK_ADD</b>	Add key
<b>VK_SEPARATOR</b>	Separator key
<b>VK_SUBTRACT</b>	Subtract key
<b>VK_DECIMAL</b>	Decimal key
<b>VK_DIVIDE</b>	Divide key
<b>VK_F1</b>	F1 key
<b>VK_F2</b>	F2 key
<b>VK_F3</b>	F3 key
<b>VK_F4</b>	F4 key
<b>VK_F5</b>	F5 key
<b>VK_F6</b>	F6 key
<b>VK_F7</b>	F7 key
<b>VK_F8</b>	F8 key
<b>VK_F9</b>	F9 key
<b>VK_F10</b>	F10 key
<b>VK_F11</b>	F11 key
<b>VK_F12</b>	F12 key
<b>VK_NUMLOCK</b>	NUM LOCK key
<b>VK_SCROLL</b>	SCROLL LOCK key

<b>VK_LSHIFT</b>	Left SHIFT key
<b>VK_RSHIFT</b>	Right SHIFT key
<b>VK_LCONTROL</b>	Left CONTROL key
<b>VK_RCONTROL</b>	Right CONTROL key

Pressing a key causes a [WM\\_KEYDOWN](#) or [WM\\_SYSKEYDOWN](#) message to be placed in the thread message. This can be defined as follows:

```
afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
```

Let us look into a simple example.

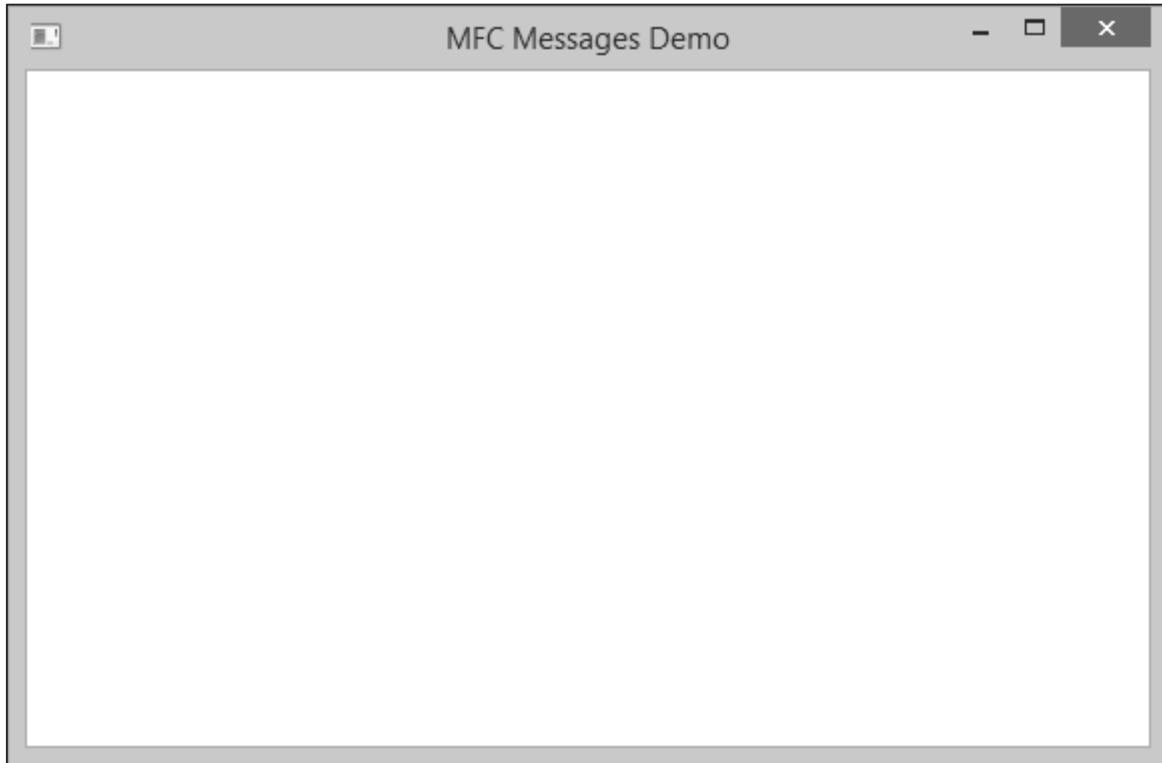
**Step 1:** Here is the message.

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_KEYDOWN()
END_MESSAGE_MAP()
```

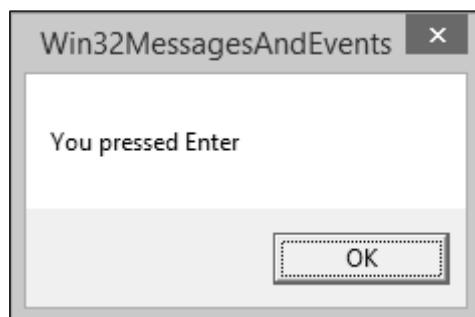
**Step 2:** Here is the implementation of OnKeyDown().

```
void CMainFrame::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    switch (nChar)
    {
        case VK_RETURN:
            MessageBox(L"You pressed Enter");
            break;
        case VK_F1:
            MessageBox(L"Help is not available at the moment");
            break;
        case VK_DELETE:
            MessageBox(L"Can't Delete This");
            break;
        default:
            MessageBox(L"Whatever");
    }
}
```

**Step 3:** When the above code is compiled and executed, you will see the following output.



**Step 4:** When you press Enter, it will display the following message.



## Mouse Messages

---

The mouse is another object that is attached to the computer allowing the user to interact with the machine.

- If the left mouse button was pressed, an ON\_WM\_LBUTTONDOWN message is sent. The syntax of this message is:
  - `afx_msg void OnLButtonDown(UINT nFlags, CPoint point)`
- If the right mouse button was pressed, an ON\_WM\_RBUTTONDOWN message is sent. Its syntax is:
  - `afx_msg void OnRButtonDown(UINT nFlags, CPoint point)`

- Similarly If the left mouse is being released, the ON\_WM\_LBUTTONDOWN message is sent. Its syntax is:
  - afx\_msg void OnLButtonDown(UINT nFlags, CPoint point)
- If the right mouse is being released, the ON\_WM\_RBUTTONDOWN message is sent. Its syntax is:
  - afx\_msg void OnRButtonUp(UINT nFlags, CPoint point)

Let us look into a simple example.

**Step 1:** Add the following two functions in CMainFrame class definition as shown in the following code.

```
class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();
protected:
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
    DECLARE_MESSAGE_MAP()
};
```

**Step 2:** Add the following two Message Maps.

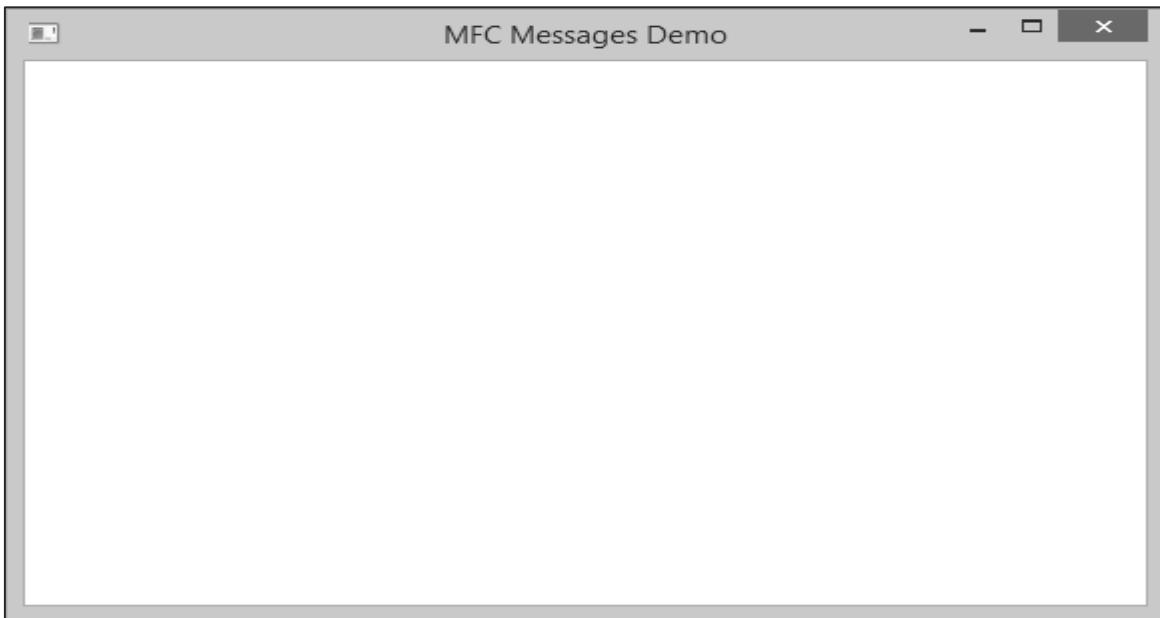
```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_KEYDOWN()
    ON_WM_LBUTTONDOWN()
    ON_WM_RBUTTONUP()
END_MESSAGE_MAP()
```

**Step 3:** Here is the functions definition.

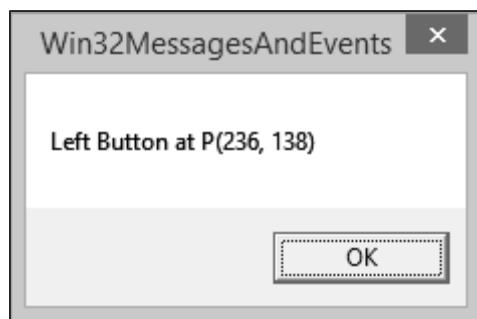
```
void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    CString MsgCoord;
    MsgCoord.Format(L"Left Button at P(%d, %d)", point.x, point.y);
    MessageBox(MsgCoord);}

void CMainFrame::OnRButtonUp(UINT nFlags, CPoint point)
{
    MessageBox(L"Right Mouse Button Up");}
```

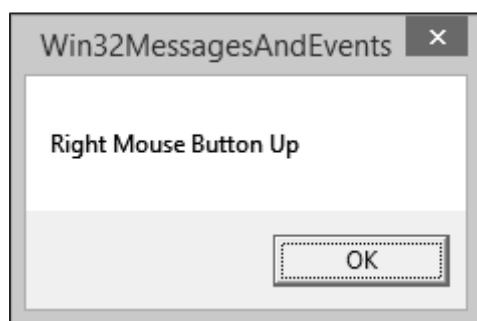
**Step 4:** When you run this application, you will see the following output.



**Step 5:** When you click OK, you will see the following message.



**Step 6:** Right-click on this window. Now, when you release the right button of the mouse, it will display the following message.



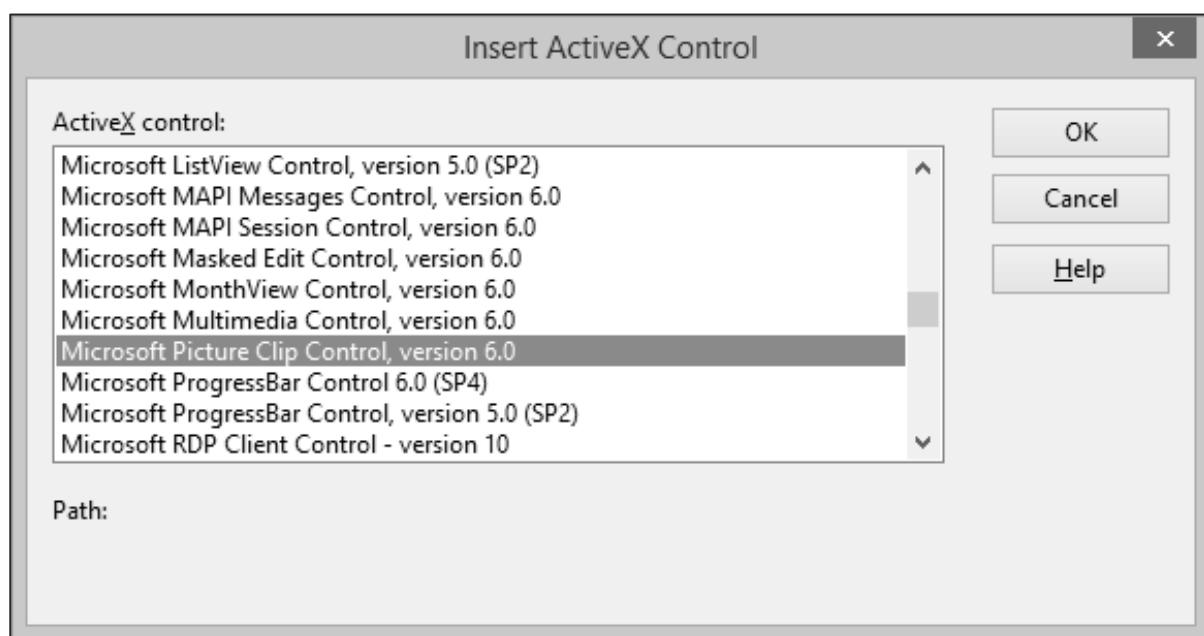
# 13. MFC - ActiveX Control

An **ActiveX control container** is a parent program that supplies the environment for an ActiveX (formerly OLE) control to run.

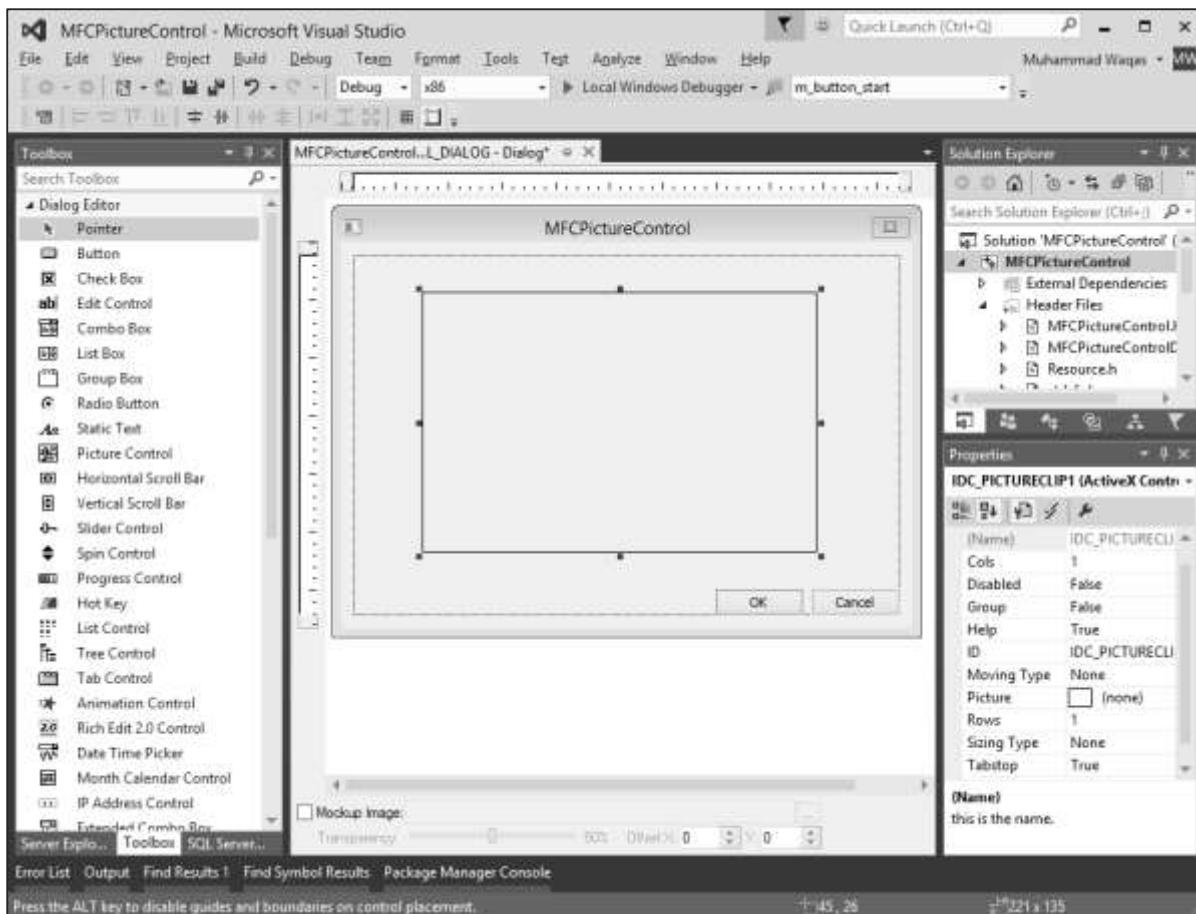
- ActiveX control is a control using Microsoft ActiveX technologies.
- ActiveX is not a programming language, but rather a set of rules for how applications should share information.
- Programmers can develop ActiveX controls in a variety of languages, including C, C++, Visual Basic, and Java.
- You can create an application capable of containing ActiveX controls with or without MFC, but it is much easier to do with MFC.

Let us look into simple example of add ActiveX controls in your MFC dialog based application.

**Step 1:** Right-click on the dialog in the designer window and select Insert ActiveX Control.



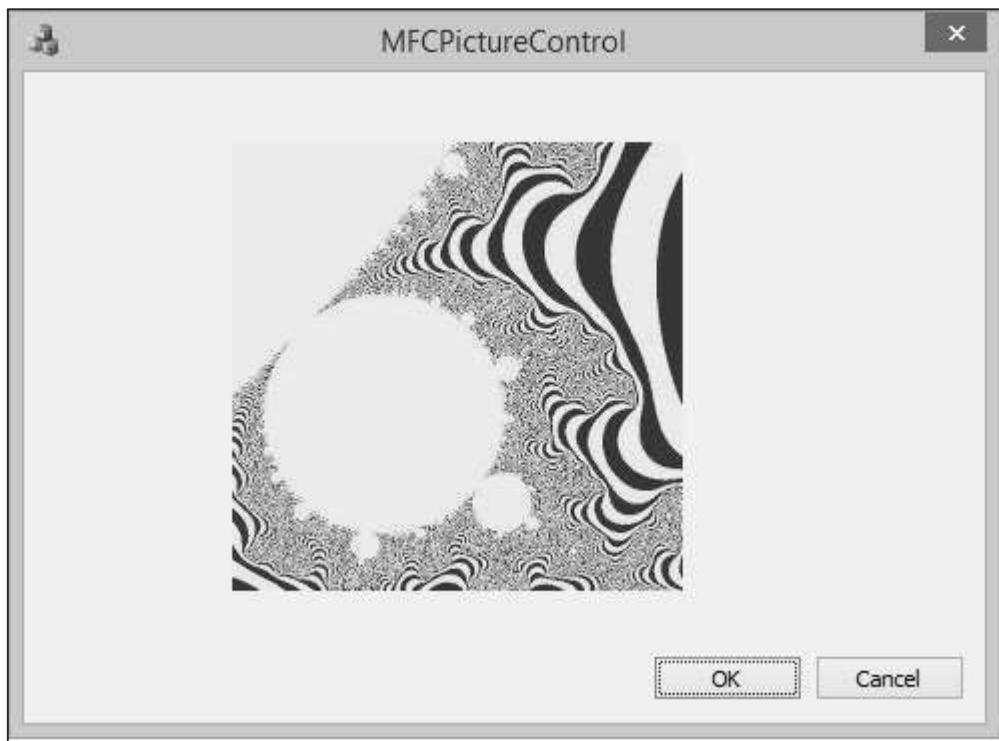
**Step 2:** Select the Microsoft Picture Clip Control and click OK.



**Step 3:** Resize the Picture control and in the Properties window, click the Picture field.

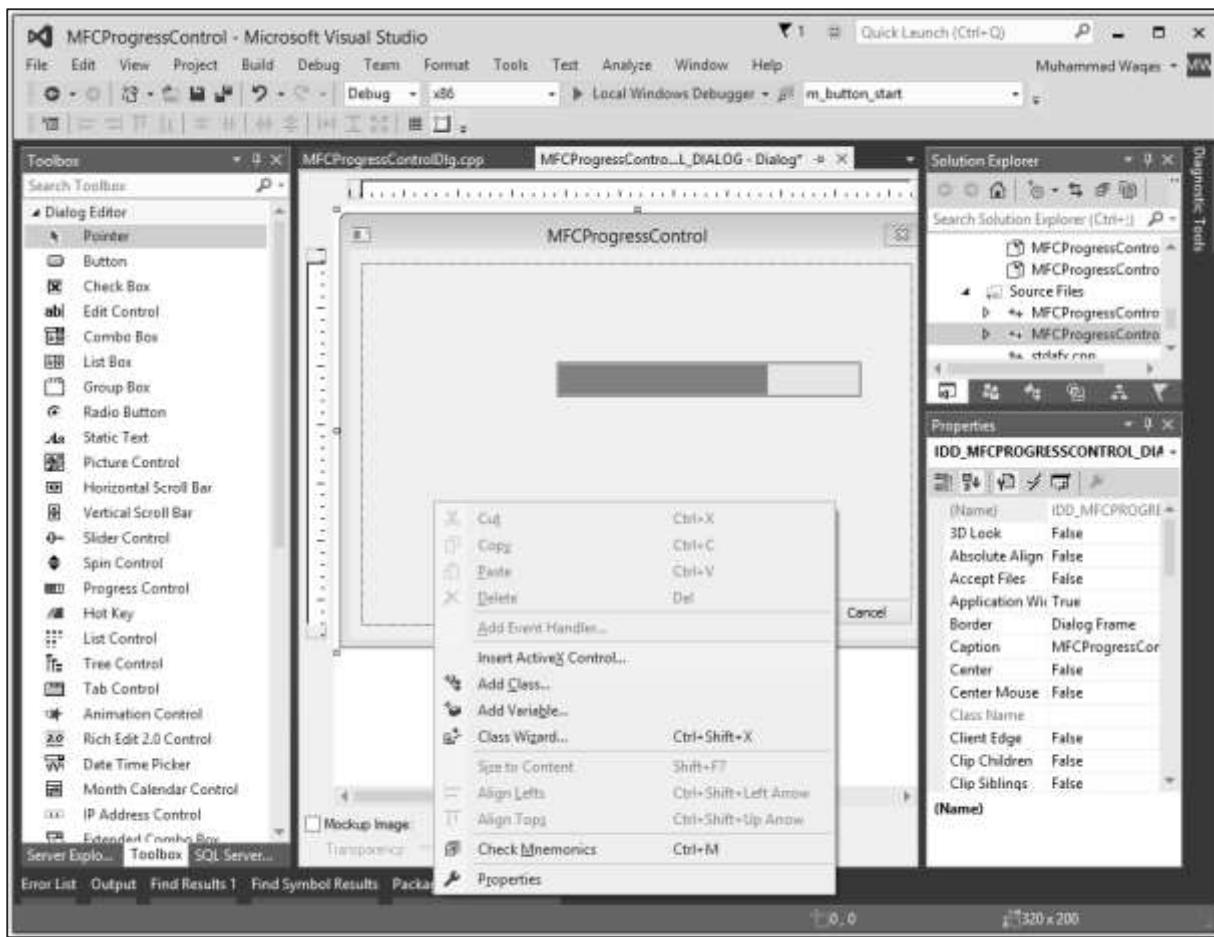
**Step 4:** Browse the folder that contains Pictures. Select any picture.

**Step 5:** When you run this application, you will see the following output.

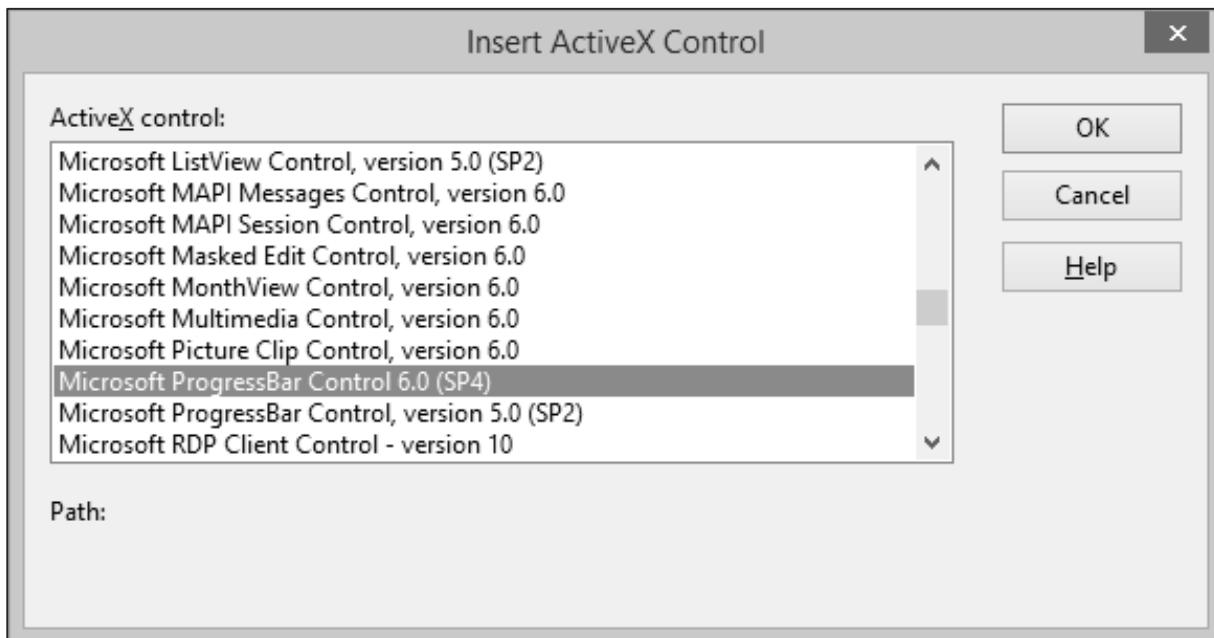


Let us have a look into another simple example.

**Step 1:** Right-click on the dialog in the designer window.



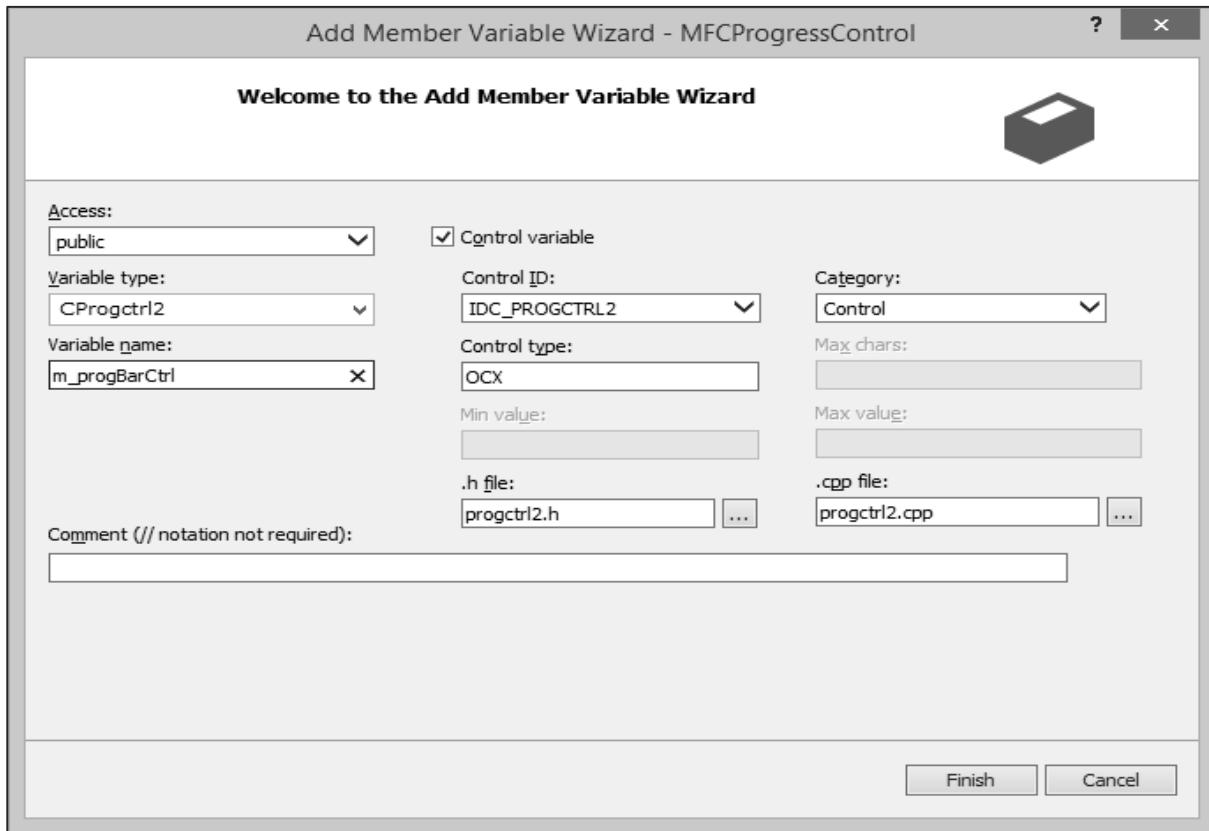
**Step 2:** Select Insert ActiveX Control.



**Step 3:** Select the Microsoft ProgressBar Control 6.0, click OK.

**Step 4:** Select the progress bar and set its Orientation in the Properties Window to **1 – ccOrientationVertical**.

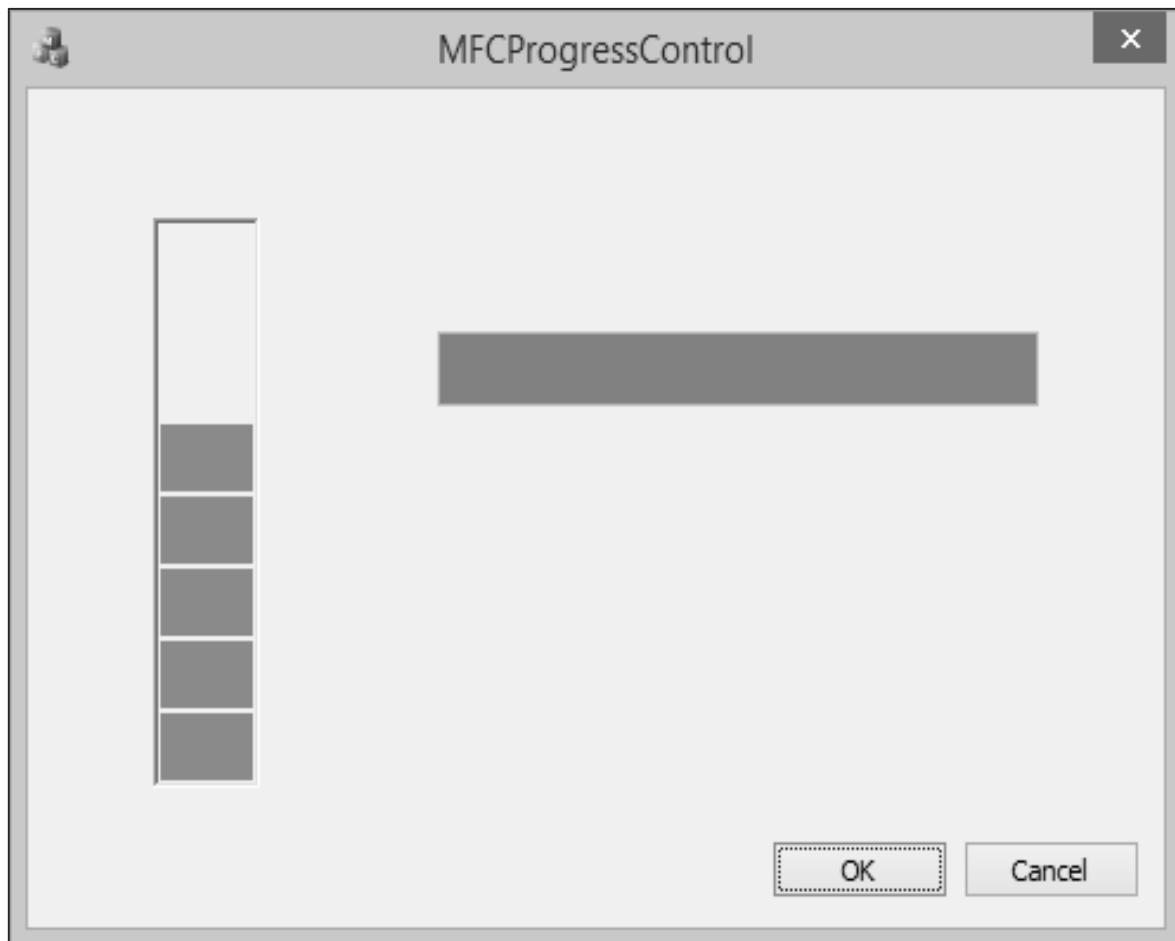
**Step 5:** Add control variable for Progress bar.



**Step 6:** Add the following code in the OnInitDialog()

```
m_progBarCtrl.SetScrollRange(0,100,TRUE);
m_progBarCtrl.put_Value(53);
```

**Step 7:** When you run this application again, you will see the progress bar in Vertical direction as well.



# 14. MFC - File System

In this chapter, we will discuss the various components of the file system.

## Drives

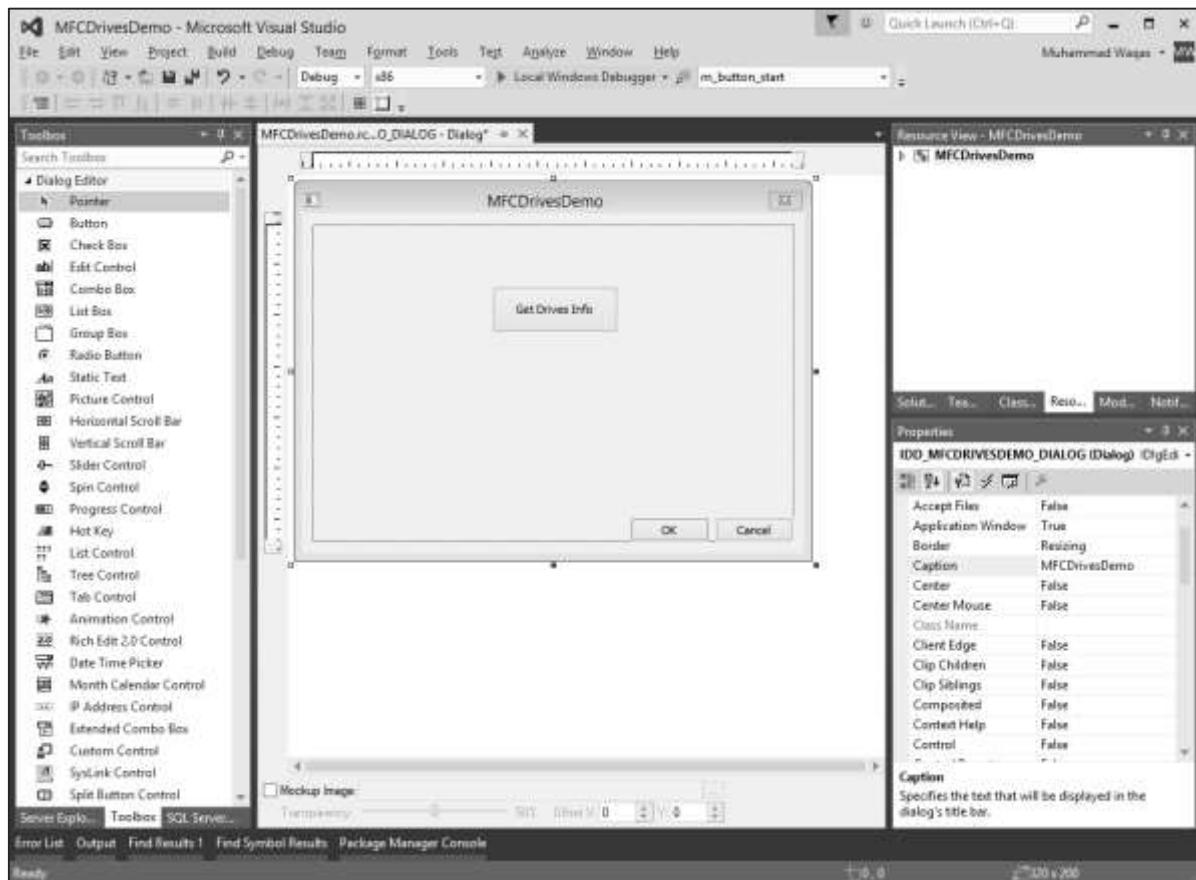
A **drive** is a physical device attached to a computer so it can store information. A logical disk, logical volume or virtual disk (VD or vdisk for short) is a virtual device that provides an area of usable storage capacity on one or more physical disk drive(s) in a computer system. A drive can be a hard disk, a CD ROM, a DVD ROM, a flash (USB) drive, a memory card, etc.

One of the primary operations you will want to perform is to get a list of drives on the computer.

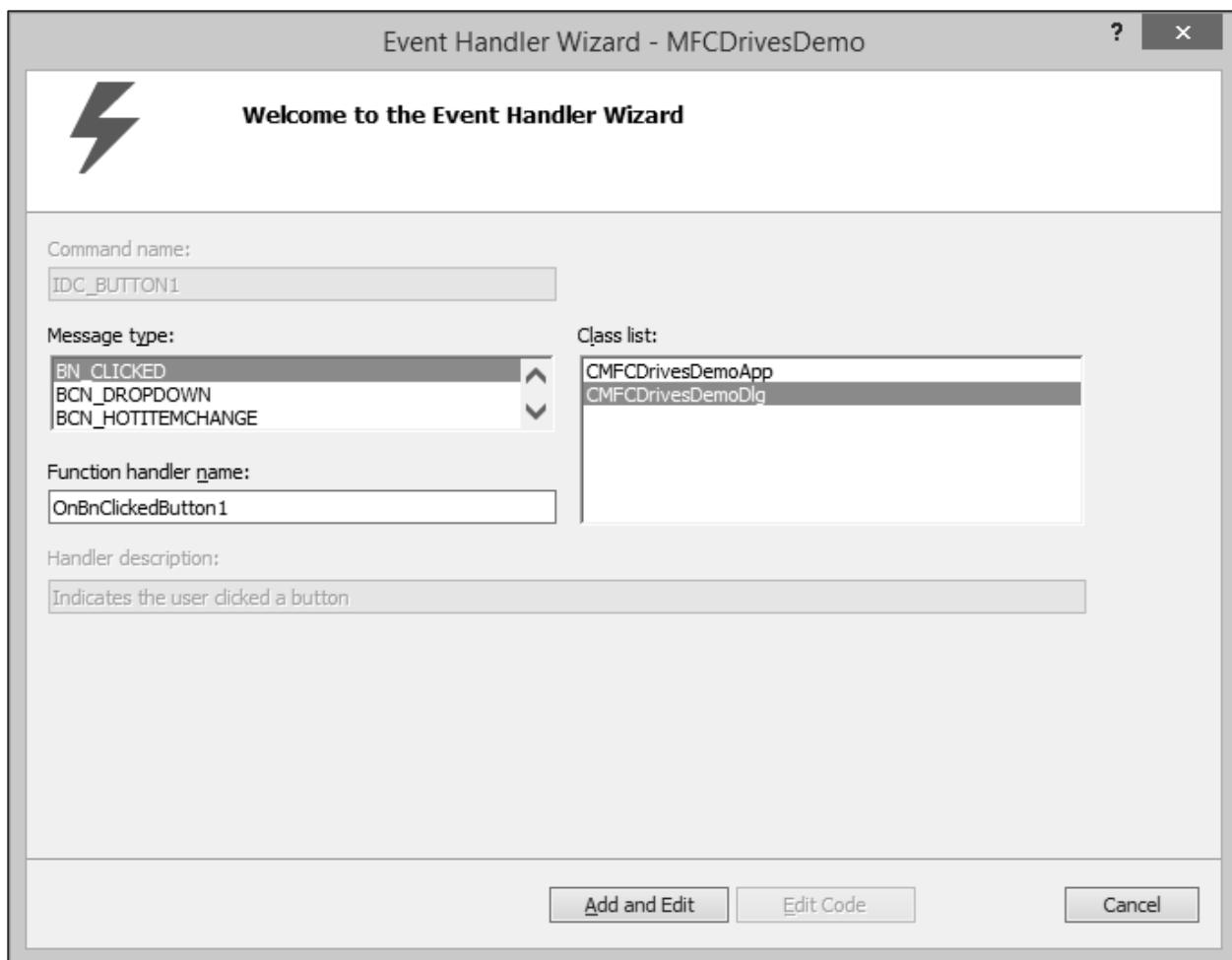
Let us look into a simple example by creating a new MFC dialog based application.

**Step 1:** Drag one button from the toolbox, change its Caption to Get Drives Info.

**Step 2:** Remove the Caption of Static control (TODO line) and change its ID to IDC\_STATIC\_TEXT.

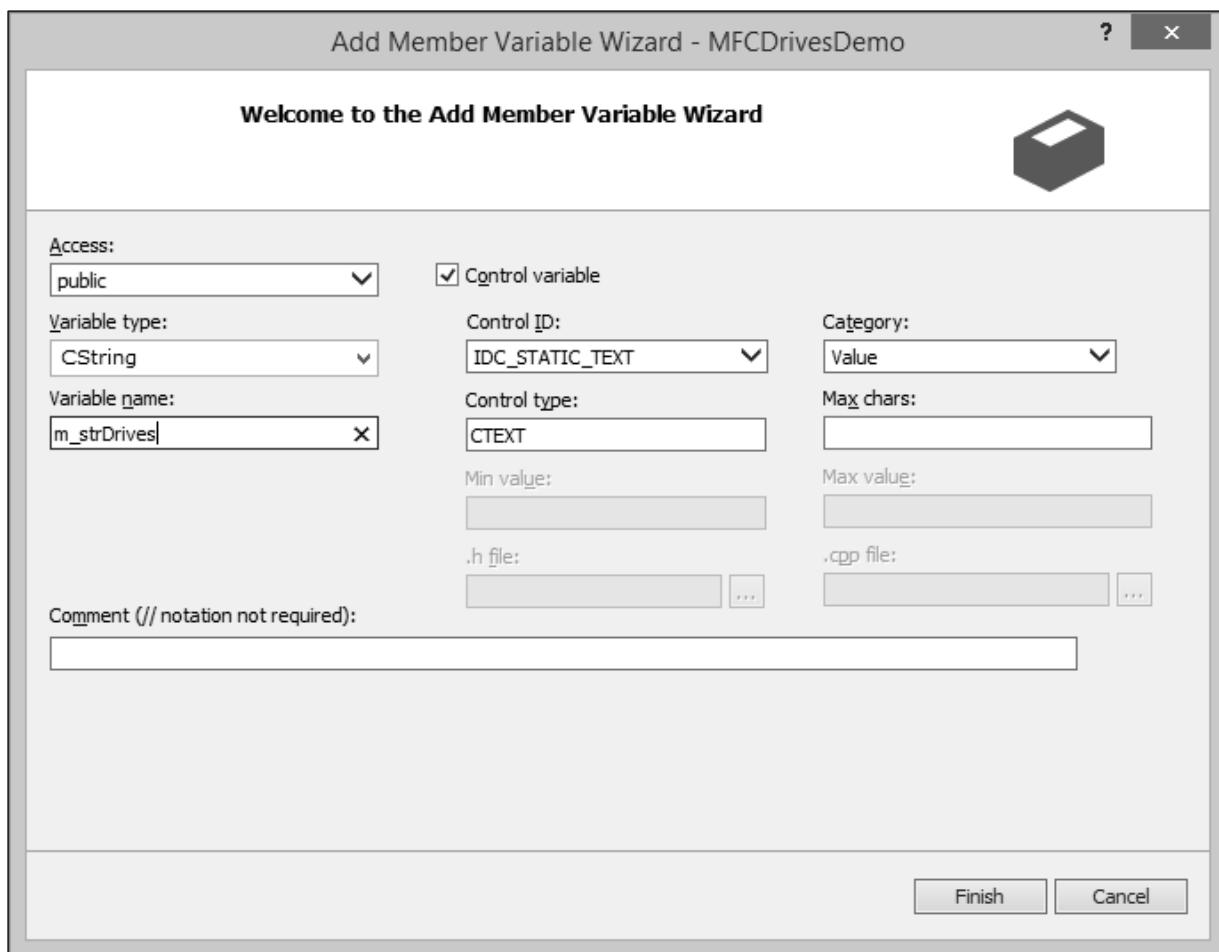


**Step 3:** Right-click on the button and select Add Event Handler.



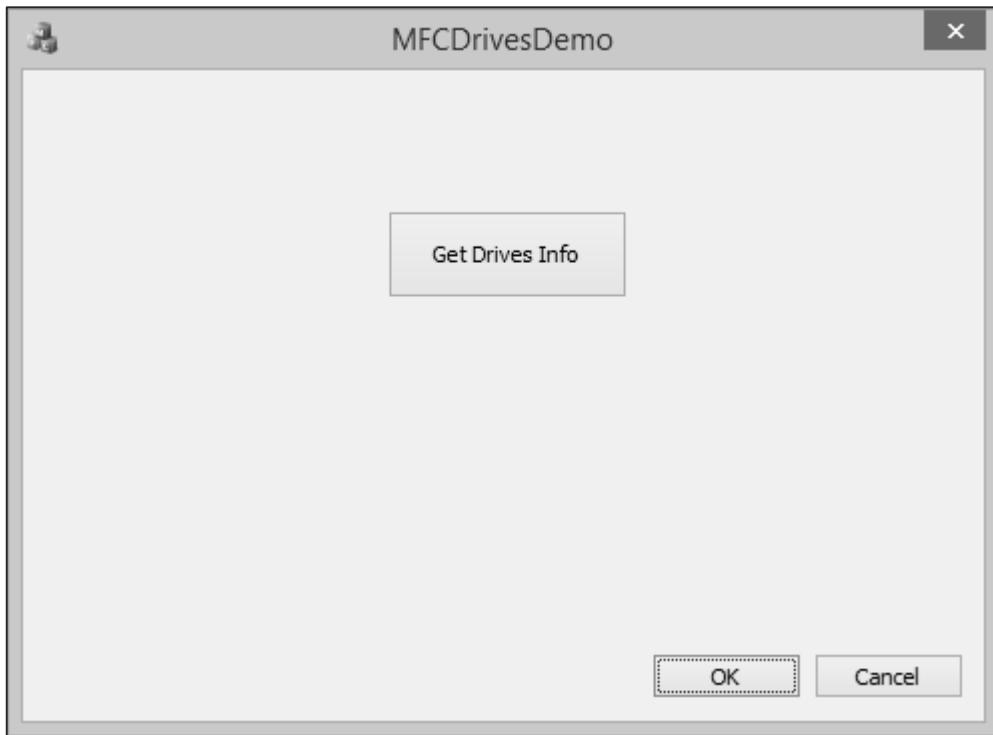
**Step 4:** Select the BN\_CLICKED message type and click the Add and Edit button.

**Step 5:** Add the value variable m\_strDrives for Static Text control.

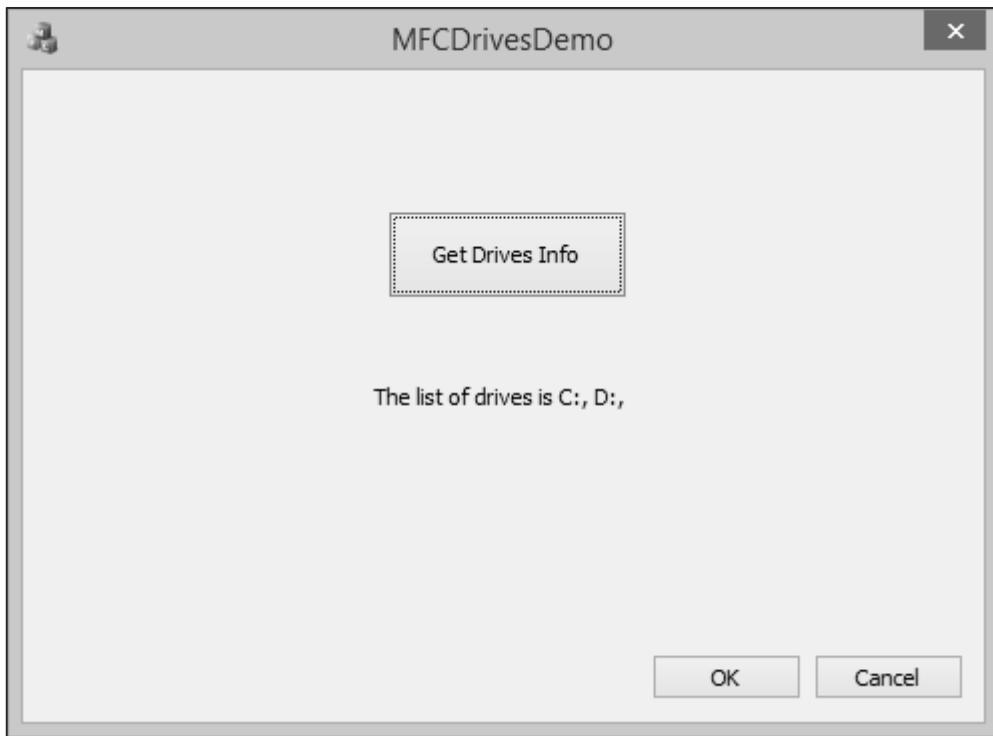


To support drives on a computer, the Win32 library provides the GetLogicalDrives() function of Microsoft Window, which will retrieve a list of all drives on the current computer.

**Step 6:** When the above code is compiled and executed, you will see the following output.



**Step 7:** When you click the button, you can see all the drives on your computer.



## Directories

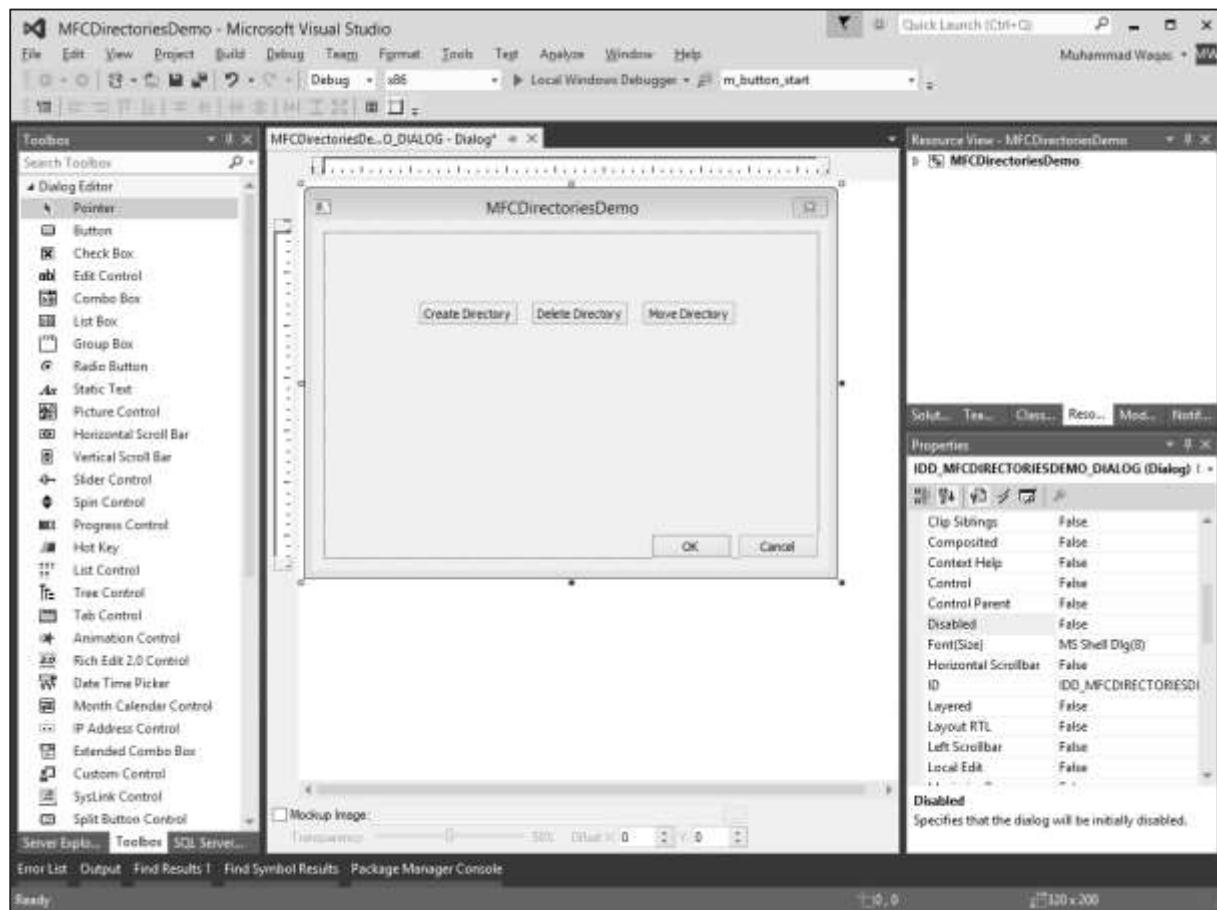
In computing, a **directory** is a file system cataloging structure which contains references to other computer files, and possibly other directories. Directory is a physical location. It can handle operations not available on a drive.

Let us look into a simple example by creating a new MFC dialog based application

**Step 1:** Drag three buttons from the toolbox. Change their Captions to Create Directory, Delete Directory and Move Directory.

**Step 2:** Change the IDs of these buttons to **IDC\_BUTTON\_CREATE**, **IDC\_BUTTON\_DELETE** and **IDC\_BUTTON\_MOVE**.

**Step 3:** Remove the TODO line.



**Step 4:** Add event handler for each button.

**Step 5:** To create a directory, you can call the `CreateDirectory()` method of the Win32 library.

**Step 6:** Here is the Create button event handler implementation in which we will create one directory and then two more sub directories.

```

void CMFCDirectoriesDemoDlg::OnBnClickedButtonCreate()
{
    // TODO: Add your control notification handler code here
    SECURITY_ATTRIBUTES saPermissions;

    saPermissions.nLength = sizeof(SECURITY_ATTRIBUTES);
    saPermissions.lpSecurityDescriptor = NULL;
    saPermissions.bInheritHandle = TRUE;

    if (CreateDirectory(L"D:\\MFCDirectoryDEMO", &saPermissions) == TRUE)
        AfxMessageBox(L"The directory was created.");
    CreateDirectory(L"D:\\MFCDirectoryDEMO\\Dir1", NULL);
    CreateDirectory(L"D:\\MFCDirectoryDEMO\\Dir2", NULL);
}

```

**Step 7:** To get rid of a directory, you can call the **RemoveDirectory()** function of the Win32 library. Here is the implementation of delete button event handler.

```

void CMFCDirectoriesDemoDlg::OnBnClickedButtonDelete()
{
    // TODO: Add your control notification handler code here
    if (RemoveDirectory(L"D:\\MFCDirectoryDEMO\\Dir1") == TRUE)
        AfxMessageBox(L"The directory has been deleted");
}

```

**Step 8:** If you want to move a directory, you can also call the same MoveFile() function. Here is the implementation of move button event handler in which we will create first new directory and then move the Dir2 to that directory.

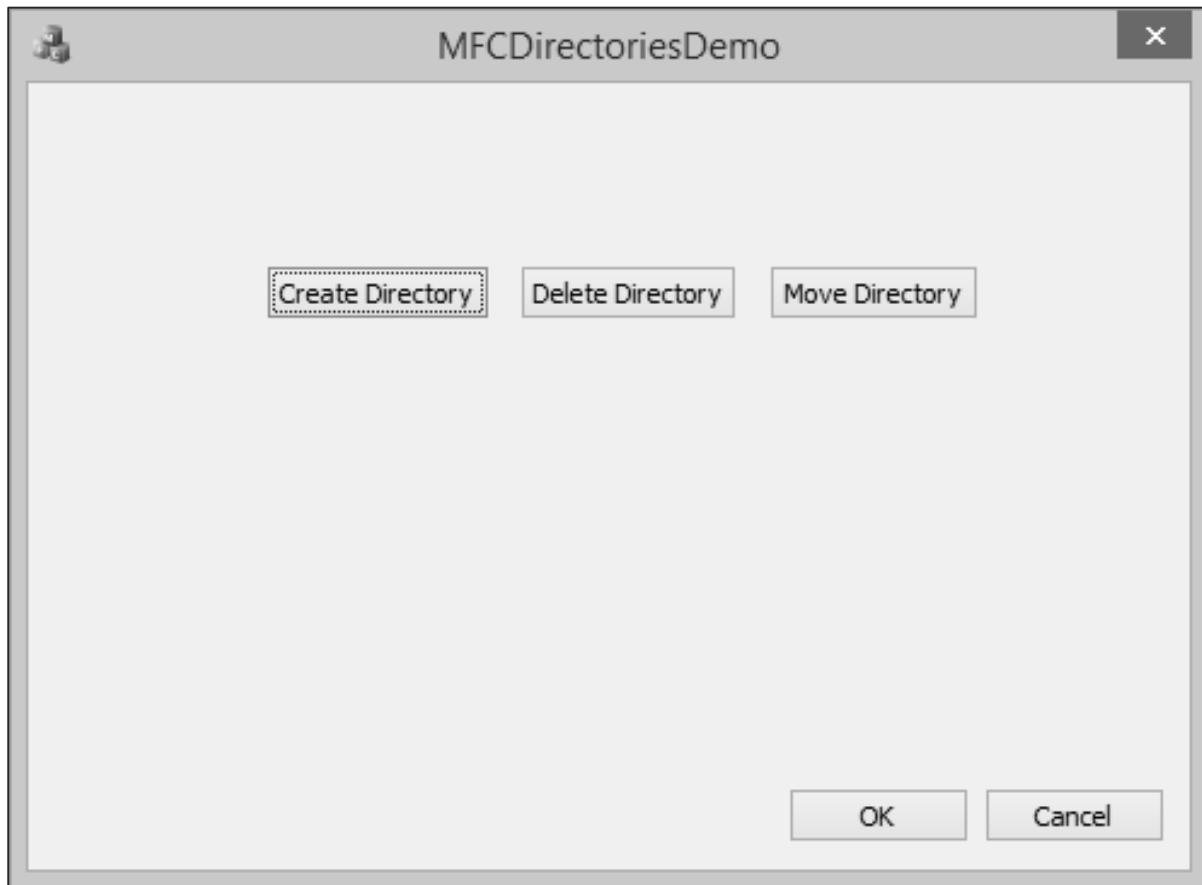
```

void CMFCDirectoriesDemoDlg::OnBnClickedButtonMove()
{
    // TODO: Add your control notification handler code here
    CreateDirectory(L"D:\\MFCDirectory", NULL);

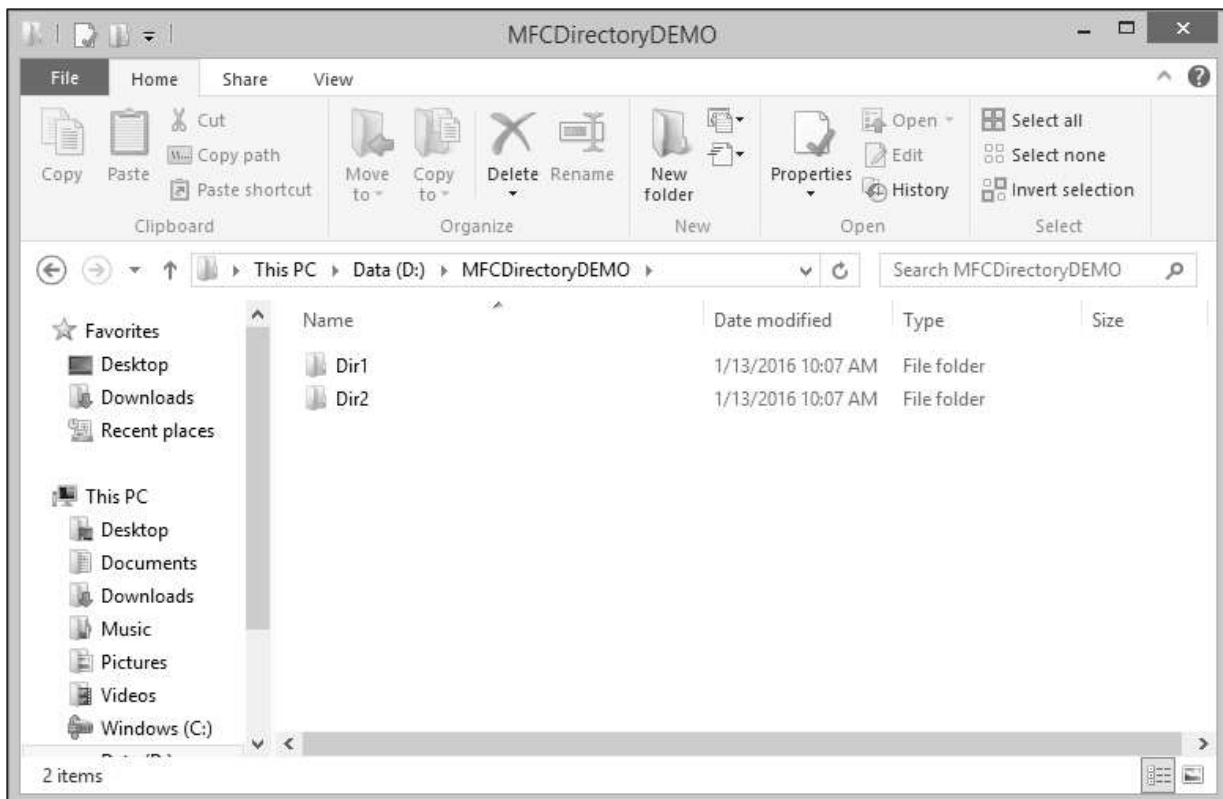
    if (MoveFile(L"D:\\MFCDirectoryDEMO\\Dir1",
                L"D:\\MFCDirectory\\Dir1") == TRUE)
        AfxMessageBox(L"The directory has been moved");
}

```

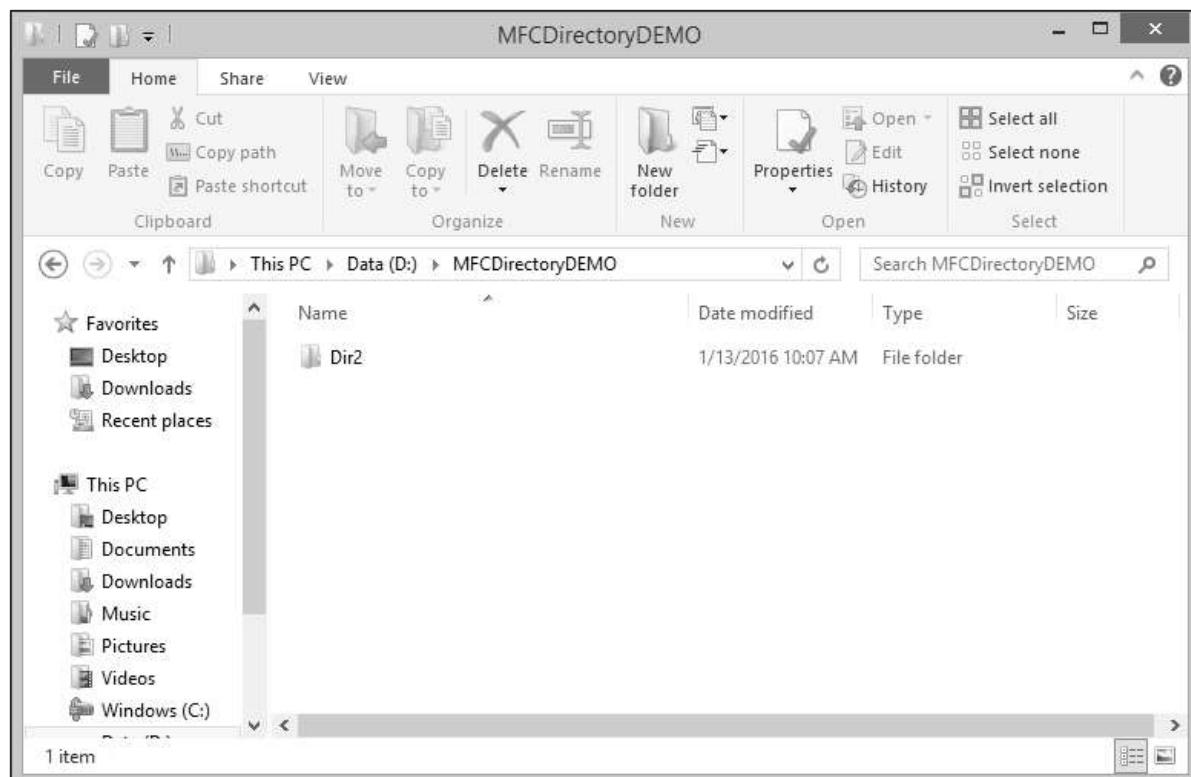
**Step 9:** When the above code is compiled and executed, you will see the following output.



**Step 10:** When you click the Create Directory button, it will create these directories.



**Step 11:** When you click on Delete Directory button, it will delete the Dir1.



## File Processing

---

Most of the **file processing** in an MFC application is performed in conjunction with a class named **CArchive**. The CArchive class serves as a relay between the application and the medium used to either store data or make it available. It allows you to save a complex network of objects in a permanent binary form (usually disk storage) that persists after those objects are deleted.

Here is the list of methods in CArchive class:

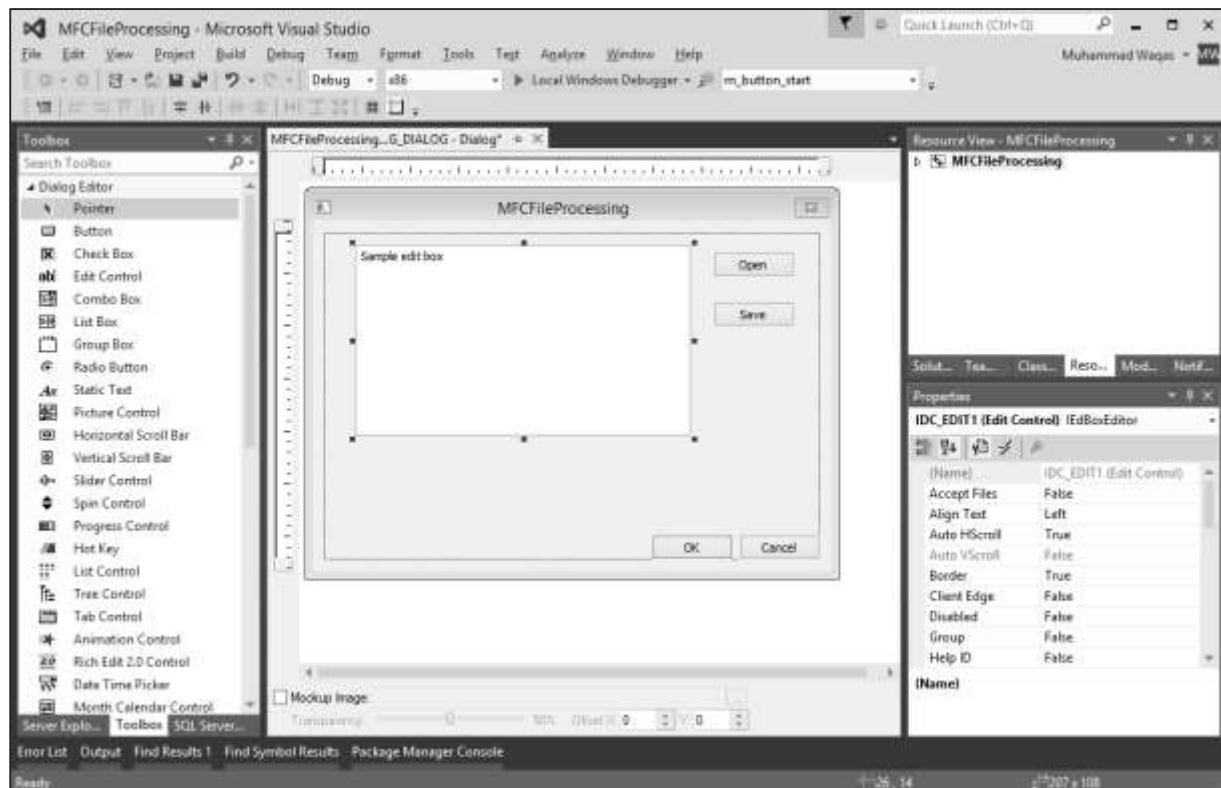
Name	Description
<b>Abort</b>	Closes an archive without throwing an exception.
<b>Close</b>	Flushes unwritten data and disconnects from the <b>CFile</b> .
<b>Flush</b>	Flushes unwritten data from the archive buffer.
<b>GetFile</b>	Gets the CFile object pointer for this archive.
<b>GetObjectSchema</b>	Called from the <b>Serialize</b> function to determine the version of the object that is being deserialized.
<b>IsBufferEmpty</b>	Determines whether the buffer has been emptied during a Windows Sockets receive process.
<b>IsLoading</b>	Determines whether the archive is loading.
<b>IsStoring</b>	Determines whether the archive is storing.
<b>MapObject</b>	Places objects in the map that are not serialized to the file, but that are available for subobjects to reference.
<b>Read</b>	Reads raw bytes.
<b>ReadClass</b>	Reads a class reference previously stored with <b>WriteClass</b> .
<b>ReadObject</b>	Calls an object's Serialize function for loading.
<b>ReadString</b>	Reads a single line of text.
<b>SerializeClass</b>	Reads or writes the class reference to the CArchive object depending on the direction of the CArchive.
<b>SetLoadParams</b>	Sets the size to which the load array grows. Must be called before any object is loaded or before <b>MapObject</b> or <b>ReadObject</b> is called.
<b>SetObjectSchema</b>	Sets the object schema stored in the archive object.
<b>SetStoreParams</b>	Sets the hash table size and the block size of the map used to identify unique objects during the serialization process.
<b>Write</b>	Writes raw bytes.
<b>WriteClass</b>	Writes a reference to the <b>CRuntimeClass</b> to the CArchive.
<b>WriteObject</b>	Calls an object's Serialize function for storing.
<b>WriteString</b>	Writes a single line of text.

Here is the list of operators used to store and retrieve data:

Name	Description
<b>operator &lt;&lt;</b>	Stores objects and primitive types to the archive.
<b>operator &gt;&gt;</b>	Loads objects and primitive types from the archive.

Let us look into a simple example by creating a new MFC dialog based application.

**Step 1:** Drag one edit control and two buttons as shown in the following snapshot.



**Step 2:** Add control variable **m\_editCtrl** and value variable **m\_strEdit** for edit control.

**Step 3:** Add click event handler for Open and Save buttons.

**Step 4:** Here is the implementation of event handlers.

```
void CMFCFileProcessingDlg::OnBnClickedButtonOpen()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    CFile file;

    file.Open(L"ArchiveText.rpr", CFile::modeRead);
    if(file)
        CArchive ar(&file, CArchive::load);
```

```
ar >> m_strEdit;

ar.Close();
file.Close();

UpdateData(FALSE);
}

void CMFCFileProcessingDlg::OnBnClickedButtonSave()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    if (m_strEdit.GetLength() == 0)
    {
        AfxMessageBox(L"You must enter the name of the text.");
        return;
    }

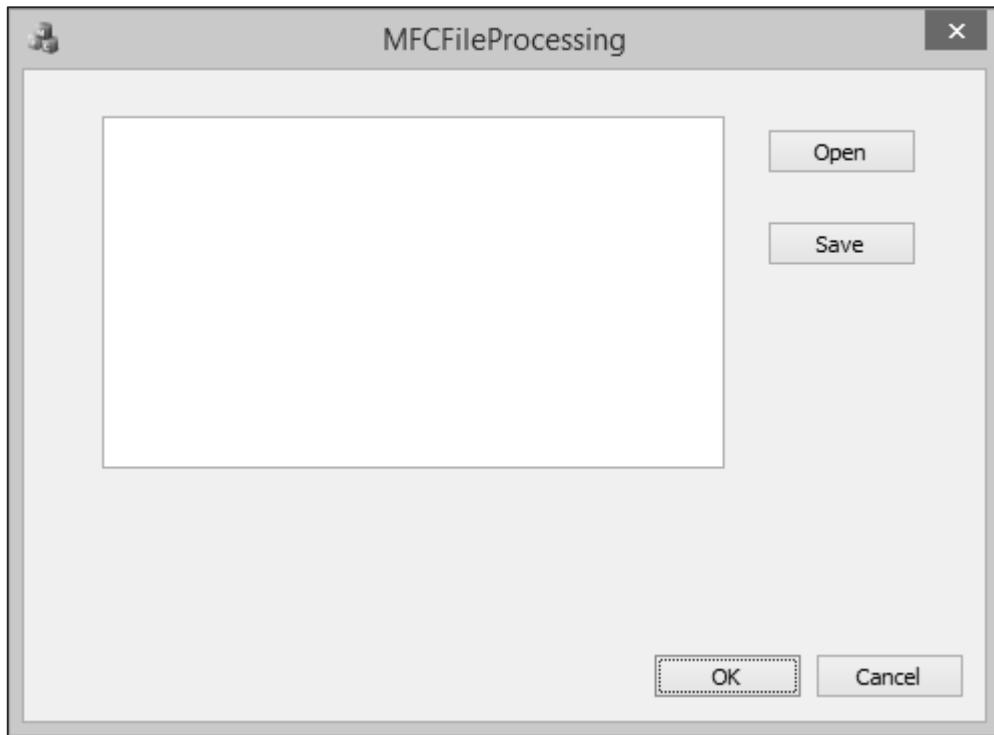
    CFile file;

    file.Open(L"ArchiveText.rpr", CFile::modeCreate | CFile::modeWrite);
    CArcive ar(&file, CArcive::store);

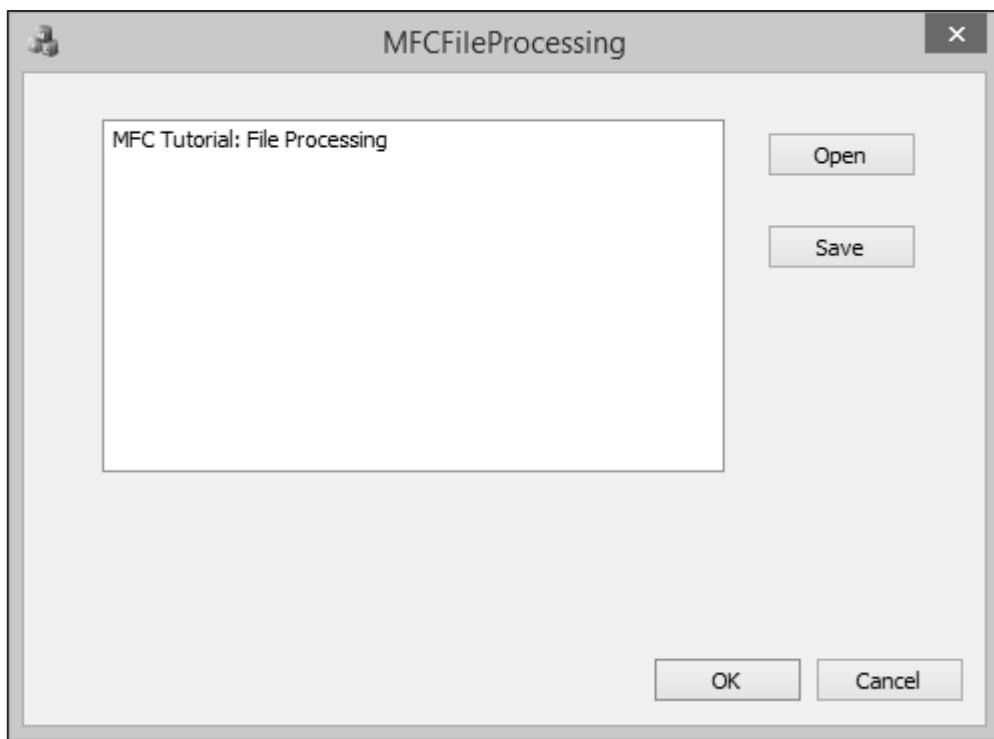
    ar << m_strEdit;

    ar.Close();
    file.Close();
}
```

**Step 5:** When the above code is compiled and executed, you will see the following output.



**Step 6:** Write something and click Save. It will save the data in binary format.



**Step 7:** Remove the test from edit control. As you click Open, observe that the same text is loaded again.

# 15. MFC - Standard I/O

The MFC library provides its own version of file processing. This is done through a class named CStdioFile. The CStdioFile class is derived from CFile. It can handle the reading and writing of Unicode text files as well as ordinary multi-byte text files.

Here is the list of constructors, which can initialize a CStdioFile object:

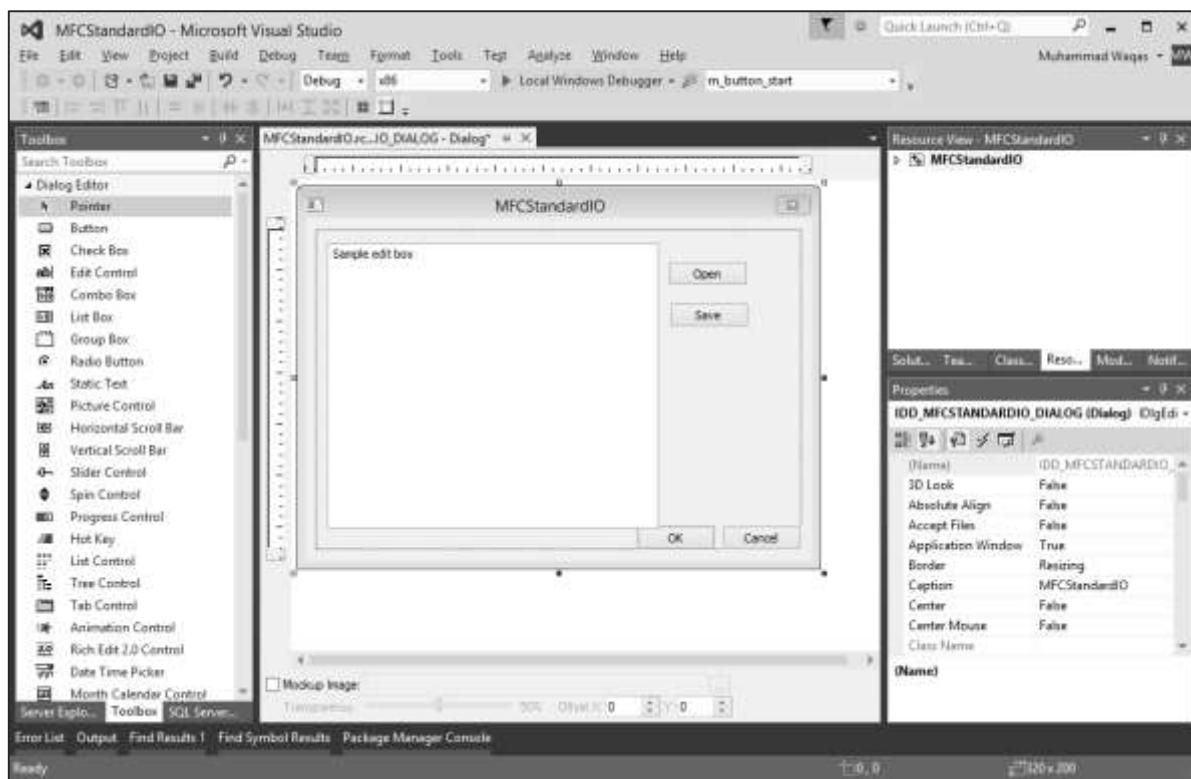
```
CStdioFile();
CStdioFile(CAtlTransactionManager* pTM);
CStdioFile(FILE* pOpenStream);
CStdioFile(LPCTSTR lpszFileName, UINT nOpenFlags);
CStdioFile(LPCTSTR lpszFileName, UINT nOpenFlags, CAtlTransactionManager* pTM);
```

Here is the list of methods in CStdioFile:

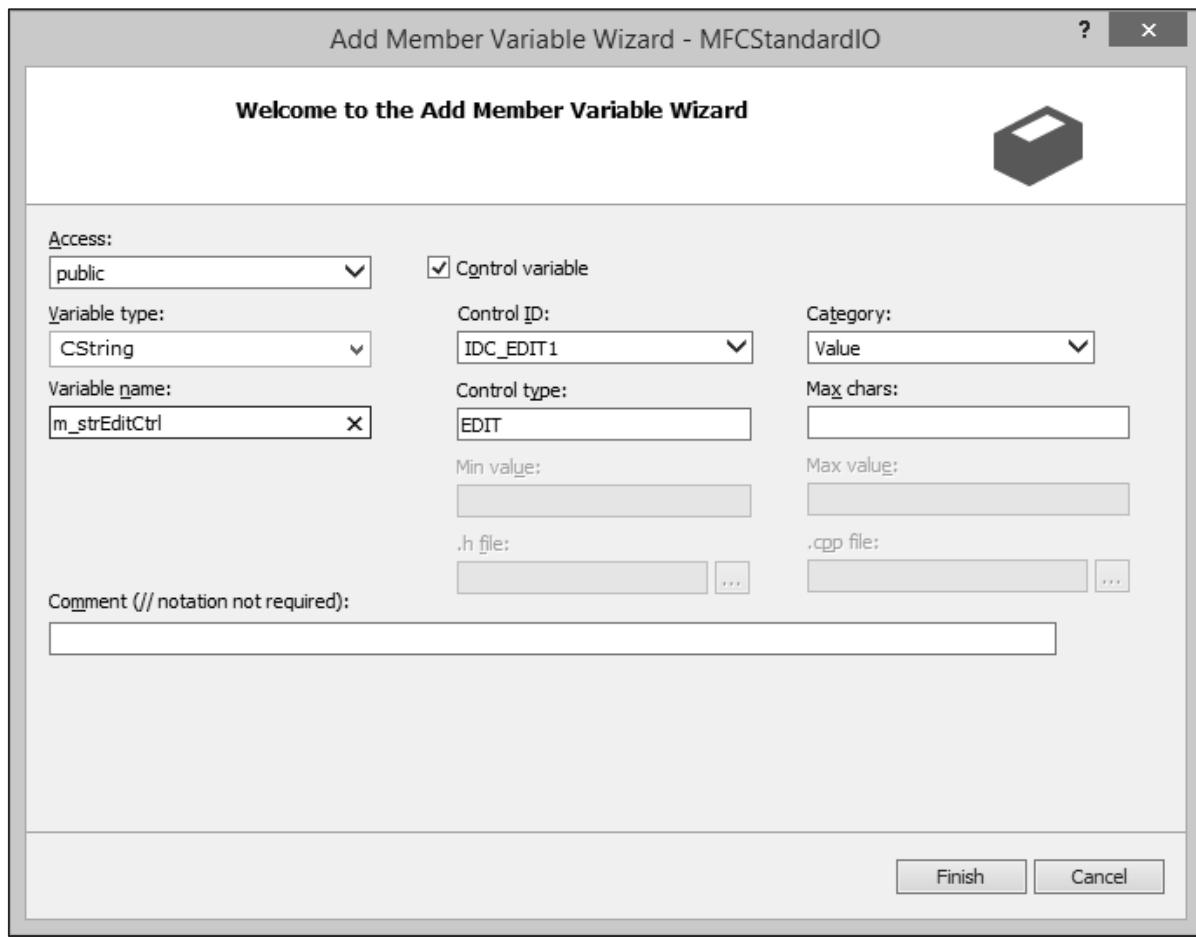
Name	Description
<b>Open</b>	Overloaded. Open is designed for use with the default CStdioFile constructor (Overrides CFile::Open).
<b>ReadString</b>	Reads a single line of text.
<b>Seek</b>	Positions the current file pointer.
<b>WriteString</b>	Writes a single line of text.

Let us look into a simple example again by creating a new MFC dialog based application.

**Step 1:** Drag one edit control and two buttons as shown in the following snapshot.



**Step 2:** Add value variable **m\_strEditCtrl** for edit control.



**Step 3:** Add click event handler for Open and Save buttons.

**Step 4:** Here is the implementation of event handlers.

```
void CMFCStandardIODlg::OnBnClickedButtonOpen()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

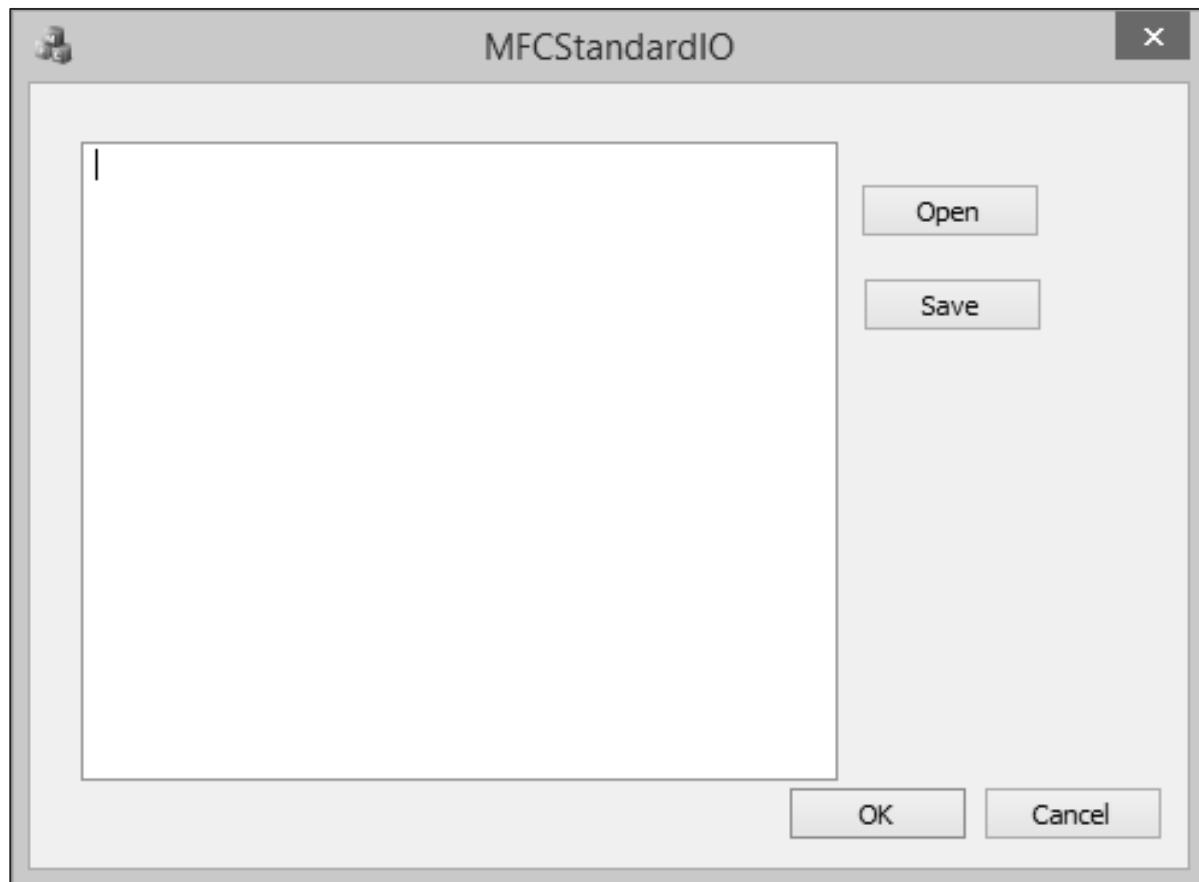
    CStdioFile file;
    file.Open(L"D:\\MFCDirectoryDEMO\\test.txt", CFile::modeRead |
    CFile::typeText);

    file.ReadString(m_strEditCtrl);
    file.Close();
    UpdateData(FALSE);
}

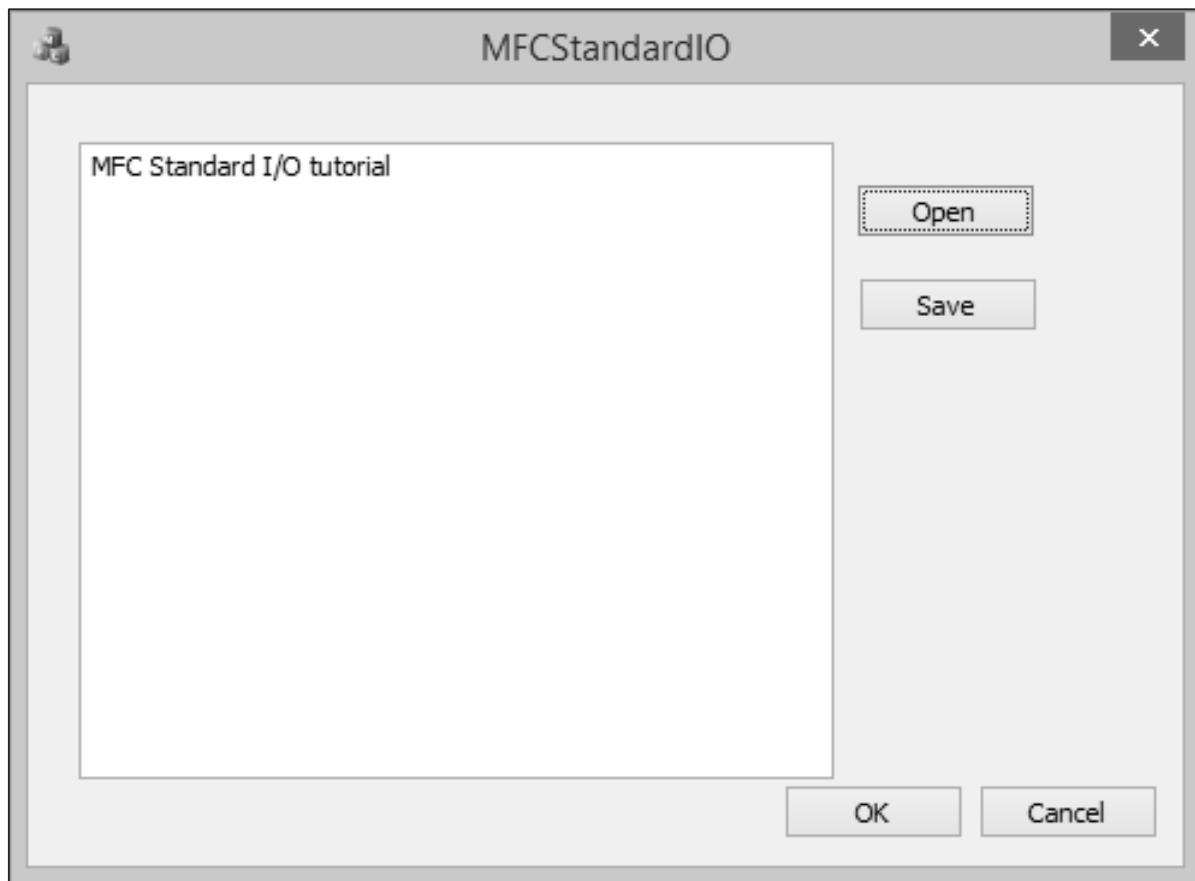
void CMFCStandardIODlg::OnBnClickedButtonSave()
```

```
{  
    // TODO: Add your control notification handler code here  
    UpdateData(TRUE);  
    CStdioFile file;  
    if (m_strEditCtrl.GetLength() == 0)  
    {  
        AfxMessageBox(L"You must specify the text.");  
        return;  
    }  
    file.Open(L"D:\\MFCDirectoryDEMO\\test.txt", CFile::modeCreate |  
    CFile::modeWrite | CFile::typeText);  
    file.WriteString(m_strEditCtrl);  
    file.Close();  
}
```

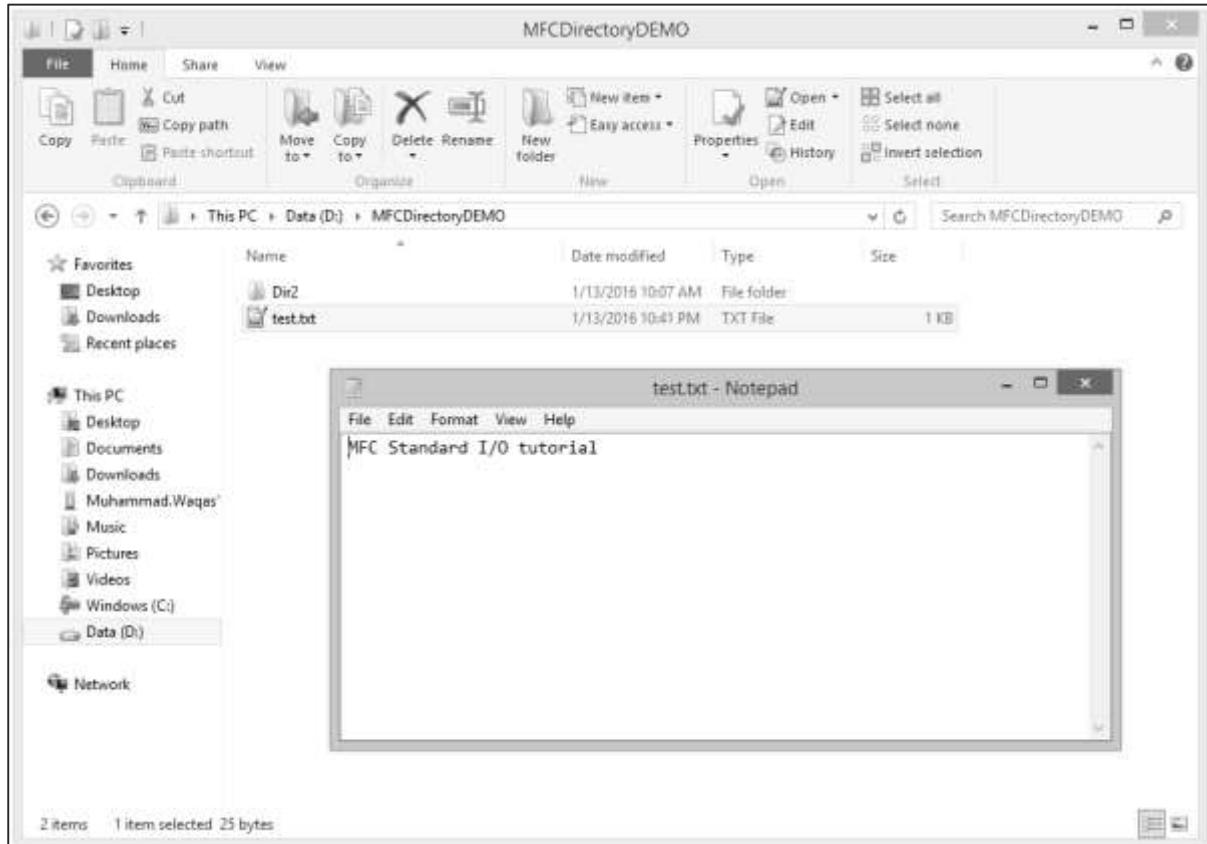
**Step 5:** When the above code is compiled and executed, you will see the following output.



**Step 6:** Write something and click Save. It will save the data in \*.txt file.



**Step 7:** If you look at the location of the file, you will see that it contains the test.txt file.



**Step 8:** Now, close the application. Run the same application. When you click Open, the same text loads again.

**Step 9:** It starts by opening the file, reading the file, followed by updating the Edit Control.

# 16. MFC - Document View

The **Document/View architecture** is the foundation used to create applications based on the Microsoft Foundation Classes library. It allows you to make distinct the different parts that compose a computer program including what the user sees as part of your application and the document a user would work on. This is done through a combination of separate classes that work as an ensemble.

The parts that compose the Document/View architecture are a frame, one or more documents, and the view. Put together, these entities make up a usable application.

## View

---

A **view** is the platform the user is working on to do his or her job. To let the user do anything on an application, you must provide a view, which is an object based on the CView class. You can either directly use one of the classes derived from CView or you can derive your own custom class from CView or one of its child classes.

## Document

---

A **document** is similar to a bucket. For a computer application, a document holds the user's data. To create the document part of this architecture, you must derive an object from the CDocument class.

## Frame

---

As the name suggests, a **frame** is a combination of the building blocks, the structure, and the borders of an item. A frame gives "physical" presence to a window. It also defines the location of an object with regards to the Windows desktop.

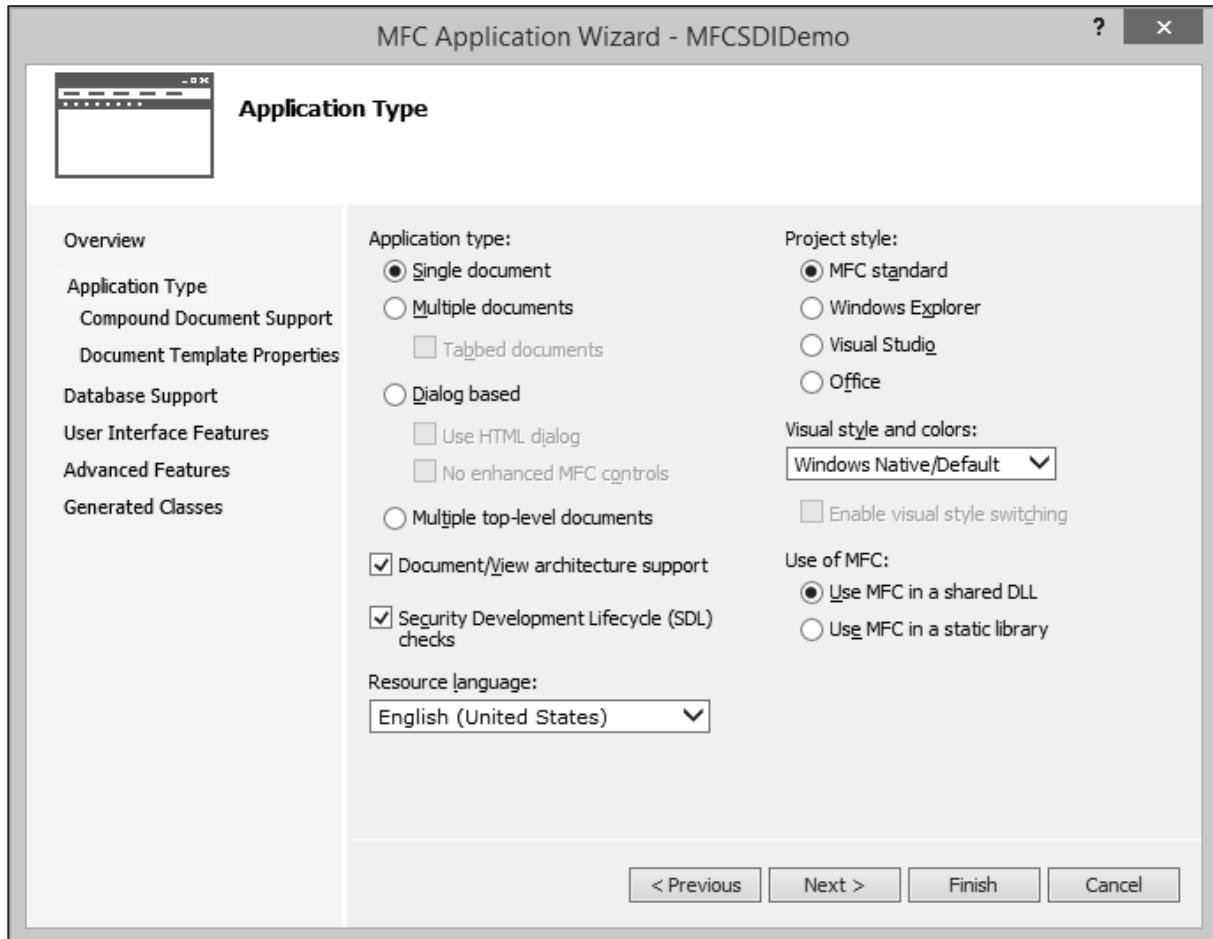
## Single Document Interface (SDI)

---

The expression **Single Document Interface** or SDI refers to a document that can present only one view to the user. This means that the application cannot display more than one document at a time. If you want to view another type of document of the current application, you must create another instance of the application. Notepad and WordPad are examples of SDI applications.

Let us look into a simple example of single document interface or SDI by creating a new MFC dialog based application.

**Step 1:** Let us create a new MFC Application **MFCSDIDemo** with below mentioned settings.



**Step 2:** Select Single document from the Application type and MFC standard from Project Style.

**Step 3:** Click Finish to Continue.

**Step 4:** Once the project is created, run the application and you will see the following output.

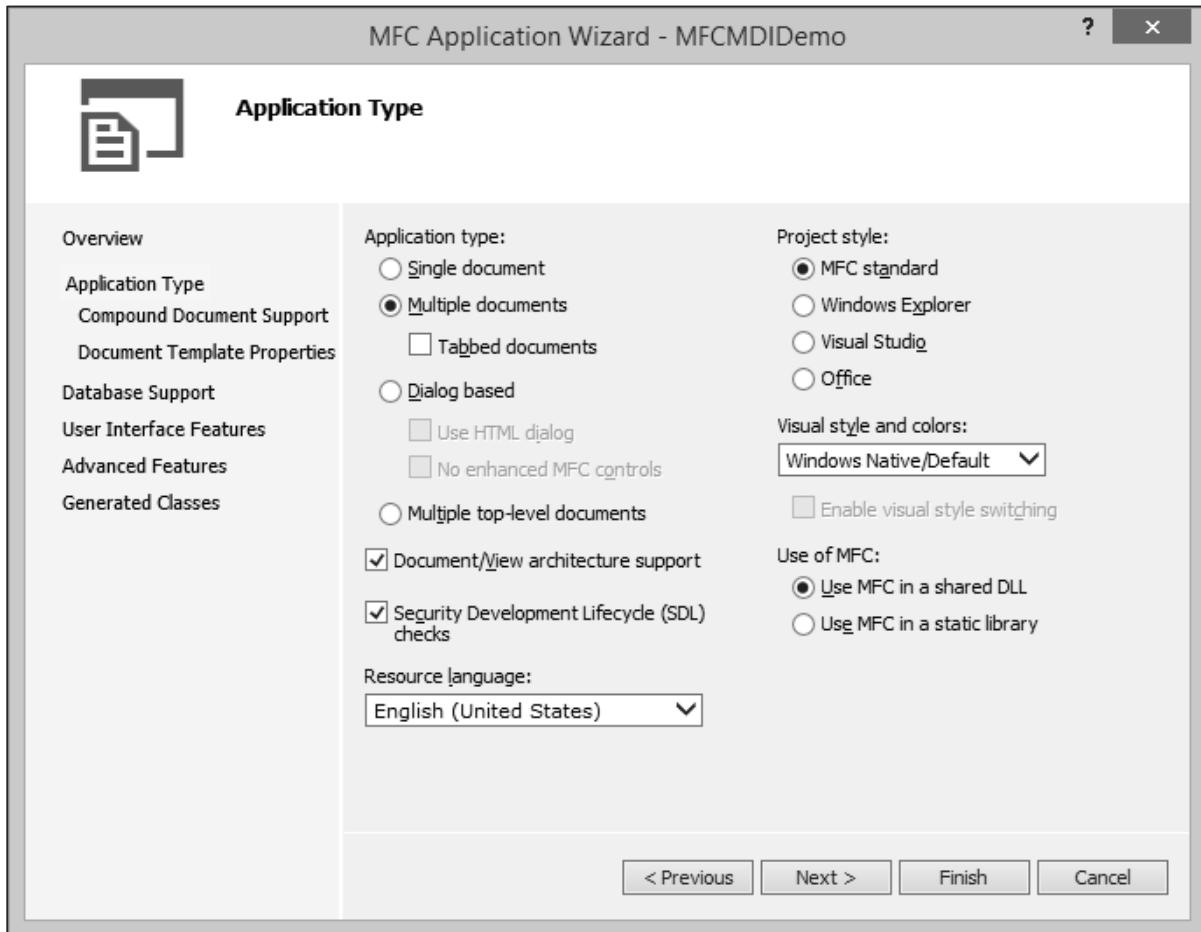


## Multiple Document Interface (MDI)

An application is referred to as a **Multiple Document Interface**, or MDI, if the user can open more than one document in the application without closing it. To provide this functionality, the application provides a parent frame that acts as the main frame of the computer program. Inside this frame, the application allows creating views with individual frames, making each view distinct from the other.

Let us look into a simple example of single document interface or SDI by creating a new MFC dialog based application.

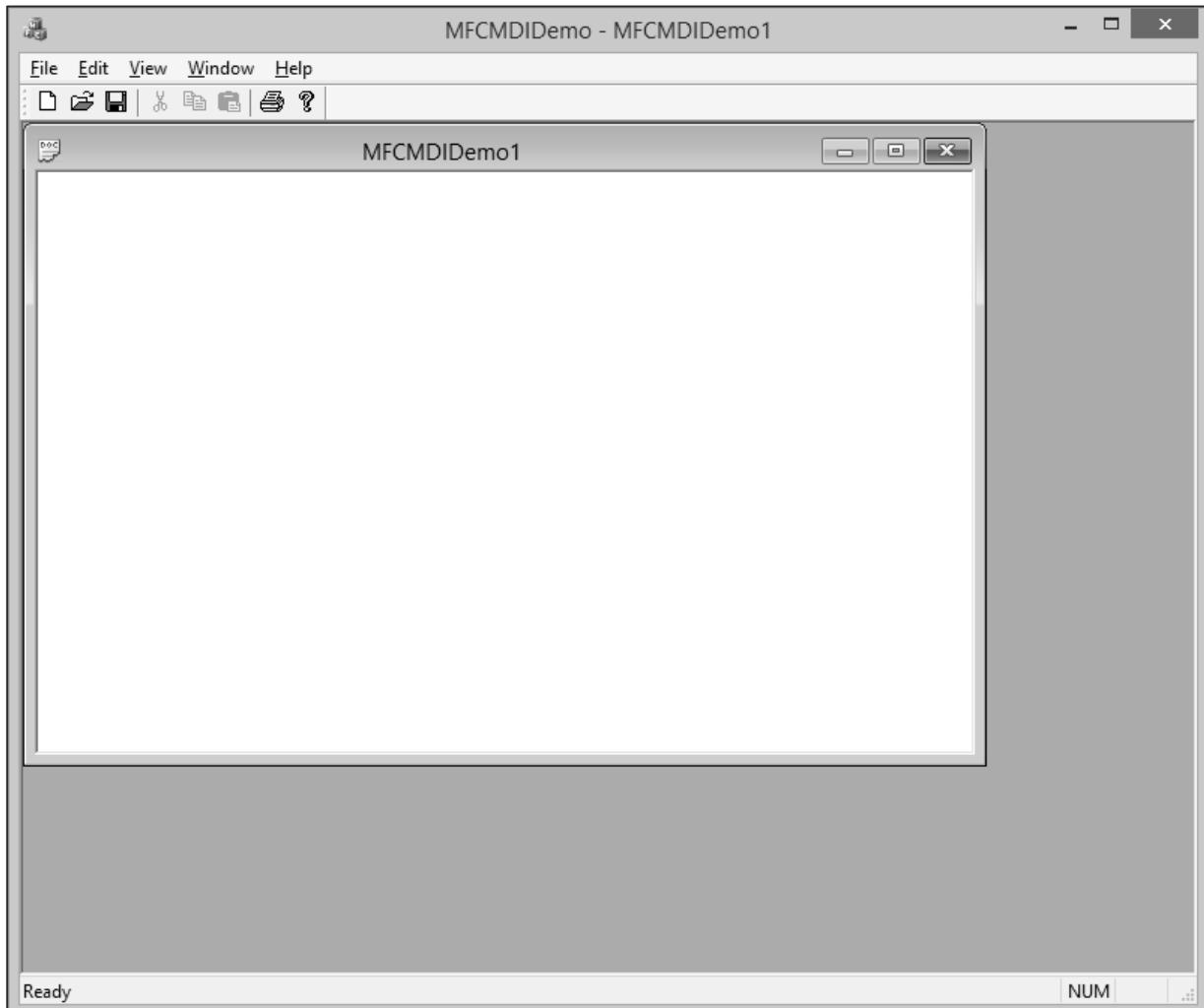
**Step 1:** Let us create a new MFC Application **MFCMDIDemo** with below mentioned settings.



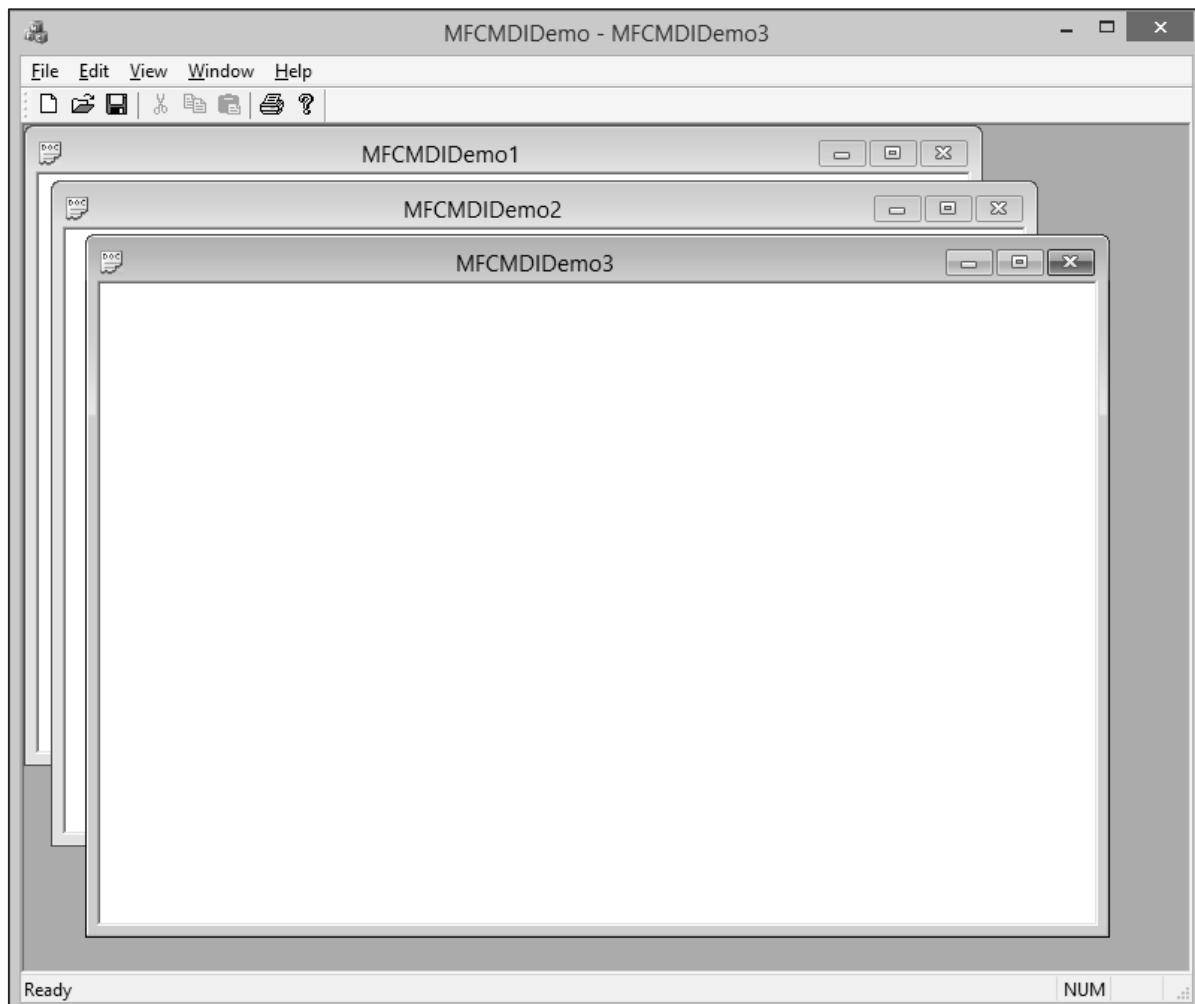
**Step 2:** Select Single document from the Application type and MFC standard from Project Style.

**Step 3:** Click Finish to Continue.

**Step 4:** Once the project is created, run the application and you will see the following output.



**Step 5:** When you click on File -> New menu option, it will create another child window as shown in the following snapshot.



**Step 6:** In Multiple Document Interface (MDI) applications, there is one main frame per application. In this case, a CMDIFrameWnd, and one CMDIChildWnd derived child frame for each document.

# 17. MFC - Strings

**Strings** are objects that represent sequences of characters. The C-style character string originated within the C language and continues to be supported within C++.

- This string is actually a one-dimensional array of characters which is terminated by a null character '\0'.
- A null-terminated string contains the characters that comprise the string followed by a null.

Here is the simple example of character array.

```
char word[12] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0' };
```

Following is another way to represent it.

```
char word[] = "Hello, World";
```

Microsoft Foundation Class (MFC) library provides a class to manipulate string called **CString**. Following are some important features of CString.

- CString does not have a base class.
- A CString object consists of a variable-length sequence of characters.
- CString provides functions and operators using a syntax similar to that of Basic.
- Concatenation and comparison operators, together with simplified memory management, make CString objects easier to use than ordinary character arrays.

Here is the constructor of CString.

Method	Description
CString	Constructs CString objects in various ways

Here is a list of Array Methods:

Method	Description
<b>GetLength</b>	Returns the number of characters in a CString object.
<b>IsEmpty</b>	Tests whether a CString object contains no characters.
<b>Empty</b>	Forces a string to have 0 length.
<b>GetAt</b>	Returns the character at a specified position.
<b>SetAt</b>	Sets a character at a specified position.

Here is a list of Comparison Methods:

<b>Method</b>	<b>Description</b>
<b>Compare</b>	Compares two strings (case sensitive).
<b>CompareNoCase</b>	Compares two strings (case insensitive).

Here is a list of Extraction Methods:

<b>Method</b>	<b>Description</b>
<b>Mid</b>	Extracts the middle part of a string (like the Basic MID\$ function).
<b>Left</b>	Extracts the left part of a string (like the Basic LEFT\$ function).
<b>Right</b>	Extracts the right part of a string (like the Basic RIGHT\$ function).
<b>SpanIncluding</b>	Extracts a substring that contains only the characters in a set.
<b>SpanExcluding</b>	Extracts a substring that contains only the characters not in a set.

Here is a list of Conversion Methods.

<b>Method</b>	<b>Description</b>
<b>MakeUpper</b>	Converts all the characters in this string to uppercase characters.
<b>MakeLower</b>	Converts all the characters in this string to lowercase characters.
<b>MakeReverse</b>	Reverses the characters in this string.
<b>Format</b>	Format the string as sprintf does.
<b>TrimLeft</b>	Trim leading white-space characters from the string.
<b>TrimRight</b>	Trim trailing white-space characters from the string.

Here is a list of Searching Methods.

<b>Method</b>	<b>Description</b>
<b>Find</b>	Finds a character or substring inside a larger string.
<b>ReverseFind</b>	Finds a character inside a larger string; starts from the end.
<b>FindOneOf</b>	Finds the first matching character from a set.

Here is a list of Buffer Access Methods.

<b>Method</b>	<b>Description</b>
<b>GetBuffer</b>	Returns a pointer to the characters in the CString.
<b>GetBufferSetLength</b>	Returns a pointer to the characters in the CString, truncating to the specified length.
<b>ReleaseBuffer</b>	Releases control of the buffer returned by GetBuffer.

<b>FreeExtra</b>	Removes any overhead of this string object by freeing any extra memory previously allocated to the string.
<b>LockBuffer</b>	Disables reference counting and protects the string in the buffer.
<b>UnlockBuffer</b>	Enables reference counting and releases the string in the buffer.

Here is a list of Windows-Specific Methods.

<b>Method</b>	<b>Description</b>
<b>AllocSysString</b>	Allocates a BSTR from CString data.
<b>SetSysString</b>	Sets an existing BSTR object with data from a CString object.
<b>LoadString</b>	Loads an existing CString object from a Windows CE resource.

Following are the different operations on CString objects:

## Create String

You can create a string by either using a string literal or creating an instance of CString class.

```
BOOL CMFCStringDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

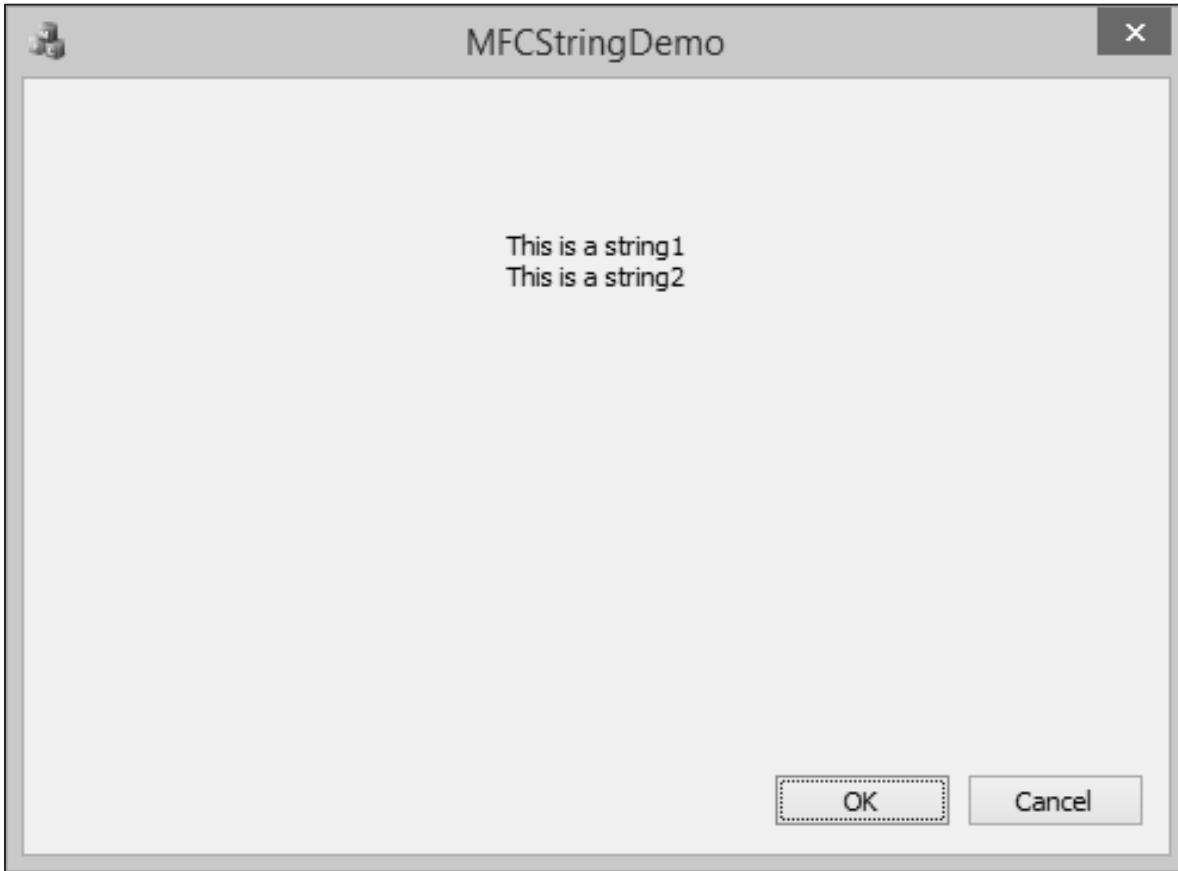
    CString string1 = _T("This is a string1");
    CString string2("This is a string2");

    m_strText.Append(string1 + L"\n");
    m_strText.Append(string2);

    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed, you will see the following output.



## Empty String

You can create an empty string by either using an empty string literal or by using `CString::Empty()` method. You can also check whether a string is empty or not using Boolean property `isEmpty`.

```
BOOL CMFCStringDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    CString string1 = _T("");
    CString string2;
```

```
string2.Empty();

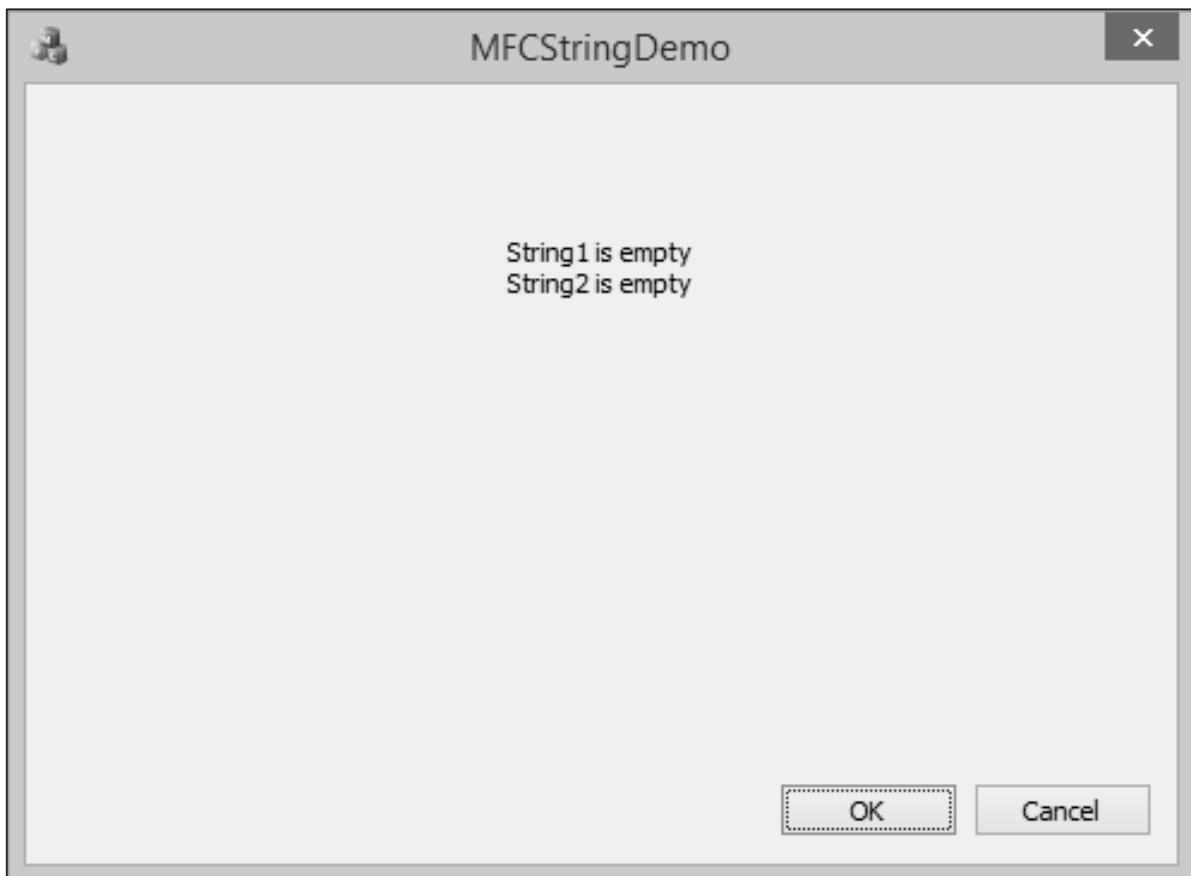
if(string1.IsEmpty())
    m_strText.Append(L"String1 is empty\n");
else
    m_strText.Append(string1 + L"\n");

if(string2.IsEmpty())
    m_strText.Append(L"String2 is empty");
else
    m_strText.Append(string2);

UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed you will see the following output.



## String Concatenation

To concatenate two or more strings, you can use + operator to concatenate two strings or a CString::Append() method.

```
BOOL CMFCStringDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

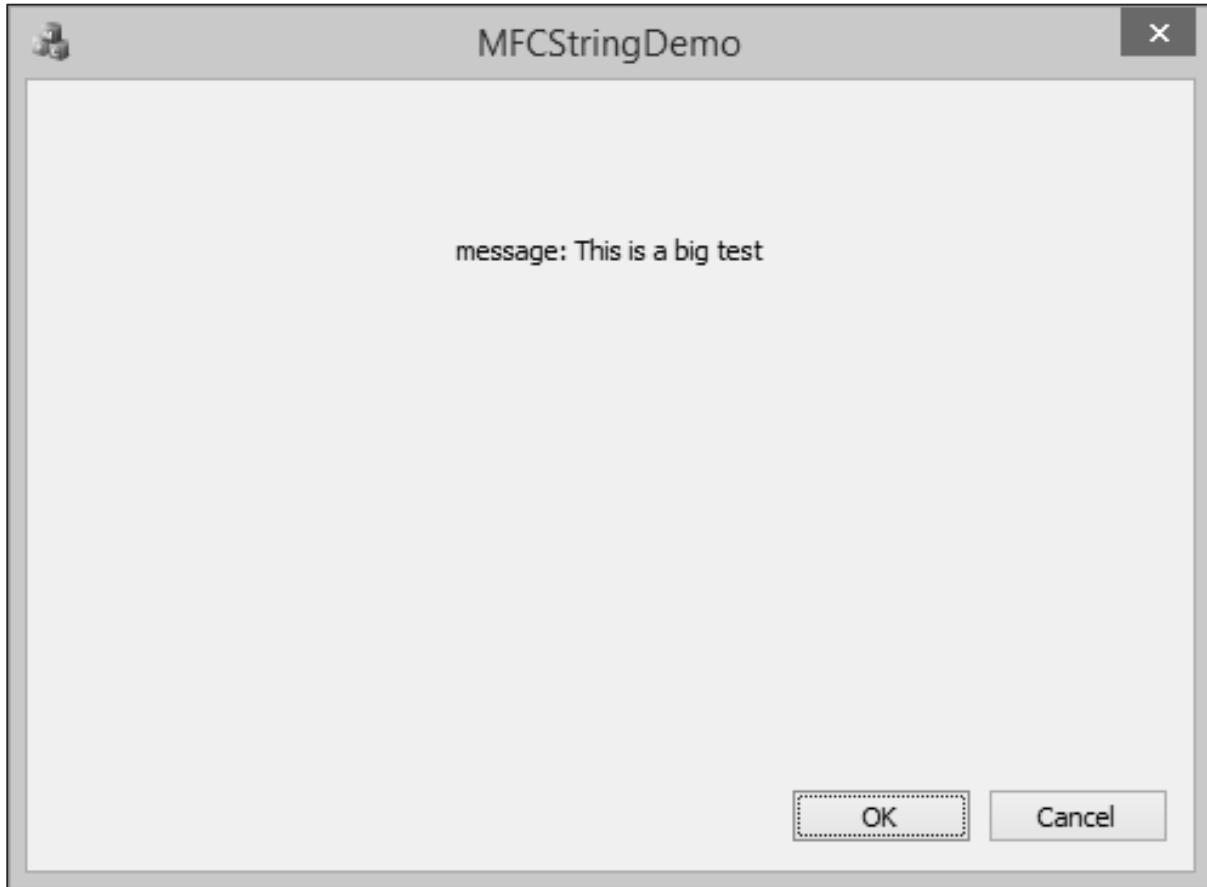
    //To concatenate two CString objects
    CString s1 = _T("This ");        // Cascading concatenation
    s1 += _T("is a ");
    CString s2 = _T("test");
    CString message = s1;
    message.Append(_T("big ") + s2);
    // Message contains "This is a big test".

    m_strText = L"message: " + message;

    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed you will see the following output.



## String Length

To find the length of the string you can use the `CString::GetLength()` method, which returns the number of characters in a `CString` object.

```
BOOL CMFCStringDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    CString string1 = _T("This is string 1");
    int length = string1.GetLength();
    CString strLen;
```

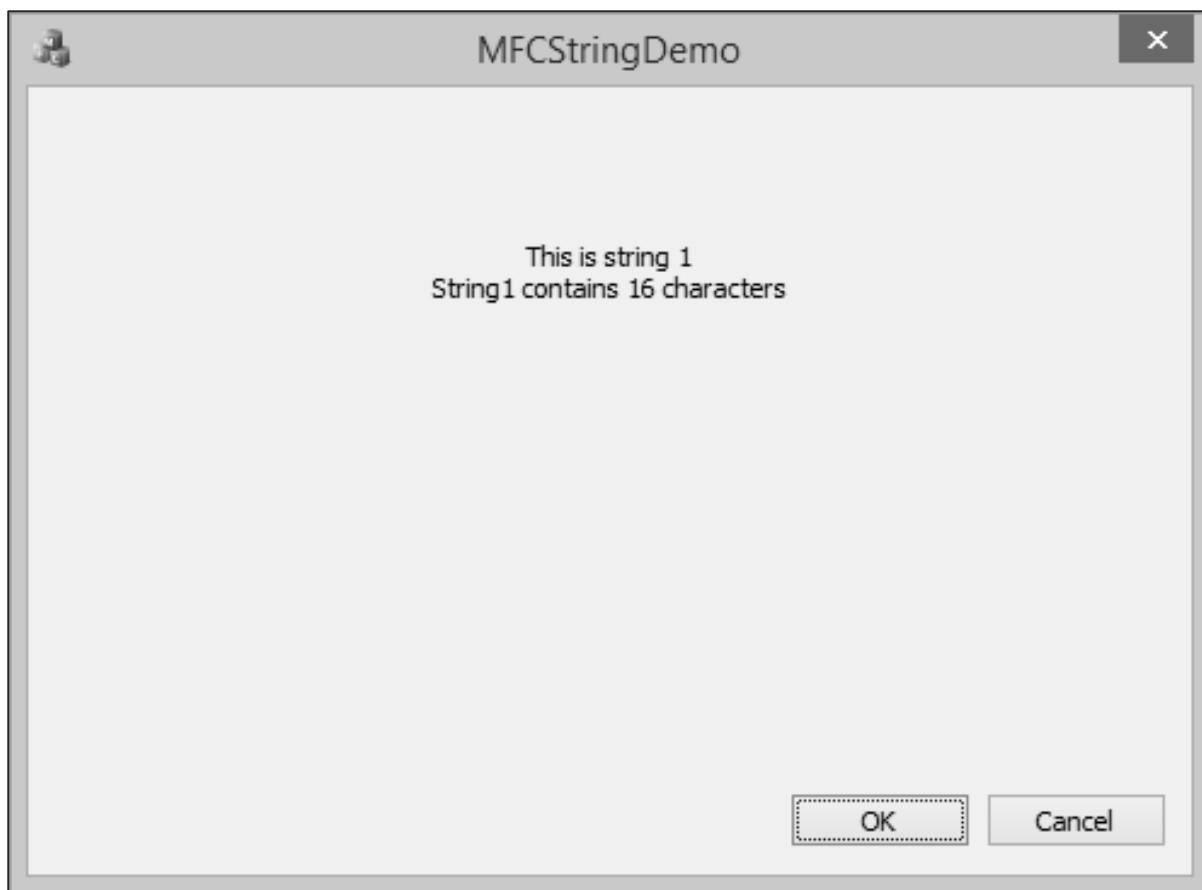
306

```
strLen.Format(L"\nString1 contains %d characters", length);
m_strText = string1 + strLen;

UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed you will see the following output.



## String Comparison

To compare two strings variables you can use == operator

```
BOOL CMFCStringDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    CString string1 = _T("Hello");
    CString string2 = _T("World");

    CString string3 = _T("MFC Tutorial");
    CString string4 = _T("MFC Tutorial");

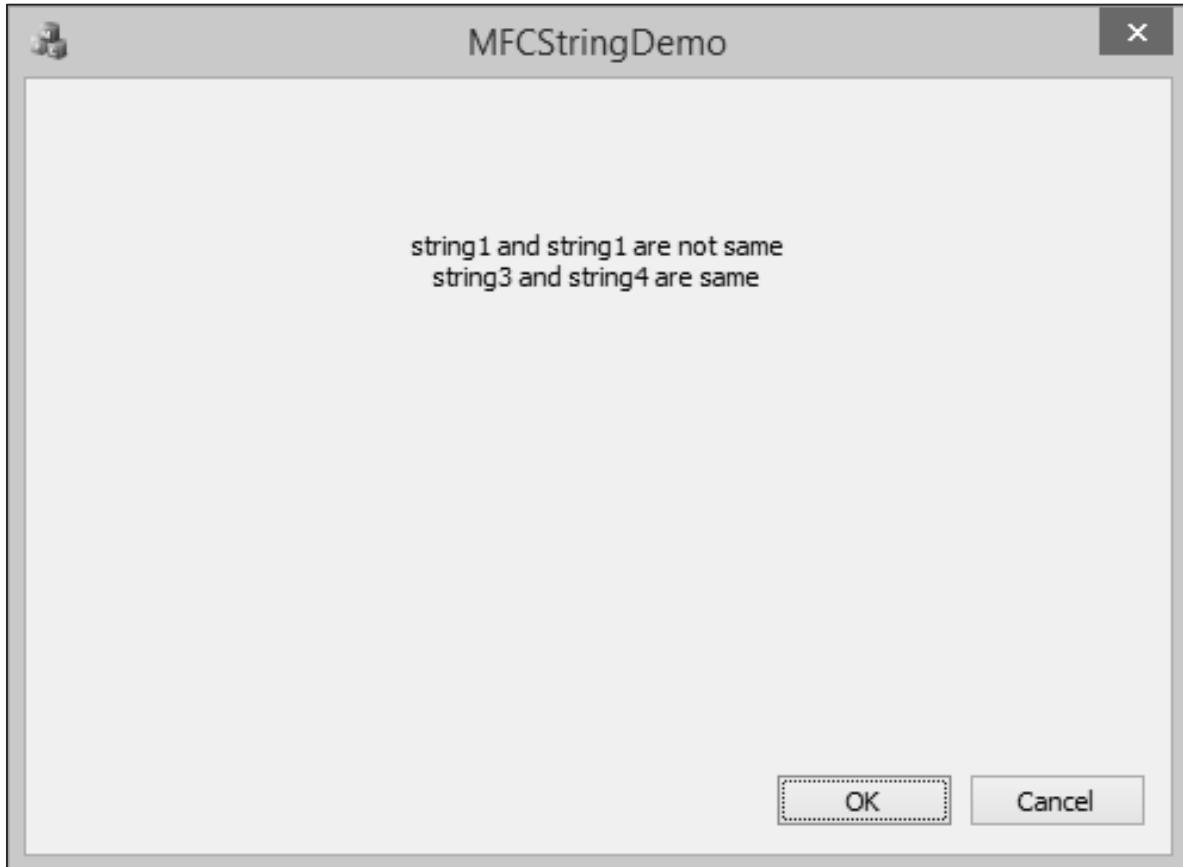
    if (string1 == string2)
        m_strText = "string1 and string1 are same\n";
    else
        m_strText = "string1 and string1 are not same\n";

    if (string3 == string4)
        m_strText += "string3 and string4 are same";
    else
        m_strText += "string3 and string4 are not same";

    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed you will see the following output.



# 18. MFC - CArray

**CArray** is a collection that is best used for data that is to be accessed in a random or non sequential manner. CArray class supports arrays that are like C arrays, but can dynamically shrink and grow as necessary.

- Array indexes always start at position 0.
- You can decide whether to fix the upper bound or enable the array to expand when you add elements past the current bound.
- Memory is allocated contiguously to the upper bound, even if some elements are null.

Here is a list of CArray class methods.

Name	Description
<b>Add</b>	Adds an element to the end of the array; grows the array if necessary.
<b>Append</b>	Appends another array to the array; grows the array if necessary
<b>Copy</b>	Copies another array to the array; grows the array if necessary.
<b>ElementAt</b>	Returns a temporary reference to the element pointer within the array.
<b>FreeExtra</b>	Frees all unused memory above the current upper bound.
<b>GetAt</b>	Returns the value at a given index.
<b>GetCount</b>	Gets the number of elements in this array.
<b>GetData</b>	Allows access to elements in the array. Can be <b>NULL</b> .
<b>GetSize</b>	Gets the number of elements in this array.
<b>GetUpperBound</b>	Returns the largest valid index.
<b>InsertAt</b>	Inserts an element (or all the elements in another array) at a specified index.
<b>IsEmpty</b>	Determines whether the array is empty.
<b>RemoveAll</b>	Removes all the elements from this array.
<b>RemoveAt</b>	Removes an element at a specific index.
<b>SetAt</b>	Sets the value for a given index; array not allowed to grow.
<b>SetAtGrow</b>	Sets the value for a given index; grows the array if necessary.
<b>SetSize</b>	Sets the number of elements to be contained in this array.

Following are the different operations on CArray objects:

## Create CArray Object

---

To create a collection of CArray values or objects, you must first decide the type of values of the collection. You can use one of the existing primitive data types such as int, CString, double etc. as shown below;

```
CArray<CString, CString> strArray;
```

## Add items

---

To add an item you can use CArray::Add() function. It adds an item at the end of the array. In the OnInitDialog(), CArray object is created and three names are added as shown in the following code.

```
CArray<CString, CString> strArray;

//Add names to CArray
strArray.Add(L"Ali");
strArray.Add(L"Ahmed");
strArray.Add(L"Mark");
```

## Retrieve Items

---

To retrieve any item, you can use the CArray::GetAt() function. This function takes one integer parameter as an index of the array.

**Step 1:** Let us look at a simple example, which will retrieve all the names.

```
//Retrieve names from CArray
for (int i = 0; i < strArray.GetSize(); i++)
{
    m_strText.Append(strArray.GetAt(i) + L"\n");
}
```

**Step 2:** Here is the complete implementation of CMFCCArrayDlg::OnInitDialog()

```
BOOL CMFCCArrayDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();
```

```
// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

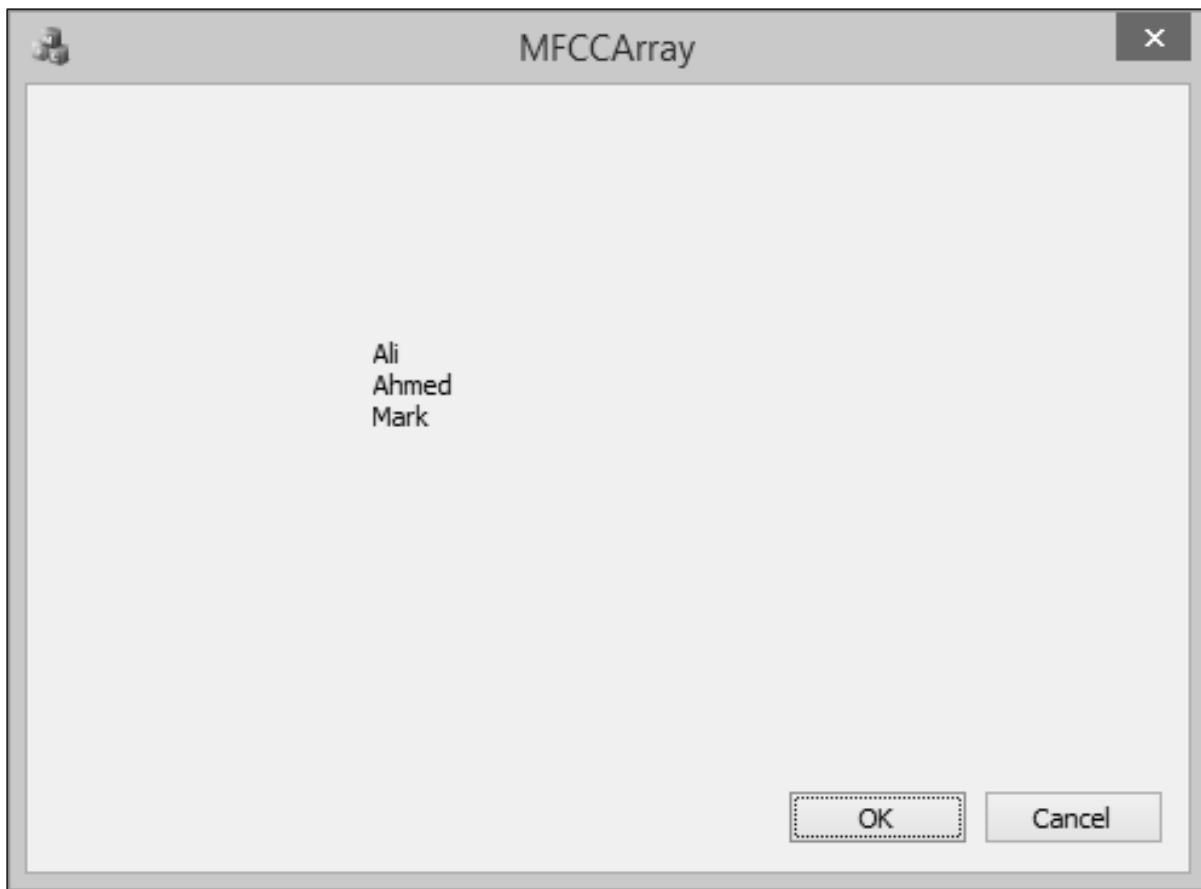
// TODO: Add extra initialization here
CArray<CString, CString> strArray;

//Add names to CArray
strArray.Add(L"Ali");
strArray.Add(L"Ahmed");
strArray.Add(L"Mark");

//Retrive names from CArray
for (int i = 0; i < strArray.GetSize(); i++)
{
    m_strText.Append(strArray.GetAt(i) + L"\n");
}

UpdateData(FALSE);
return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 3:** When the above code is compiled and executed, you will see the following output.



## Add Items in the Middle

To add item in the middle of array you can use the `CArray::InsertAt()` function. It takes two parameters — First, the index and Second, the value.

Let us insert a new item at index 1 as shown in the following code.

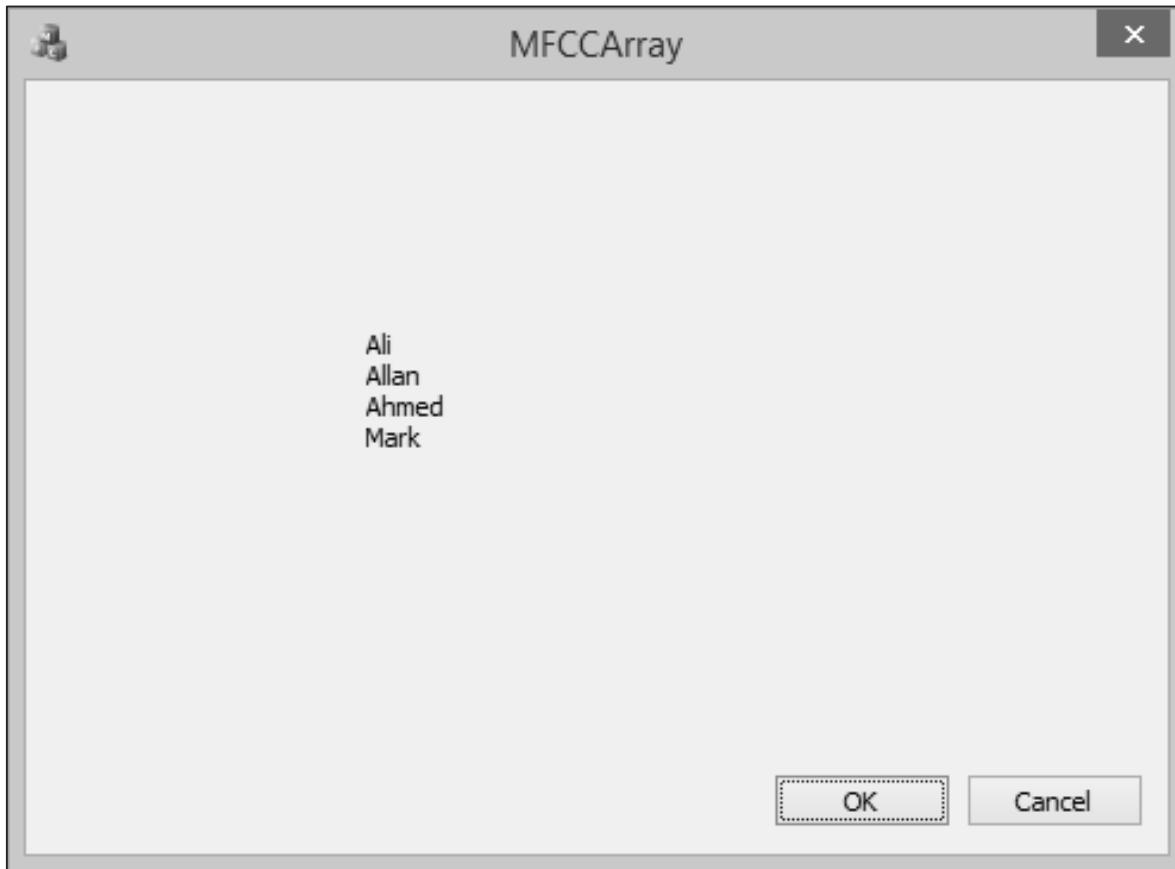
```
BOOL CMFCCArrayDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    CArray<CString, CString> strArray;
```

```
//Add names to CArray  
strArray.Add(L"Ali");  
strArray.Add(L"Ahmed");  
strArray.Add(L"Mark");  
  
strArray.InsertAt(1, L"Allan");  
  
//Retrive names from CArray  
for (int i = 0; i < strArray.GetSize(); i++)  
{  
    m_strText.Append(strArray.GetAt(i) + L"\n");  
}  
  
UpdateData(FALSE);  
return TRUE; // return TRUE unless you set the focus to a control  
}
```

When the above code is compiled and executed, you will see the following output. You can now see the name Allan added as the second index.



## Update Item Value

To update item in the middle of array you can use the CArray::SetAt() function. It takes two parameters — First, the index and Second, the value.

Let us update the third element in the array as shown in the following code.

```
BOOL CMFCCArrayDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // TODO: Add extra initialization here
    CArray<CString, CString> strArray;
```

```
//Add names to CArray
strArray.Add(L"Ali");
strArray.Add(L"Ahmed");
strArray.Add(L"Mark");

strArray.InsertAt(1, L"Allan");

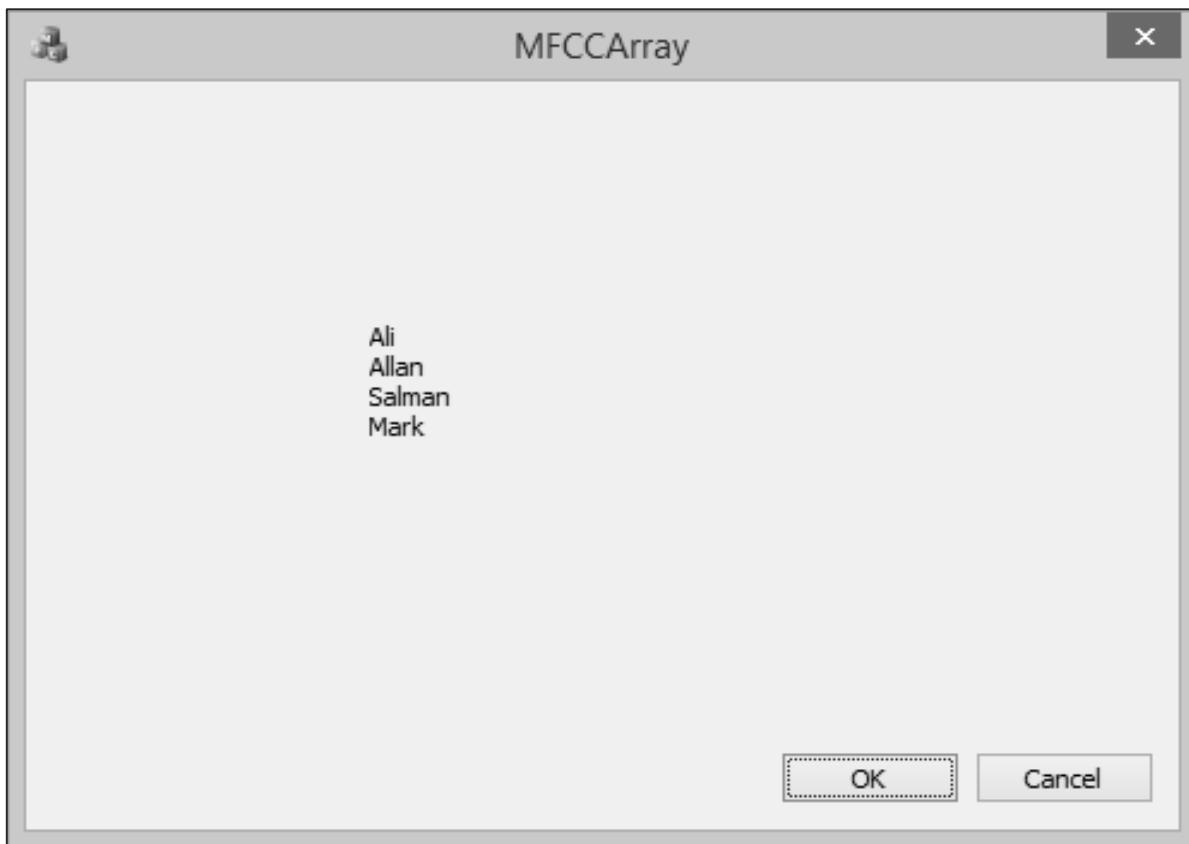
strArray.SetAt(2, L"Salman");

//Retrive names from CArray
for (int i = 0; i < strArray.GetSize(); i++)
{
    m_strText.Append(strArray.GetAt(i) + L"\n");
}

UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed, you will see the following output. You can now see that the value of third element is updated.



## Copy Array

To copy the entire array into another CArray object, you can use `CArray::Copy()` function.

**Step 1:** Let us create another array and copy all the elements from first array as shown in the following code.

```
BOOL CMFCCArrayDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;
```

```

bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
ASSERT(bNameValid);
if (!strAboutMenu.IsEmpty())
{
    pSysMenu->AppendMenu(MF_SEPARATOR);
    pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

// TODO: Add extra initialization here
CArray<CString, CString> strArray;
//Add names to CArray
strArray.Add(L"Ali");
strArray.Add(L"Ahmed");
strArray.Add(L"Mark");

strArray.InsertAt(1, L"Allan");

strArray.SetAt(2, L"Salman");

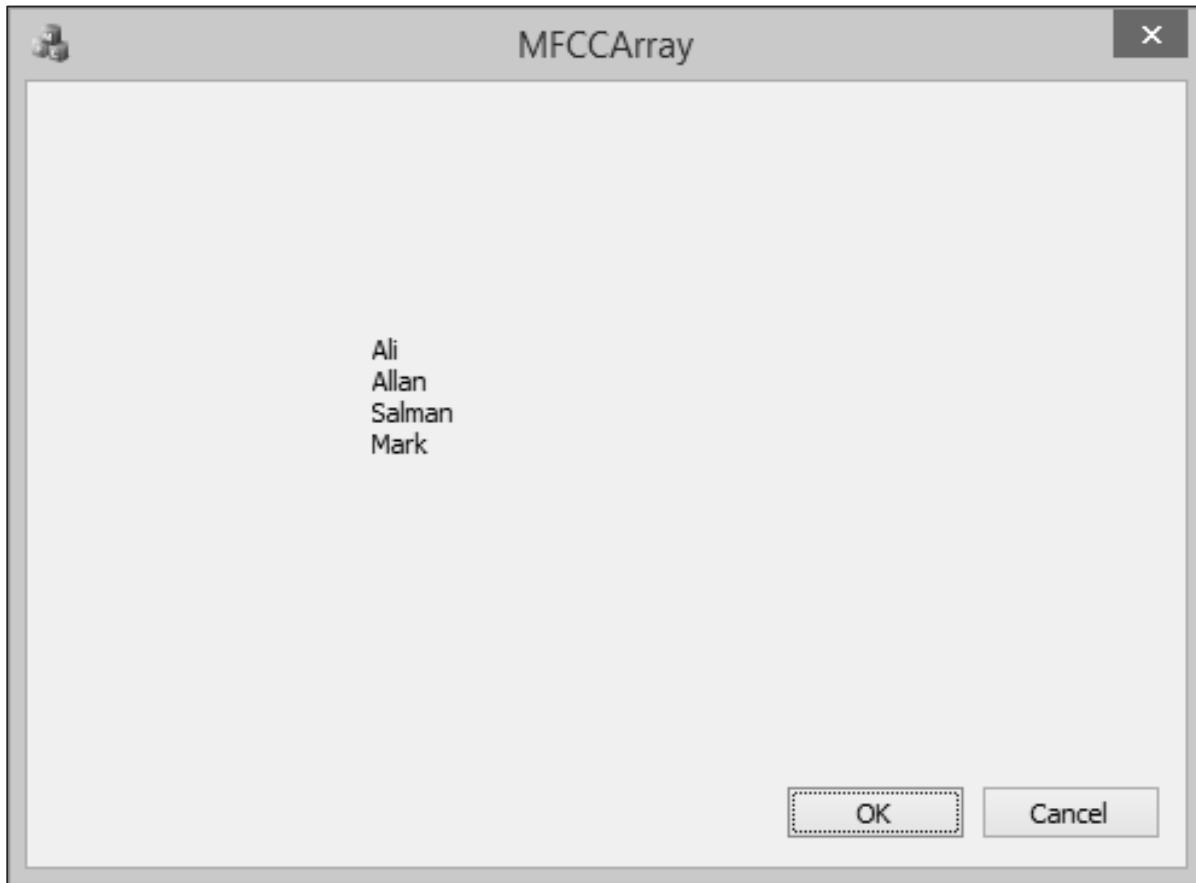
CArray<CString, CString> strArray2;
strArray2.Copy(strArray);
//Retrive names from CArray
for (int i = 0; i < strArray2.GetSize(); i++)
{
    m_strText.Append(strArray2.GetAt(i) + L"\n");
}

UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}

```

You can now see that we have retrieved element from the 2<sup>nd</sup> array and the output is the same because we have used the copy function.



## Remove Items

To remove any particular item, you can use CArray::RemoveAt() function. To remove all the element from the list, CArray::RemoveAll() function can be used.

Let us remove the second element from an array.

```
BOOL CMFCCArrayDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    CArray<CString, CString> strArray;

    //Add names to CArray
```

```
strArray.Add(L"Ali");
strArray.Add(L"Ahmed");
strArray.Add(L"Mark");

strArray.InsertAt(1, L"Allan");

strArray.SetAt(2, L"Salman");

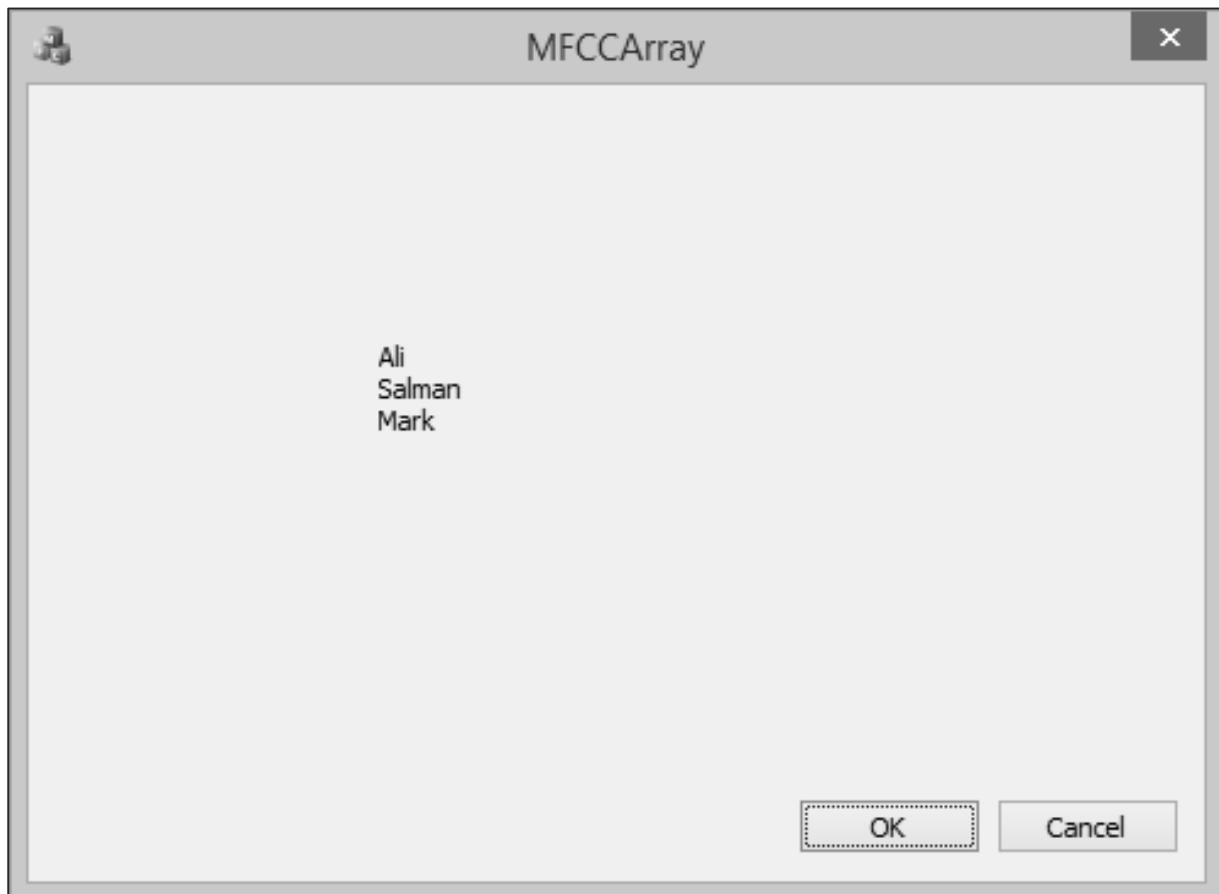
CArray<CString, CString> strArray2;
strArray2.Copy(strArray);

strArray2.RemoveAt(1);

//Retrive names from CArray
for (int i = 0; i < strArray2.GetSize(); i++)
{
    m_strText.Append(strArray2.GetAt(i) + L"\n");
}

UpdateData(FALSE);
return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed, you will see the following output. You can now see that the name Allan is no longer part of the array.



# 19. MFC - Linked Lists

A **linked list** is a linear data structure where each element is a separate object. Each element (we will call it a node) of a list comprises two items — the data and a reference to the next node. The last node has a reference to null.

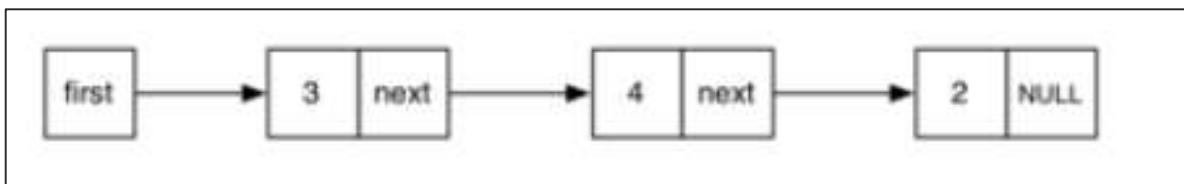
A linked list is a data structure consisting of a group of nodes which together represent a sequence. It is a way to store data with structures so that the programmer can automatically create a new place to store data whenever necessary. Some of its salient features are:

- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link.
- Each item in the list is called a node.
- If the list contains at least one node, then a new node is positioned as the last element in the list.
- If the list has only one node, that node represents the first and the last item.

There are two types of link list:

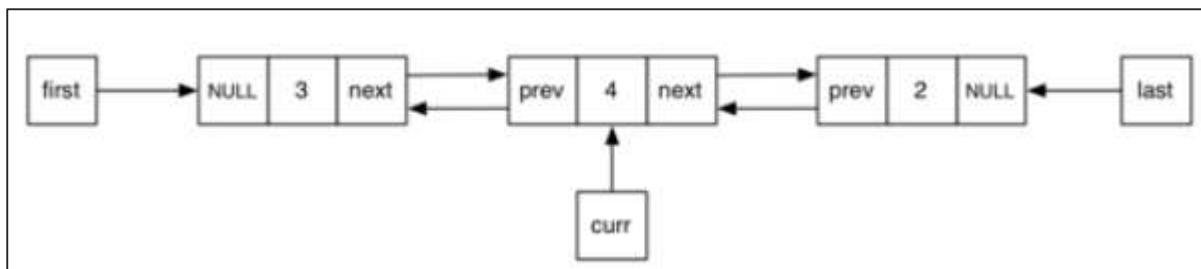
## Singly Linked List

Singly Linked Lists are a type of data structure. In a singly linked list, each node in the list stores the contents of the node and a pointer or reference to the next node in the list.



## Doubly Linked List

A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields that are references to the previous and to the next node in the sequence of nodes.



## CList Class

MFC provides a class **CList** which is a template linked list implementation and works perfectly. CList lists behave like doubly-linked lists. A variable of type POSITION is a key for the list. You can use a POSITION variable as an iterator to traverse a list sequentially and as a bookmark to hold a place.

Here is the list of methods in CList class.

Name	Description
<b>AddHead</b>	Adds an element (or all the elements in another list) to the head of the list (makes a new head).
<b>AddTail</b>	Adds an element (or all the elements in another list) to the tail of the list (makes a new tail).
<b>Find</b>	Gets the position of an element specified by pointer value.
<b>FindIndex</b>	Gets the position of an element specified by a zero-based index.
<b>GetAt</b>	Gets the element at a given position.
<b>GetCount</b>	Returns the number of elements in this list.
<b>GetHead</b>	Returns the head element of the list (cannot be empty).
<b>GetHeadPosition</b>	Returns the position of the head element of the list.
<b>GetNext</b>	Gets the next element for iterating.
<b>GetPrev</b>	Gets the previous element for iterating.
<b>GetSize</b>	Returns the number of elements in this list.
<b>GetTail</b>	Returns the tail element of the list (cannot be empty).
<b>GetTailPosition</b>	Returns the position of the tail element of the list.
<b>InsertAfter</b>	Inserts a new element after a given position.
<b>InsertBefore</b>	Inserts a new element before a given position.
<b>IsEmpty</b>	Tests for the empty list condition (no elements).
<b>RemoveAll</b>	Removes all the elements from this list.
<b>RemoveAt</b>	Removes an element from this list, specified by position.
<b>RemoveHead</b>	Removes the element from the head of the list.
<b>RemoveTail</b>	Removes the element from the tail of the list.
<b>SetAt</b>	Sets the element at a given position.

Following are the different operations on CList objects:

## Create CList Object

---

To create a collection of CList values or objects, you must first decide the type of values of the collection. You can use one of the existing primitive data types such as int, CString, double etc. as shown below in the following code.

```
CList<double, double> m_list;
```

## Add items

---

To add an item, you can use CList::AddTail() function. It adds an item at the end of the list. To add an element at the start of the list, you can use the CList::AddHead() function. In the OnInitDialog() CList, object is created and four values are added as shown in the following code.

```
CList<double, double> m_list;

//Add items to the list
m_list.AddTail(100.75);
m_list.AddTail(85.26);
m_list.AddTail(95.78);
m_list.AddTail(90.1);
```

## Retrieve Items

---

A variable of type POSITION is a key for the list. You can use a POSITION variable as an iterator to traverse a list sequentially.

**Step 1:** To retrieve the element from the list, we can use the following code which will retrieve all the values.

```
//iterate the list
POSITION pos = m_list.GetHeadPosition();
while (pos)
{
    double nData = m_list.GetNext(pos);
    CString strVal;
    strVal.Format(L"%2f\n", nData);
    m_strText.Append(strVal);}
```

**Step 2:** Here is the complete CMFCCListDemoDlg::OnInitDialog() function.

```
BOOL CMFCCListDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

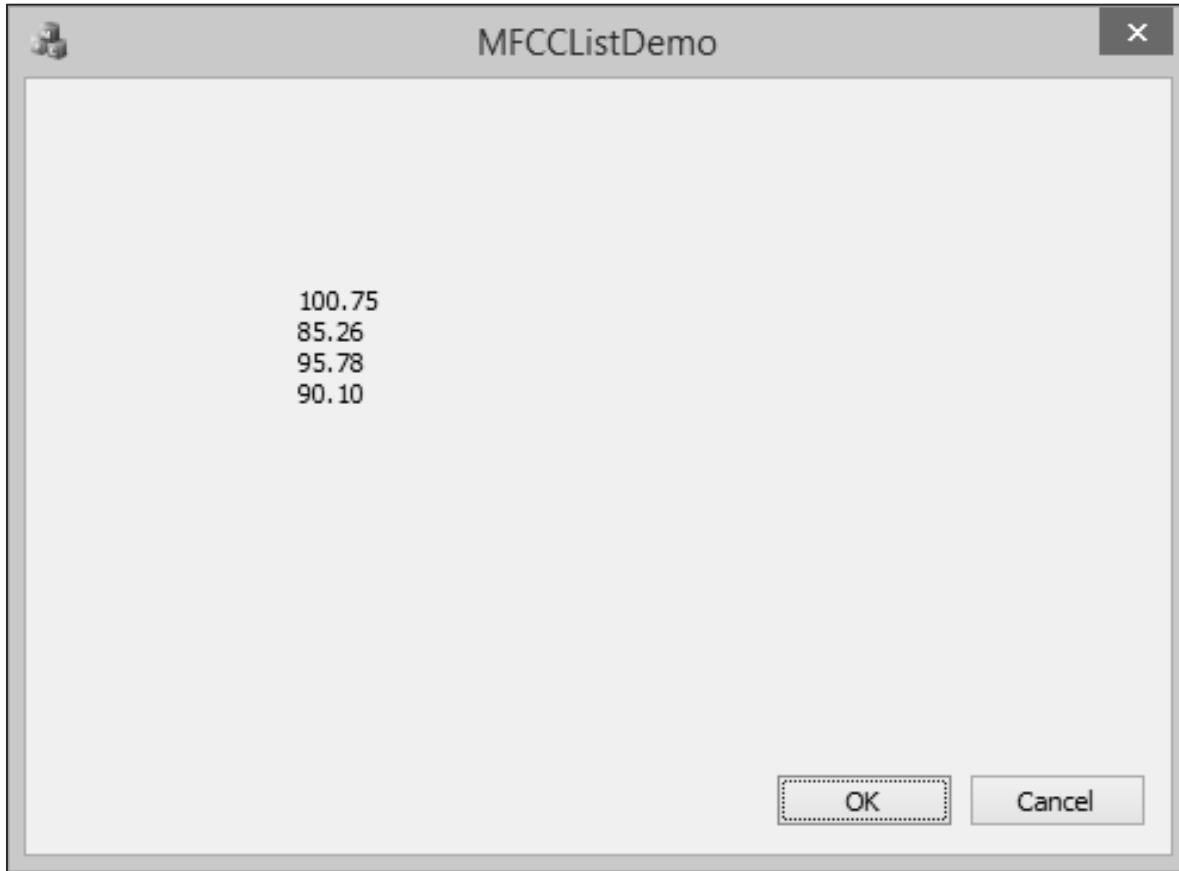
    // TODO: Add extra initialization here
    CList<double, double> m_list;

    //Add items to the list
    m_list.AddTail(100.75);
    m_list.AddTail(85.26);
    m_list.AddTail(95.78);
    m_list.AddTail(90.1);

    //iterate the list
    POSITION pos = m_list.GetHeadPosition();
    while (pos)
    {
        double nData = m_list.GetNext(pos);
        CString strVal;
        strVal.Format(L"%.\f\n", nData);
        m_strText.Append(strVal);
    }
    UpdateData(FALSE);

    return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 3:** When the above code is compiled and executed, you will see the following output.



## Add Items in the Middle

To add item in the middle of the list, you can use the `CList::InsertAfter()` and `CList::InsertBefore()` functions. It takes two parameters — First, the position (where it can be added) and Second, the value.

**Step 1:** Let us insert a new item as shown in the following code.

```
BOOL CMFCCLListDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
}
```

```
CList<double, double> m_list;

//Add items to the list
m_list.AddTail(100.75);
m_list.AddTail(85.26);
m_list.AddTail(95.78);
m_list.AddTail(90.1);

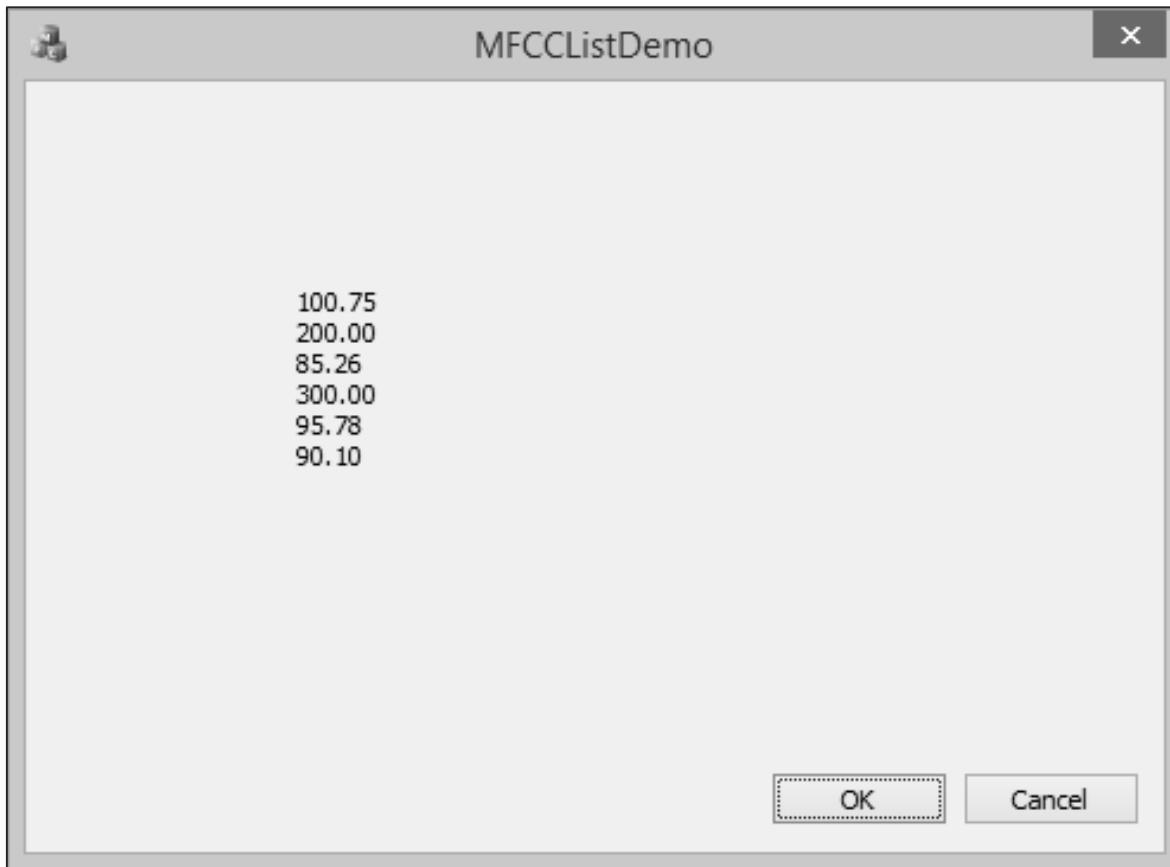
POSITION position = m_list.Find(85.26);
m_list.InsertBefore(position, 200.0);
m_list.InsertAfter(position, 300.0);

//iterate the list
POSITION pos = m_list.GetHeadPosition();
while (pos)
{
    double nData = m_list.GetNext(pos);
    CString strVal;
    strVal.Format(L"%#.2f\n", nData);
    m_strText.Append(strVal);
}
UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

**Step 2:** You can now see that we first retrieved the position of value 85.26 and then inserted one element before and one element after that value.

**Step 3:** When the above code is compiled and executed, you will see the following output.



## Update Item Value

To update item at the middle of array, you can use the CArray::SetAt() function. It takes two parameters — First, the position and Second, the value.

Let us update the 300.00 to 400 in the list as shown in the following code.

```

BOOL CMFCCLListDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // TODO: Add extra initialization here
    CList<double, double> m_list;
}

```

```
//Add items to the list
m_list.AddTail(100.75);
m_list.AddTail(85.26);
m_list.AddTail(95.78);
m_list.AddTail(90.1);

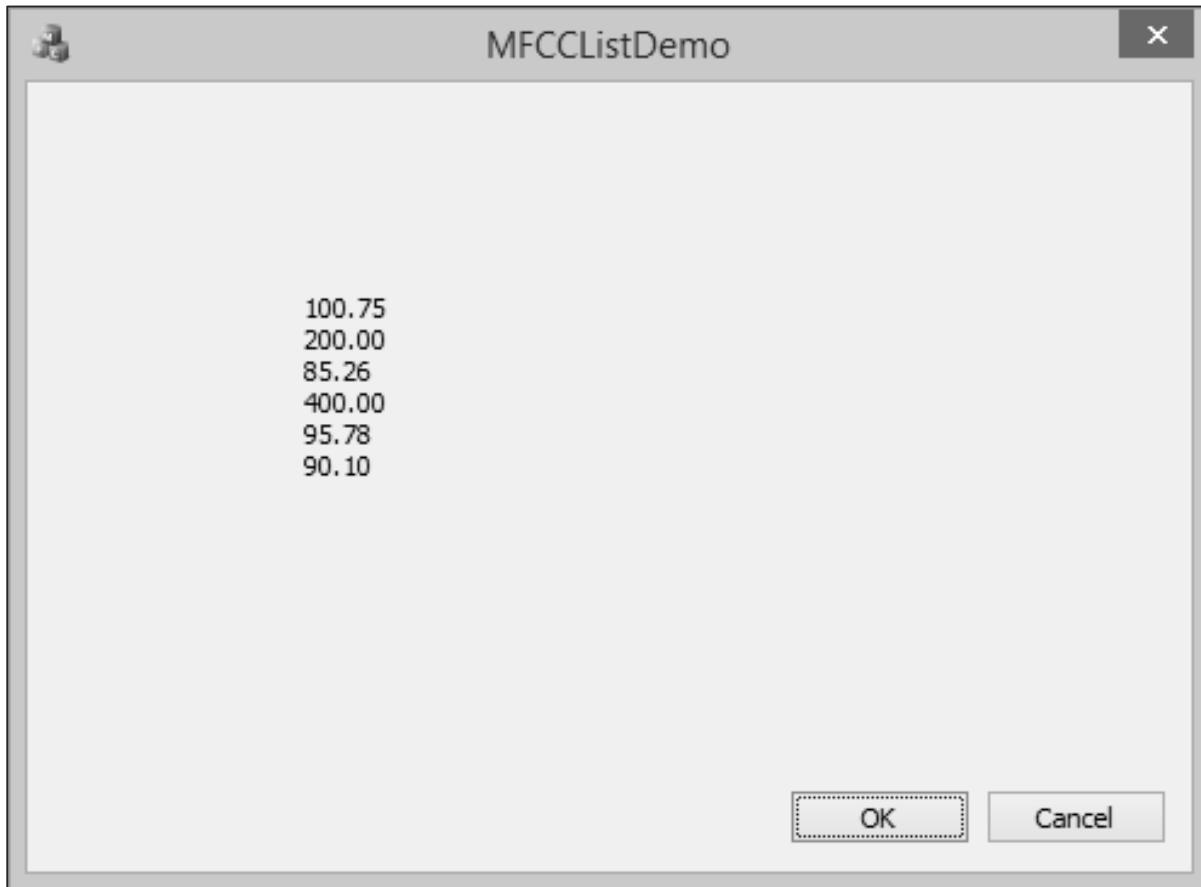
POSITION position = m_list.Find(85.26);
m_list.InsertBefore(position, 200.0);
m_list.InsertAfter(position, 300.0);

position = m_list.Find(300.00);
m_list.SetAt(position, 400.00);

//iterate the list
POSITION pos = m_list.GetHeadPosition();
while (pos)
{
    double nData = m_list.GetNext(pos);
    CString strVal;
    strVal.Format(L"%2f\n", nData);
    m_strText.Append(strVal);
}
UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed, you will see the following output. You can now see that the value of 300.00 is updated to 400.00.



## Remove Items

To remove any particular item, you can use `CList::RemoveAt()` function. To remove all the element from the list, `CList::RemoveAll()` function can be used.

Let us remove the element, which has 95.78 as its value.

```

BOOL CMFCCLListDemoDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // TODO: Add extra initialization here
    CList<double, double> m_list;
  
```

330

```
//Add items to the list
m_list.AddTail(100.75);
m_list.AddTail(85.26);
m_list.AddTail(95.78);
m_list.AddTail(90.1);

POSITION position = m_list.Find(85.26);
m_list.InsertBefore(position, 200.0);
m_list.InsertAfter(position, 300.0);

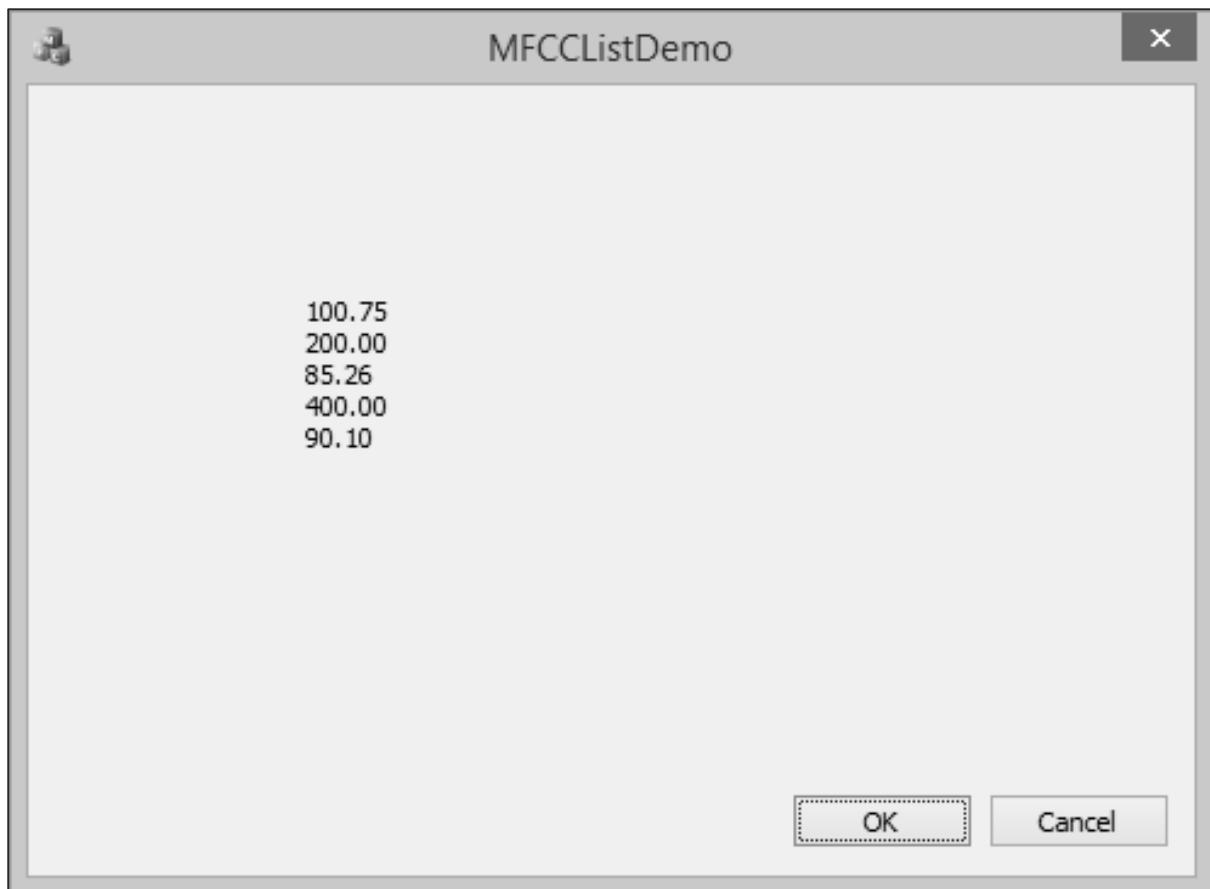
position = m_list.Find(300.00);
m_list.SetAt(position, 400.00);

position = m_list.Find(95.78);
m_list.RemoveAt(position);

//iterate the list
POSITION pos = m_list.GetHeadPosition();
while (pos)
{
    double nData = m_list.GetNext(pos);
    CString strVal;
    strVal.Format(L"%#.2f\n", nData);
    m_strText.Append(strVal);
}
UpdateData(FALSE);

return TRUE; // return TRUE unless you set the focus to a control
}
```

When the above code is compiled and executed, you will see the following output. You can now see that the value of 95.78 is no longer part of the list.



# 20. MFC - Database Classes

A **database** is a collection of information that is organized so that it can easily be accessed, managed, and updated. The MFC database classes based on ODBC are designed to provide access to any database for which an ODBC driver is available. Because the classes use ODBC, your application can access data in many different data formats and different local/remote configurations.

You do not have to write special-case code to handle different database management systems (DBMSs). As long as your users have an appropriate ODBC driver for the data they want to access, they can use your program to manipulate data in tables stored there. A data source is a specific instance of data hosted by some database management system (DBMS). Examples include Microsoft SQL Server, Microsoft Access, etc.

## CDatabase

MFC provides a class **CDatabase** which represents a connection to a data source, through which you can operate on the data source. You can have one or more CDatabase objects active at a time in your application.

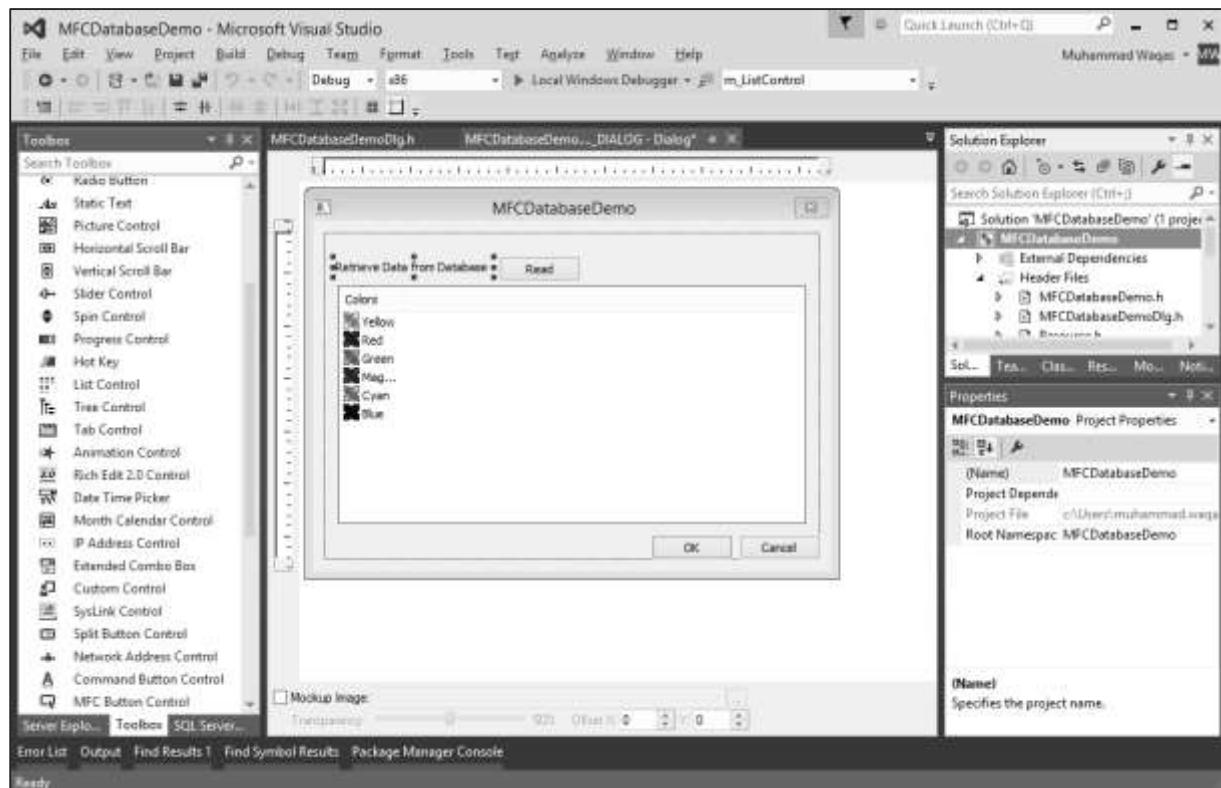
Here is the list of methods in CDatabase class.

Name	Description
<b>BeginTrans</b>	Starts a "transaction" — a series of reversible calls to the AddNew, Edit, Delete, and Update member functions of class <b>CRecordset</b> — on the connected data source. The data source must support transactions for <b>BeginTrans</b> to have any effect.
<b>BindParameters</b>	Allows you to bind parameters before calling <b>ExecuteSQL</b> .
<b>Cancel</b>	Cancels an asynchronous operation or a process from a second thread.
<b>CanTransact</b>	Returns nonzero if the data source supports transactions.
<b>CanUpdate</b>	Returns nonzero if the <b>CDatabase</b> object is updatable (not read-only).
<b>Close</b>	Closes the data source connection.
<b>CommitTrans</b>	Completes a transaction begun by BeginTrans. Commands in the transaction that alter the data source are carried out.
<b>ExecuteSQL</b>	Executes a SQL statement. No data records are returned.
<b>GetBookmarkPersistence</b>	Identifies the operations through which bookmarks persist on recordset objects.
<b>GetConnect</b>	Returns the ODBC connection string used to connect the CDatabase object to a data source.
<b>GetCursorCommitBehavior</b>	Identifies the effect of committing a transaction on an open recordset object.
<b>GetCursorRollbackBehavior</b>	Identifies the effect of rolling back a transaction on an open recordset object.
<b>GetDatabaseName</b>	Returns the name of the database currently in use.
<b>IsOpen</b>	Returns nonzero if the <b>CDatabase</b> object is currently connected to a data source.

<b>OnSetOptions</b>	Called by the framework to set standard connection options. The default implementation sets the query timeout value. You can establish these options ahead of time by calling <b>SetQueryTimeout</b> .
<b>Open</b>	Establishes a connection to a data source (through an ODBC driver).
<b>OpenEx</b>	Establishes a connection to a data source (through an ODBC driver).
<b>Rollback</b>	Reverses changes made during the current transaction. The data source returns to its previous state, as defined at the BeginTrans call, unaltered.
<b>SetLoginTimeout</b>	Sets the number of seconds after which a data source connection attempt will time out.
<b>SetQueryTimeout</b>	Sets the number of seconds after which database query operations will time out. Affects all subsequent recordset Open, AddNew, Edit, and Delete calls.

Let us look into a simple example by creating a new MFC dialog based application.

**Step 1:** Change the caption of TODO line to **Retrieve Data from Database** and drag one button and one List control as shown in the following snapshot.



**Step 2:** Add click event handler for button and control variable `m_ListControl` for List Control.

**Step 3:** We have simple database which contains one Employees table with some records as shown in the following snapshot.

ID	Name	Age	Click to Add
1	Mark	29	
2	Allan	33	
3	Ali	40	
4	Ahmed	37	
*	(New)	0	

**Step 4:** We need to include the following headers file so that we can use CDatabase class.

```
#include "odbcinst.h"
#include "afxdb.h"
```

## Insert Query

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

To add new records, we will use the ExecuteSQL() function of CDatabase class as shown in the following code.

```
CDatabase database;
CString SqlString;
CString strID, strName, strAge;
CString sDriver = L"MICROSOFT ACCESS DRIVER (*.mdb)";
CString sDsn;
CString sFile = L"D:\\Test.mdb";
```

```
// You must change above path if it's different
int iRec = 0;

// Build ODBC connection string
sDsn.Format(L"ODBC;DRIVER=%s;DSN='';DBQ=%s", sDriver, sFile);

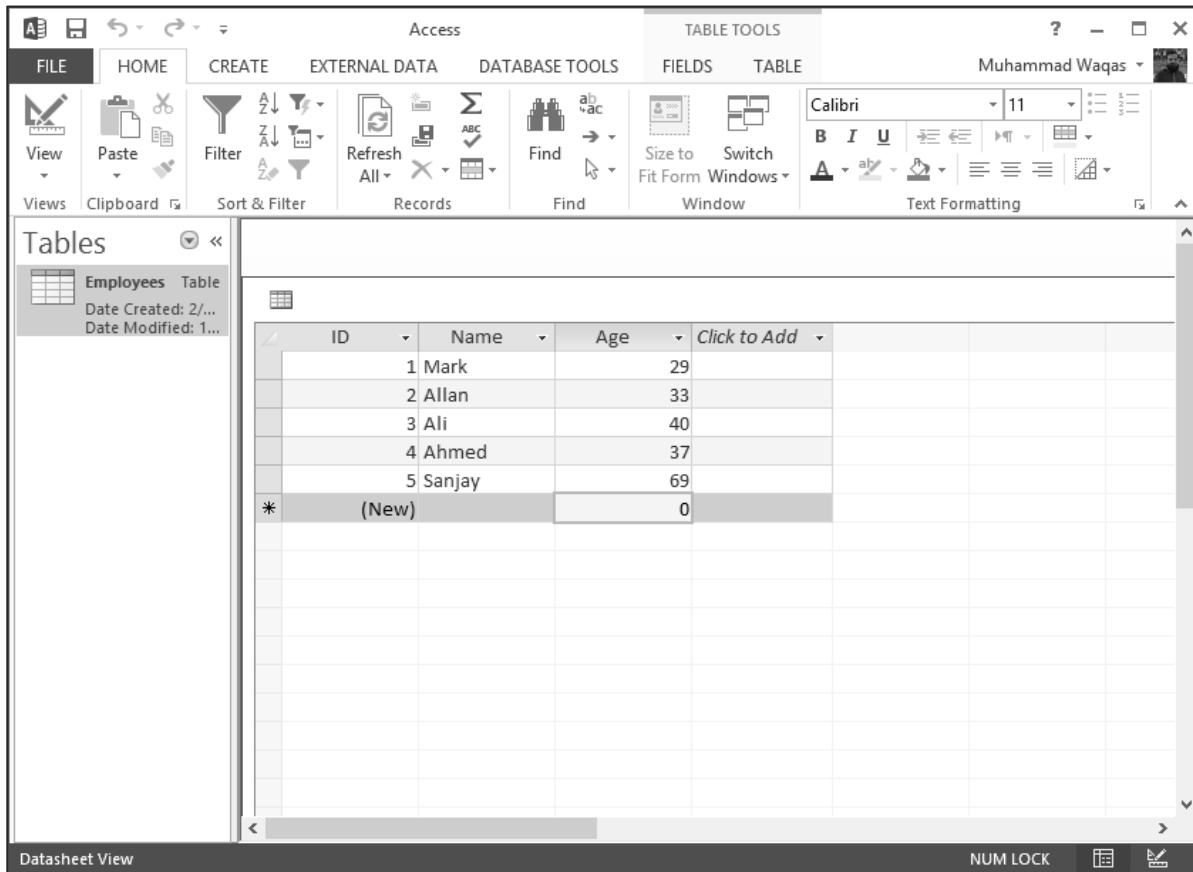
TRY
{
    // Open the database
    database.Open(NULL, false, false, sDsn);

    SqlString = "INSERT INTO Employees (ID,Name,age) VALUES (5,'Sanjay',69)";
    database.ExecuteSQL(SqlString);
    // Close the database
    database.Close();
}

CATCH(CDBException, e)
{
    // If a database exception occurred, show error msg
    AfxMessageBox(L"Database error: " + e->m_strError);
}

END_CATCH;
```

**Step 2:** When the above code is compiled and executed, you will see that a new record is added in your database.



## Retrieve Record

To retrieve the above table in MFC application, we implement the database related operations in the button event handler as shown in the following steps.

**Step 1:** To use CDatabase, construct a CDatabase object and call its Open() function. This will open the connection.

**Step 2:** Construct CRecordset objects for operating on the connected data source, pass the recordset constructor a pointer to your CDatabase object.

**Step 3:** After using the connection, call the Close function and destroy the CDatabase object.

```
void CMFCDatabaseDemoDlg::OnBnClickedButtonRead()
{
    // TODO: Add your control notification handler code here
    CDatabase database;
    CString SqlString;
    CString strID, strName, strAge;
    CString sDriver = "MICROSOFT ACCESS DRIVER (*.mdb)";
```

```

CString sDsn;
CString sFile = L"D:\\Test.mdb";
// You must change above path if it's different
int iRec = 0;

// Build ODBC connection string
sDsn.Format("ODBC;DRIVER=%s;DSN='';DBQ=%s",sDriver,sFile);
TRY
{
    // Open the database
    database.Open(NULL,false,false,sDsn);

    // Allocate the recordset
    CRecordset recset( &database );

    // Build the SQL statement
    SqlString = "SELECT ID, Name, Age "
                "FROM Employees";

    // Execute the query

    recset.Open(CRecordset::forwardOnly,SqlString,CRecordset::readOnly);
    // Reset List control if there is any data
    ResetListControl();
    // populate Grids
    ListView_SetExtendedListViewStyle(m_ListControl,LVS_EX_GRIDLINES);

    // Column width and heading
    m_ListControl.InsertColumn(0,"Emp ID",LVCFMT_LEFT,-1,0);
    m_ListControl.InsertColumn(1,"Name",LVCFMT_LEFT,-1,1);
    m_ListControl.InsertColumn(2, "Age", LVCFMT_LEFT, -1, 1);
    m_ListControl.SetColumnWidth(0, 120);
    m_ListControl.SetColumnWidth(1, 200);
    m_ListControl.SetColumnWidth(2, 200);

    // Loop through each record
    while( !recset.IsEOF() )
    {

```

```

        // Copy each column into a variable
        recset.GetFieldValue("ID",strID);
        recset.GetFieldValue("Name",strName);
        recset.GetFieldValue("Age", strAge);

        // Insert values into the list control
        iRec = m_ListControl.InsertItem(0,strID,0);
        m_ListControl.SetItemText(0,1,strName);
        m_ListControl.SetItemText(0, 2, strAge);

        // goto next record
        recset.MoveNext();
    }

    // Close the database
    database.Close();
}

CATCH(CDBException, e)
{
    // If a database exception occurred, show error msg
    AfxMessageBox("Database error: "+e->m_strError);
}

END_CATCH; }

// Reset List control
void CMFCDatabaseDemoDlg::ResetListControl()
{
    m_ListControl.DeleteAllItems();
    int iNbrOfColumns;
    CHeaderCtrl* pHeader = (CHeaderCtrl*)m_ListControl.GetDlgItem(0);
    if (pHeader)
    {
        iNbrOfColumns = pHeader->GetItemCount();
    }
    for (int i = iNbrOfColumns; i >= 0; i--)
    {
        m_ListControl.DeleteColumn(i);
    }
}

```

**Step 4:** Here is the header file.

```
// MFCDatabaseDemoDlg.h : header file
//


#pragma once
#include "afxcmn.h"


// CMFCDatabaseDemoDlg dialog
class CMFCDatabaseDemoDlg : public CDialogEx
{
// Construction
public:
    CMFCDatabaseDemoDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
#ifdef AFX DESIGN TIME
    enum { IDD = IDD_MFCDATABASEDEMO_DIALOG };
#endif

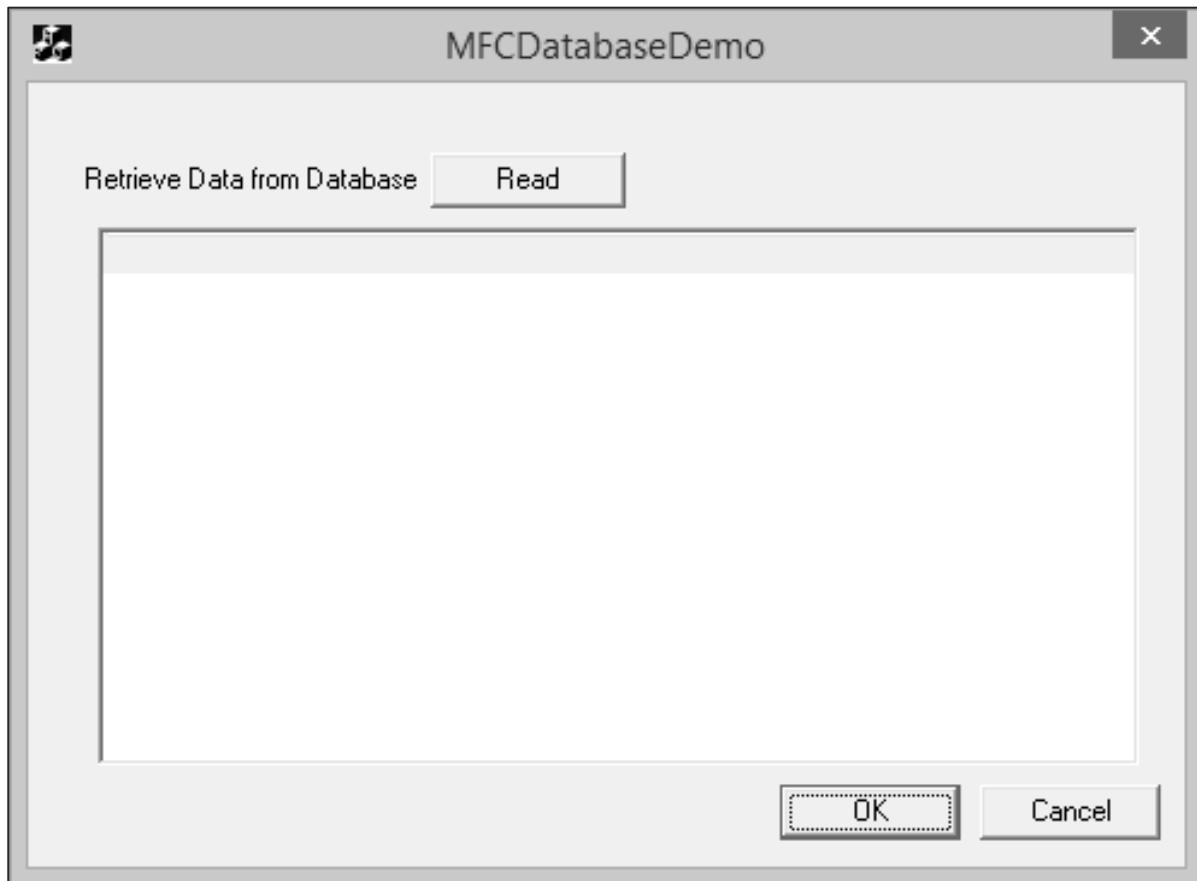
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    void ResetListControl();

// Implementation
protected:
    HICON m_hIcon;

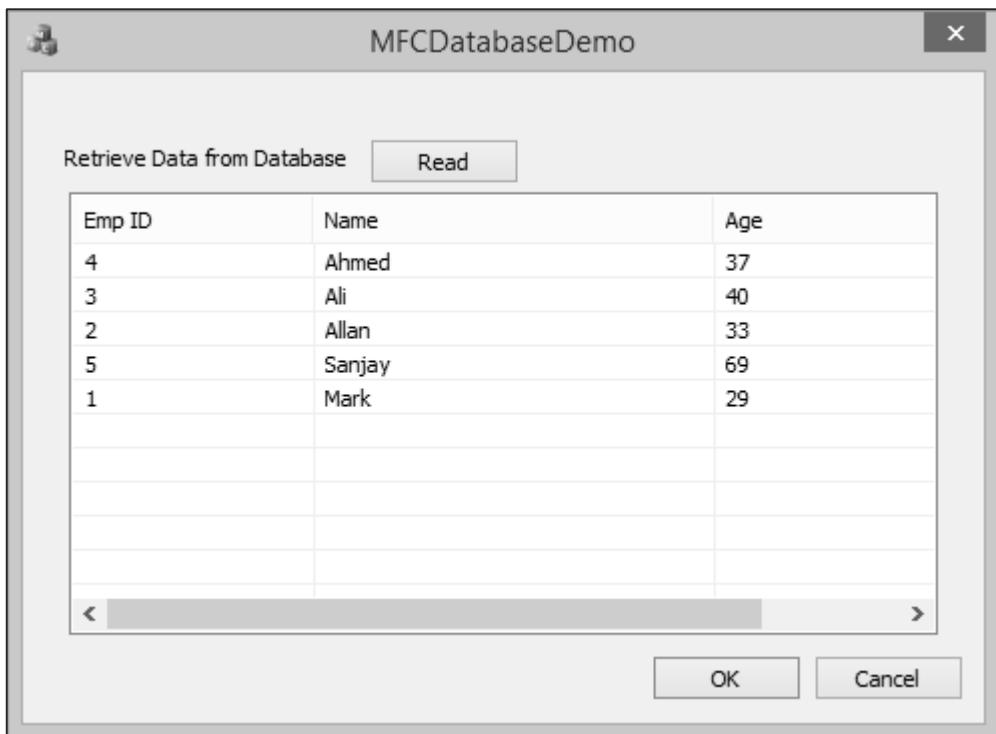
    // Generated message map functions
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    DECLARE_MESSAGE_MAP()
public:
    CListCtrl m_ListControl;
    afx_msg void OnBnClickedButtonRead();
};

}
```

**Step 5:** When the above code is compiled and executed, you will see the following output.



**Step 6:** Press the Read button to execute the database operations. It will retrieve the Employees table.



# Update Record

The SQL UPDATE Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

**Step 1:** Let us look into a simple example by updating the Age where ID is equal to 5.

```
SqlString = L"UPDATE Employees SET Age = 59 WHERE ID = 5;";  
database.ExecuteSQL(SqlString);
```

**Step 2:** Here is the complete code of button click event.

```
void CMFCDatabaseDemoDlg::OnBnClickedButtonRead()
{
    // TODO: Add your control notification handler code here

    CDatabase database;
    CString SqlString;
    CString strID, strName, strAge;
    CString sDriver = L"MICROSOFT ACCESS DRIVER (*.mdb)";
    CString sDsn;
    CString sFile =
L"C:\\\\Users\\\\Muhammad.Waqas\\\\Downloads\\\\Compressed\\\\ReadDB_demo\\\\Test.mdb";
    // You must change above path if it's different
}
```

```

int iRec = 0;

// Build ODBC connection string
sDsn.Format(L"ODBC;DRIVER=%s;DSN='';DBQ=%s", sDriver, sFile);
TRY
{
    // Open the database
    database.Open(NULL, false, false, sDsn);

    // Allocate the recordset
    CRecordset recset(&database);

    SqlString = L"UPDATE Employees SET Age = 59 WHERE ID = 5;";

    database.ExecuteSQL(SqlString);

    SqlString = "SELECT ID, Name, Age FROM Employees";

    // Build the SQL statement
    SqlString = "SELECT ID, Name, Age FROM Employees";

    // Execute the query

    recset.Open(CRecordset::forwardOnly, SqlString, CRecordset::readOnly);

    // Reset List control if there is any data
    ResetListControl();
    // populate Grids
    ListView_SetExtendedListViewStyle(m_listCtrl, LVS_EX_GRIDLINES);

    // Column width and heading
    m_listCtrl.InsertColumn(0, L"Emp ID", LVCFMT_LEFT, -1, 0);
    m_listCtrl.InsertColumn(1, L"Name", LVCFMT_LEFT, -1, 1);
    m_listCtrl.InsertColumn(2, L"Age", LVCFMT_LEFT, -1, 1);
    m_listCtrl.SetColumnWidth(0, 120);
    m_listCtrl.SetColumnWidth(1, 200);
}

```

```
m_listCtrl.SetColumnWidth(2, 200);

// Loop through each record
while (!recset.IsEOF())
{
    // Copy each column into a variable
    recset.GetFieldValue(L"ID",strID);
    recset.GetFieldValue(L"Name",strName);
    recset.GetFieldValue(L"Age", strAge);

    // Insert values into the list control
    iRec = m_listCtrl.InsertItem(0,strID,0);
    m_listCtrl.SetItemText(0,1,strName);
    m_listCtrl.SetItemText(0, 2, strAge);

    // goto next record
    recset.MoveNext();
}

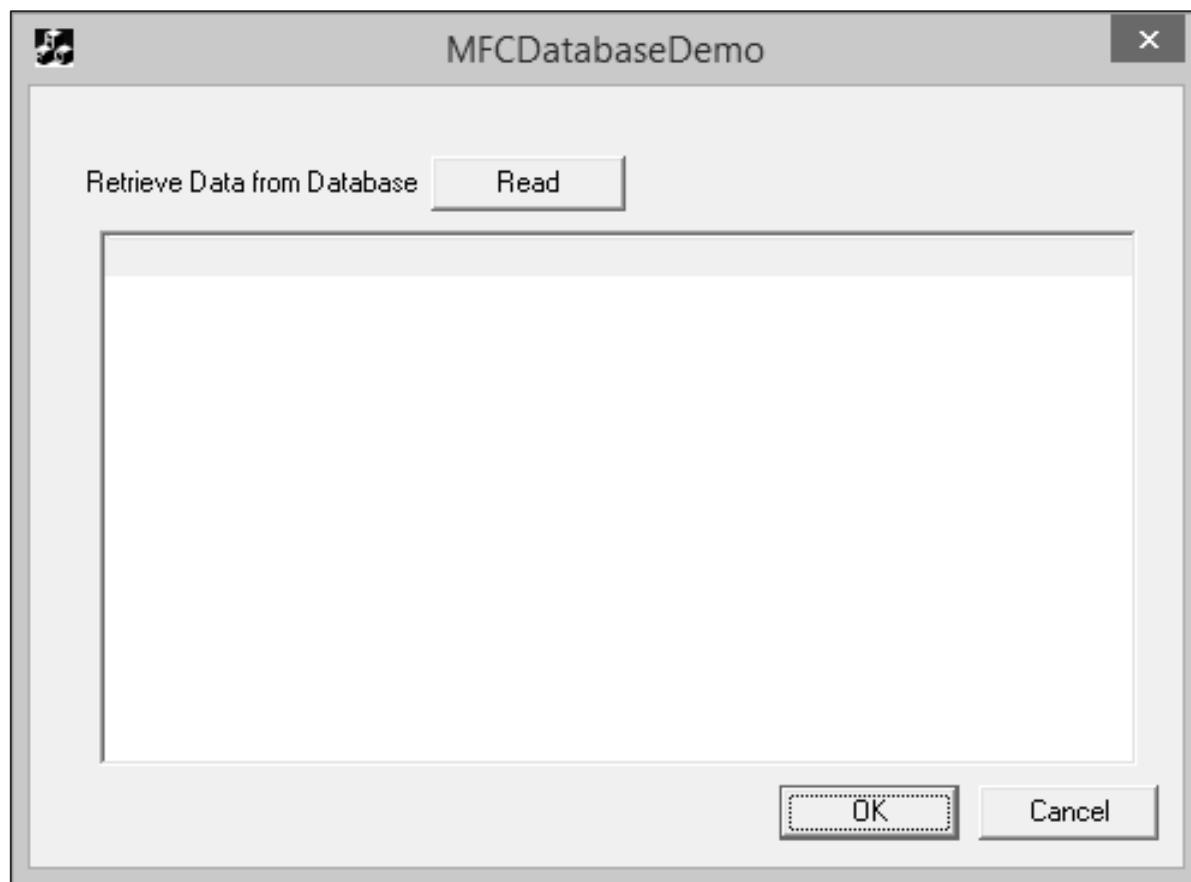
// Close the database
database.Close();
}

CATCH(CDBException, e)
{
    // If a database exception occurred, show error msg
    AfxMessageBox(L"Database error: " + e->m_strError);
}

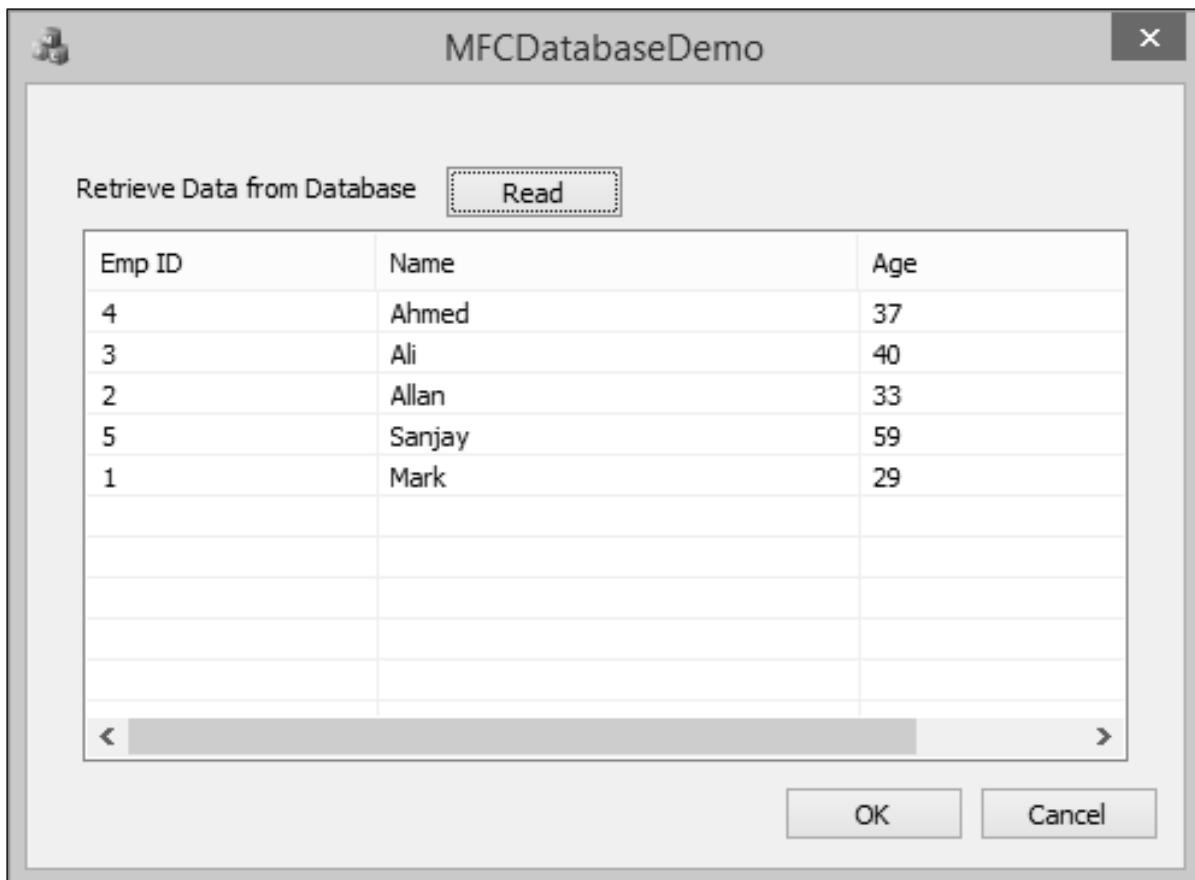
END_CATCH;
}
```

**Step 3:** When the above code is compiled and executed, you will see the following output.

344



**Step 4:** Press the Read button to execute the database operations. It will retrieve the following Employees table.



**Step 5:** You can now see that age is updated from 69 to 59.

## Delete Record

The SQL DELETE Query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

**Step 1:** Let us look into a simple example by deleting the record where ID is equal to 3.

```
SqlString = L"DELETE FROM Employees WHERE ID = 3;";

database.ExecuteSQL(SqlString);
```

**Step 2:** Here is the complete code of button click event.

```
void CMFCDatabaseDemoDlg::OnBnClickedButtonRead()
{
    // TODO: Add your control notification handler code here
    CDatabase database;
    CString SqlString;
```

```

CString strID, strName, strAge;
CString sDriver = L"MICROSOFT ACCESS DRIVER (*.mdb)";
CString sDsn;
CString sFile =
L"C:\\\\Users\\\\Muhammad.Waqas\\\\Downloads\\\\Compressed\\\\ReadDB_demo\\\\Test.mdb";
// You must change above path if it's different
int iRec = 0;

// Build ODBC connection string
sDsn.Format(L"ODBC;DRIVER=%s;DSN='';DBQ=%s", sDriver, sFile);
TRY
{
    // Open the database
    database.Open(NULL, false, false, sDsn);

    // Allocate the recordset
    CRecordset recset(&database);

    SqlString = L"DELETE FROM Employees WHERE ID = 3;";

    database.ExecuteSQL(SqlString);

    SqlString = "SELECT ID, Name, Age FROM Employees";

    // Build the SQL statement
    SqlString = "SELECT ID, Name, Age FROM Employees";

    // Execute the query
    recset.Open(CRecordset::forwardOnly, SqlString, CRecordset::readOnly);

    // Reset List control if there is any data
    ResetListControl();
    // populate Grids
    ListView_SetExtendedListViewStyle(m_listCtrl, LVS_EX_GRIDLINES);
}

```

```

// Column width and heading
m_listCtrl.InsertColumn(0,L"Emp ID",LVCFMT_LEFT,-1,0);
m_listCtrl.InsertColumn(1,L"Name",LVCFMT_LEFT,-1,1);
m_listCtrl.InsertColumn(2, L"Age", LVCFMT_LEFT, -1, 1);
m_listCtrl.SetColumnWidth(0, 120);
m_listCtrl.SetColumnWidth(1, 200);
m_listCtrl.SetColumnWidth(2, 200);

// Loop through each record
while (!recset.IsEOF())
{
    // Copy each column into a variable
    recset.GetFieldValue(L"ID",strID);
    recset.GetFieldValue(L"Name",strName);
    recset.GetFieldValue(L"Age", strAge);

    // Insert values into the list control
    iRec = m_listCtrl.InsertItem(0,strID,0);
    m_listCtrl.SetItemText(0,1,strName);
    m_listCtrl.SetItemText(0, 2, strAge);

    // goto next record
    recset.MoveNext();
}

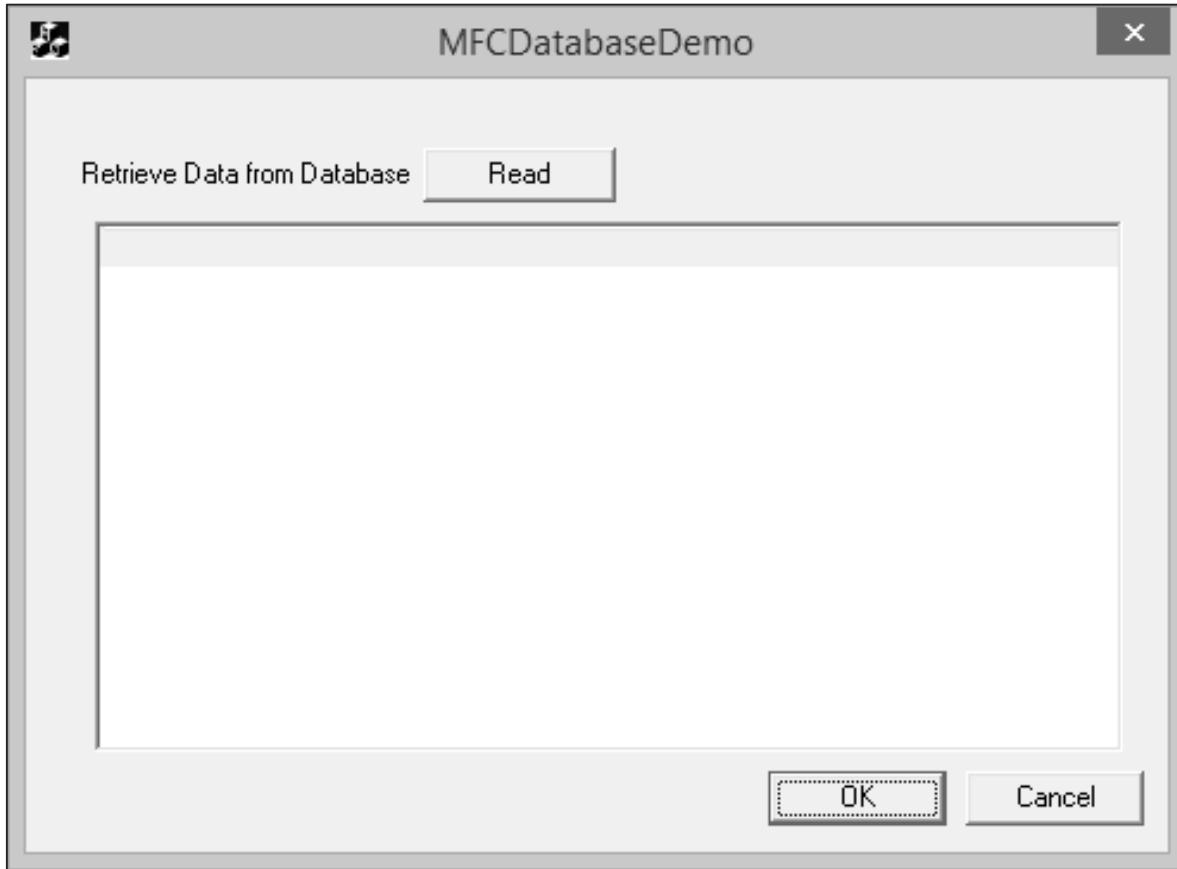
// Close the database
database.Close();
}

CATCH(CDBException, e)
{
    // If a database exception occurred, show error msg
    AfxMessageBox(L"Database error: " + e->m_strError);
}

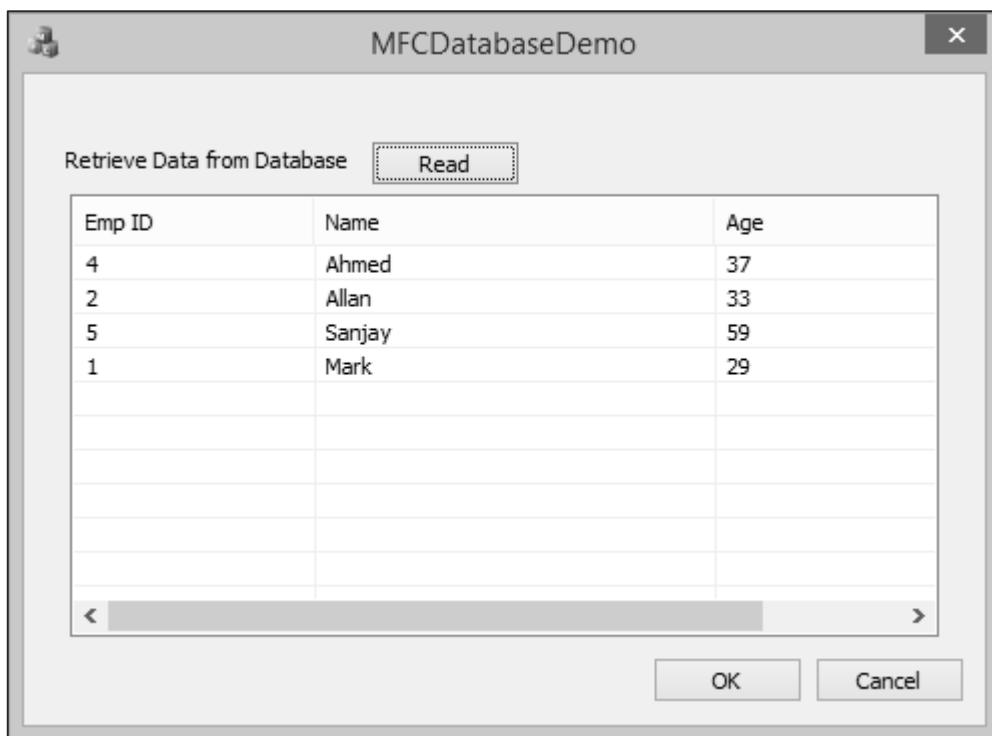
END_CATCH;
}

```

**Step 3:** When the above code is compiled and executed, you will see the following output.



**Step 4:** Press the Read button to execute the database operations. It will retrieve the Employees table.



# 21. MFC - Serialization

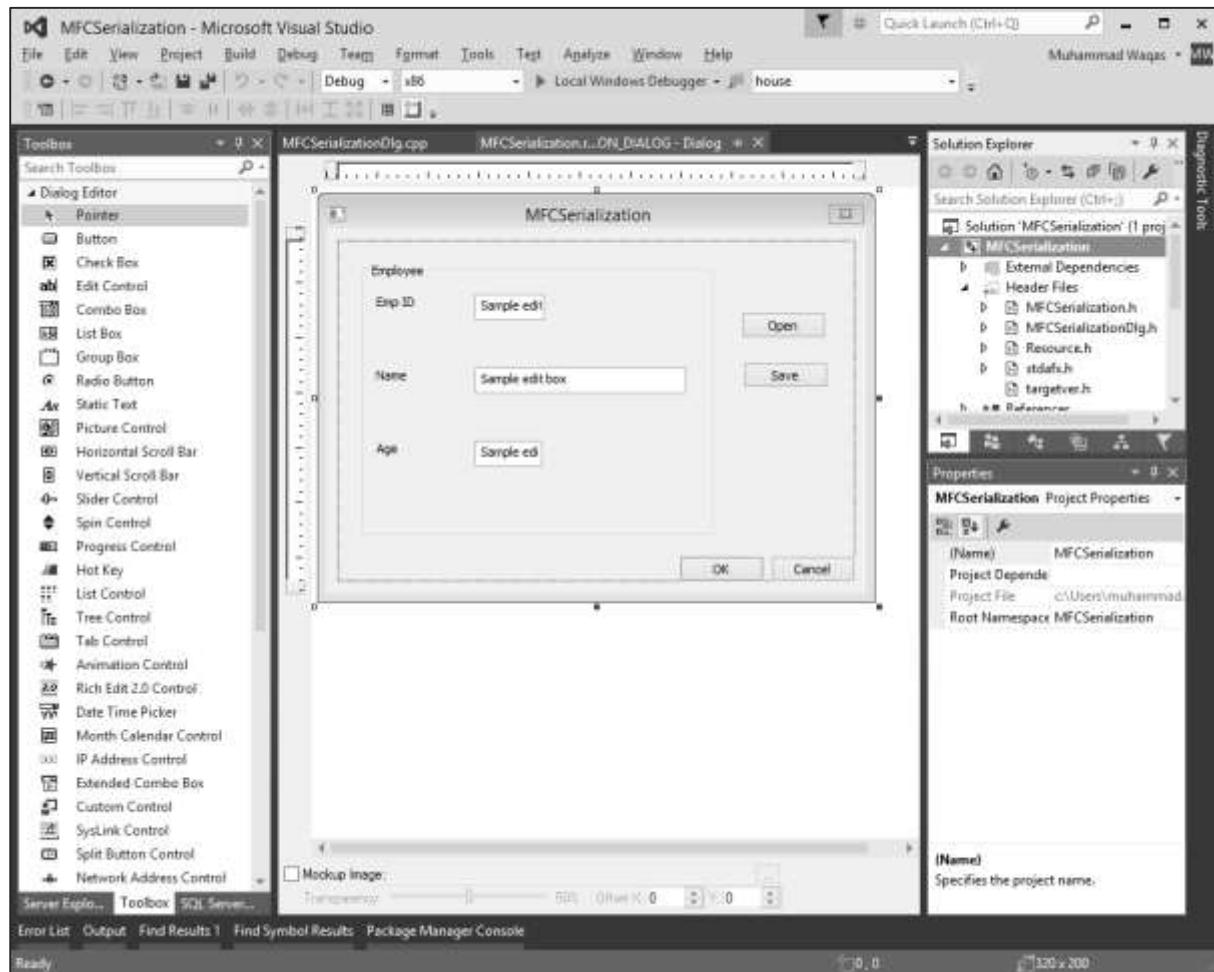
**Serialization** is the process of writing or reading an object to or from a persistent storage medium such as a disk file. Serialization is ideal for situations where it is desired to maintain the state of structured data (such as C++ classes or structures) during or after the execution of a program.

When performing file processing, the values are typically of primitive types (char, short, int, float, or double). In the same way, we can individually save many values, one at a time. This technique doesn't include an object created from (as a variable of) a class

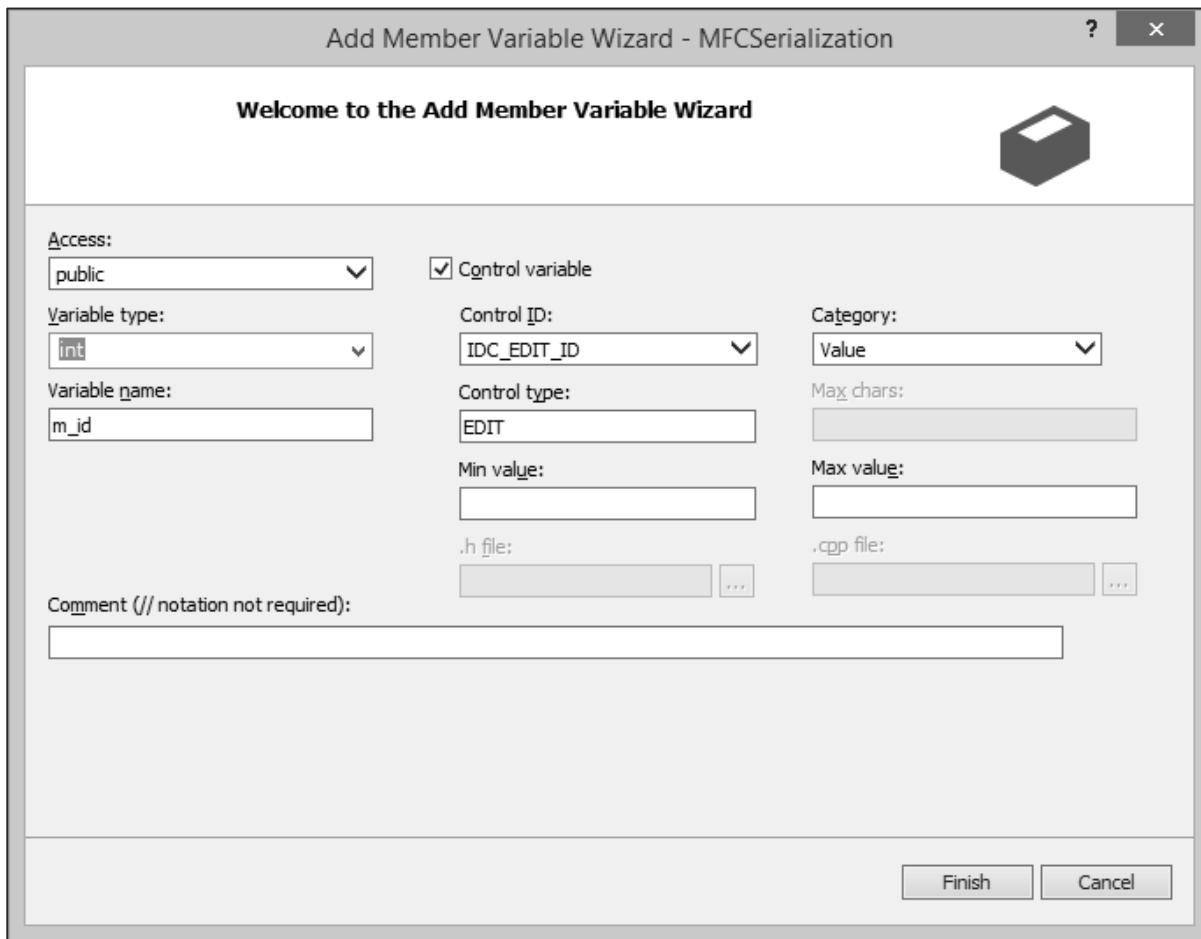
The MFC library has a high level of support for serialization. It starts with the **CObject** class that is the ancestor to most MFC classes, which is equipped with a **Serialize()** member function.

Let us look into a simple example by creating a new MFC project.

**Step 1:** Remove the TODO line and design your dialog box as shown in the following snapshot.



**Step 2:** Add value variables for all the edit controls. For Emp ID and Age mentioned, the value type is an integer as shown in the following snapshot.



**Step 3:** Add the event handler for both the buttons.

**Step 4:** Let us now add a simple Employee class, which we need to serialize. Here is the declaration of Employee class in header file.

```
class CEmployee : public CObject
{public:
    int empID;
    CString empName;
    int age;
    CEmployee(void);
    ~CEmployee(void);

private:

public:
    void Serialize(CArchive& ar);
    DECLARE_SERIAL(CEmployee);};
```

**Step 5:** Here is the definition of Employee class in source (\*.cpp) file.

```
IMPLEMENT_SERIAL(CEmployee, CObject, 0)
CEmployee::CEmployee(void)
{
}

CEmployee::~CEmployee(void)
{
}

void CEmployee::Serialize(CArchive& ar)
{
    CObject::Serialize(ar);

    if (ar.IsStoring())
        ar << empID << empName << age;
    else
        ar >> empID >> empName >> age;
}
```

**Step 6:** Here is the implementation of Save button event handler.

```
void CMFCSerializationDlg::OnBnClickedButtonSave()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    CEmployee employee;
    CFile file;
    file.Open(L"EmployeeInfo.hse", CFile::modeCreate | CFile::modeWrite);
    CArchive ar(&file, CArchive::store);
    employee.empID = m_id;
    employee.empName = m_strName;
    employee.age = m_age;
    employee.Serialize(ar);
    ar.Close();
}
```

**Step 7:** Here is the implementation of Open button event handler.

```
void CMFCSerializationDlg::OnBnClickedButtonOpen()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    CFile file;

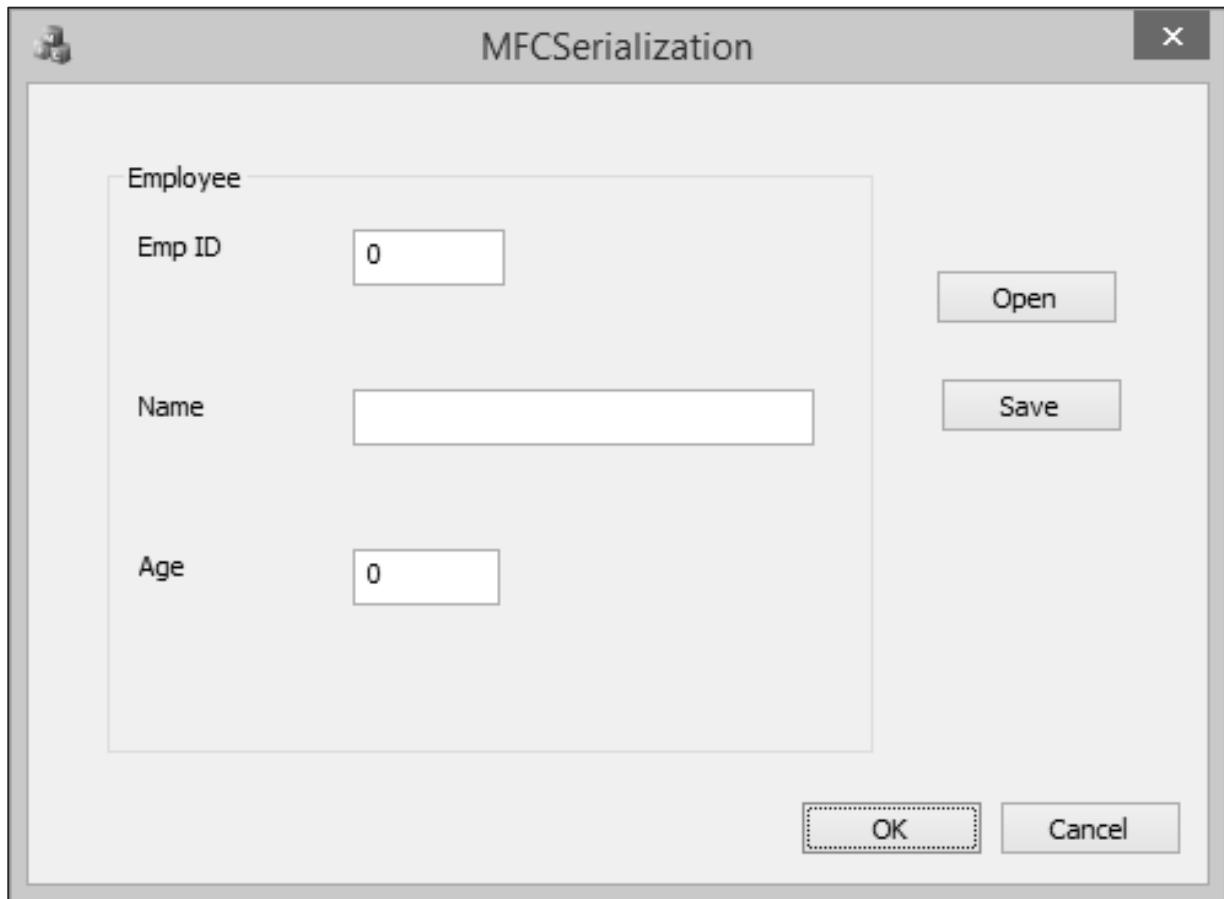
    file.Open(L"EmployeeInfo.hse", CFile::modeRead);
    CArchive ar(&file, CArchive::load);
    CEmployee employee;

    employee.Serialize(ar);

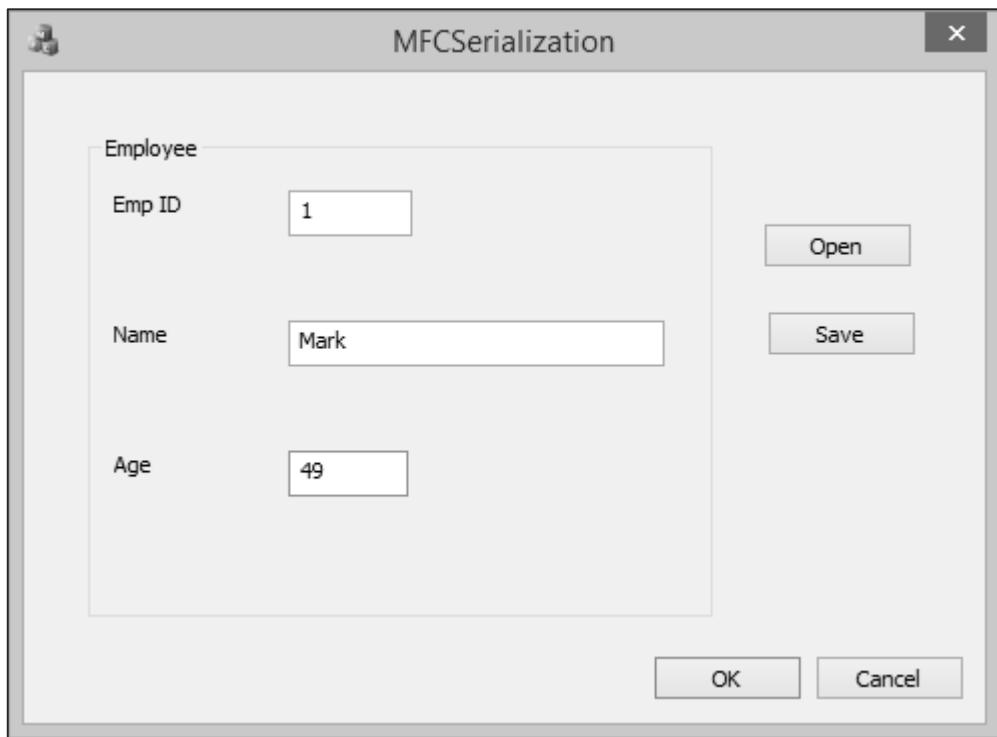
    m_id = employee.empID;
    m_strName = employee.empName;
    m_age = employee.age;
    ar.Close();
    file.Close();

    UpdateData(FALSE);
}
```

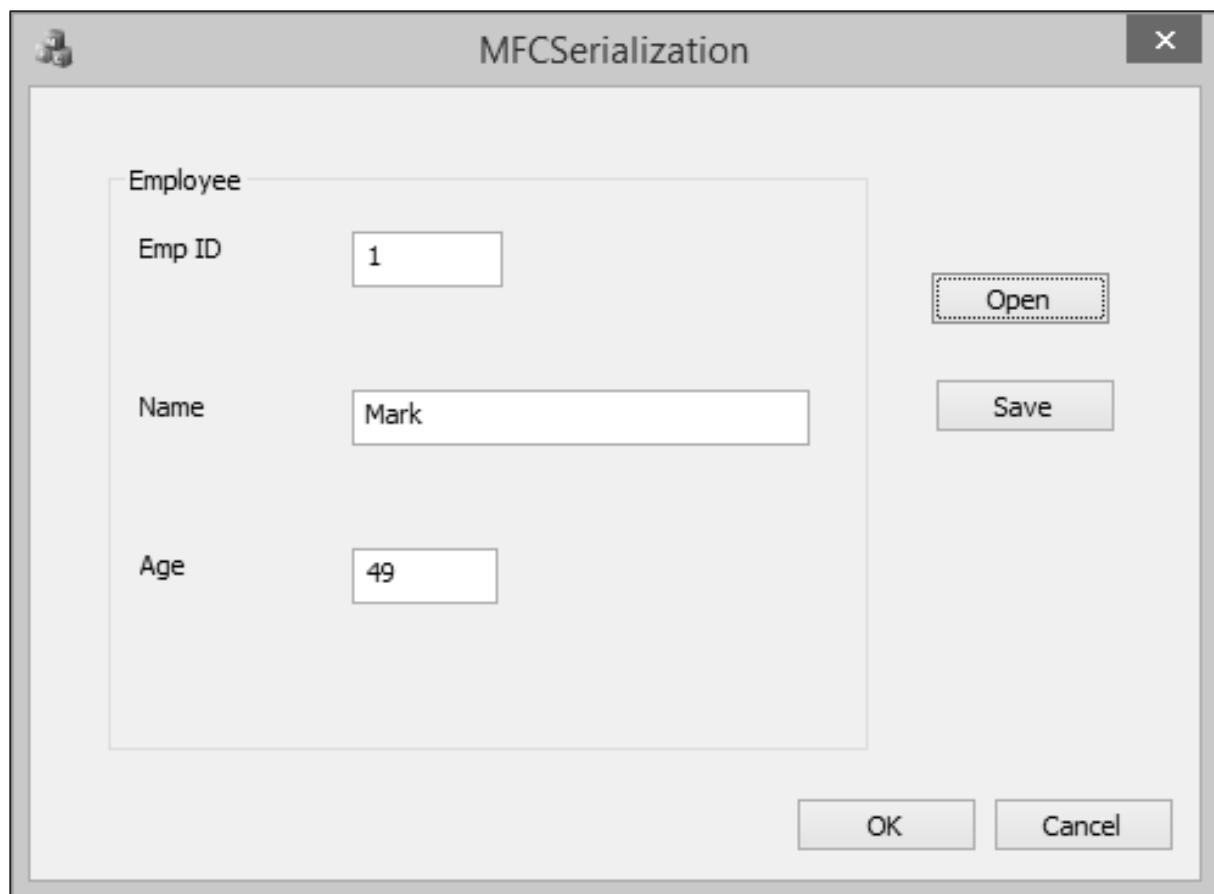
**Step 8:** When the above code is compiled and executed, you will see the following output.



**Step 9:** Enter the info in all the fields and click Save and close this program.



**Step 10:** It will save the data. Run the application again and click open. It will load the Employee information.



## 22. MFC - Multithreading

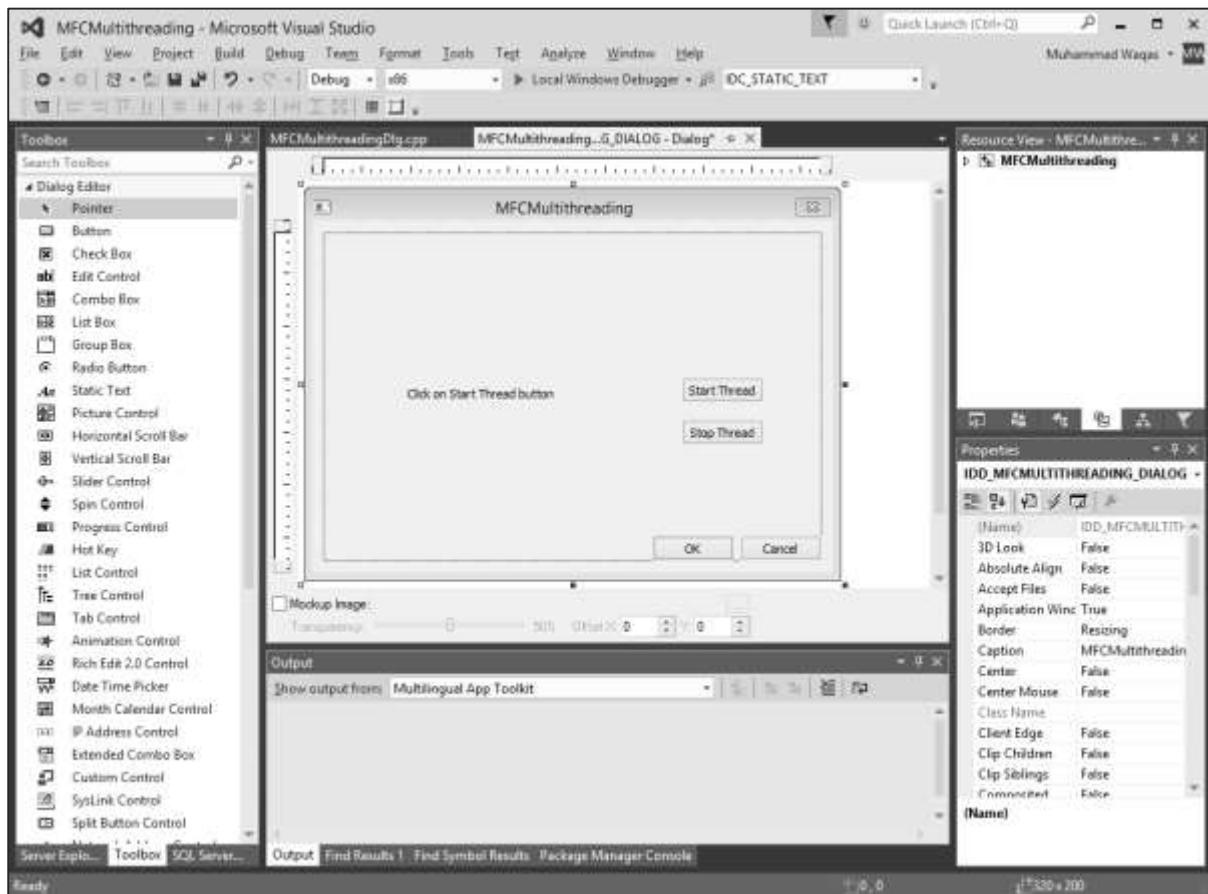
The Microsoft Foundation Class (MFC) library provides support for multithreaded applications. A thread is a path of execution within a process. When you start Notepad, the operating system creates a process and begins executing the primary thread of that process. When this thread terminates, so does the process.

You can create additional threads in your application if you want. All threads in MFC applications are represented by CWinThread objects. In most situations, you do not even have to explicitly create these objects; instead call the framework helper function AfxBeginThread, which creates the CWinThread object for you.

Let us look into a simple example by creating a new MFC dialog based application.

**Step 1:** Change the Caption and ID of Static control to **Click on Start Thread button** and **IDC\_STATIC\_TEXT** respectively.

**Step 2:** Drag two buttons and add click event handlers for these buttons.



**Step 3:** Add control variable for static text control.

**Step 4:** Now add the following three global variables at the start of CMFCMultithreadingDlg.cpp file

```
int currValue;
int maxValue;
BOOL stopNow;
```

**Step 5:** Add the WM\_TIMER message in CMFCMultithreadingDlg class.

Here is the implementation of OnTimer()

```
void CMFCMultithreadingDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    CString sStatusMsg;
    sStatusMsg.Format(L"Running: %d", currValue);
    m_ctrlStatus.SetWindowText(sStatusMsg);

    CDialogEx::OnTimer(nIDEvent);
}
```

**Step 6:** Now add a sample function for using in AfxBeginThread in CMFCMultithreadingDlg class.

```
UINT MyThreadProc(LPVOID Param)
{
    while (!stopNow && (currValue < maxValue))
    {
        currValue++;
        Sleep(50); // would do some work here
    }

    return TRUE;
}
```

**Step 7:** Here is the implementation of event handler for Start Thread button, which will start the thread.

```
void CMFCMultithreadingDlg::OnBnClickedButtonStart()
```

```
{
    // TODO: Add your control notification handler code here
    currValue = 0;
    maxValue = 5000;
    stopNow = 0;
    m_ctrlStatus.SetWindowText(L"Starting...");
    SetTimer(1234, 333, 0); // 3 times per second

    AfxBeginThread(MyThreadProc, 0); // <<== START THE THREAD
}
```

**Step 8:** Here is the implementation of event handler for Stop Thread button, which will stop the thread.

```
void CMFCMultithreadingDlg::OnBnClickedButtonStop()
{
    // TODO: Add your control notification handler code here
    stopNow = TRUE;
    KillTimer(1234);
    m_ctrlStatus.SetWindowText(L"Stopped");
}
```

**Step 9:** Here is the complete source file.

```
// MFCMultithreadingDlg.cpp : implementation file
//

#include "stdafx.h"
#include "MFCMultithreading.h"
#include "MFCMultithreadingDlg.h"
#include "afxdialogex.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif
```

```

// CMFCMultithreadingDlg dialog

int currValue;
int maxValue;
BOOL stopNow;

CMFCMultithreadingDlg::CMFCMultithreadingDlg(CWnd* pParent /*=NULL*/)
    : CDialogEx(IDD_MFCMULTITHREADING_DIALOG, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMFCMultithreadingDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_STATIC_TEXT, m_ctrlStatus);
}

BEGIN_MESSAGE_MAP(CMFCMultithreadingDlg, CDialogEx)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_START,
&CMFCMultithreadingDlg::OnBnClickedButtonStart)
    ON_WM_TIMER()
    ON_BN_CLICKED(IDC_BUTTON_STOP,
&CMFCMultithreadingDlg::OnBnClickedButtonStop)
END_MESSAGE_MAP()

// CMFCMultithreadingDlg message handlers

BOOL CMFCMultithreadingDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
}

```

```
SetIcon(m_hIcon, FALSE);           // Set small icon

// TODO: Add extra initialization here

return TRUE; // return TRUE unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CMFCMultithreadingDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialogEx::OnPaint();
    }
}
```

```

// The system calls this function to obtain the cursor to display while the
// user drags
// the minimized window.
HCURSOR CMFCMultithreadingDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

UINT /*CThreadDlg::* */MyThreadProc(LPVOID Param) //Sample function for using in
AfxBeginThread
{
    while (!stopNow && (currValue < maxValue))
    {
        currValue++;
        Sleep(50); // would do some work here
    }

    return TRUE;
}

void CMFCMultithreadingDlg::OnBnClickedButtonStart()
{
    // TODO: Add your control notification handler code here
    currValue = 0;
    maxValue = 5000;
    stopNow = 0;
    m_ctrlStatus.SetWindowText(L"Starting...");
    SetTimer(1234, 333, 0); // 3 times per second

    AfxBeginThread(MyThreadProc, 0); // <<== START THE THREAD
}

void CMFCMultithreadingDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    CString sStatusMsg;
    sStatusMsg.Format(L"Running: %d", currValue);
}

```

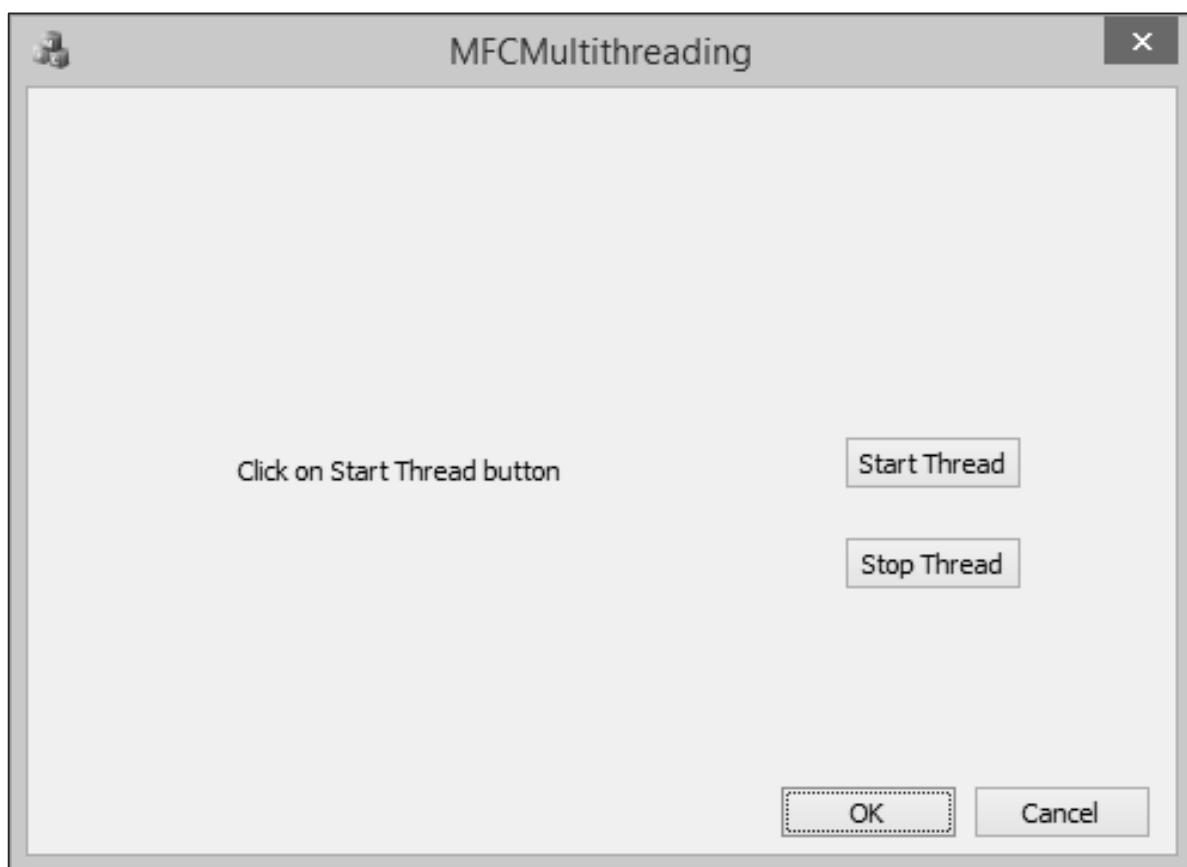
```
m_ctrlStatus.SetWindowText(sStatusMsg);

CDialogEx::OnTimer(nIDEvent);

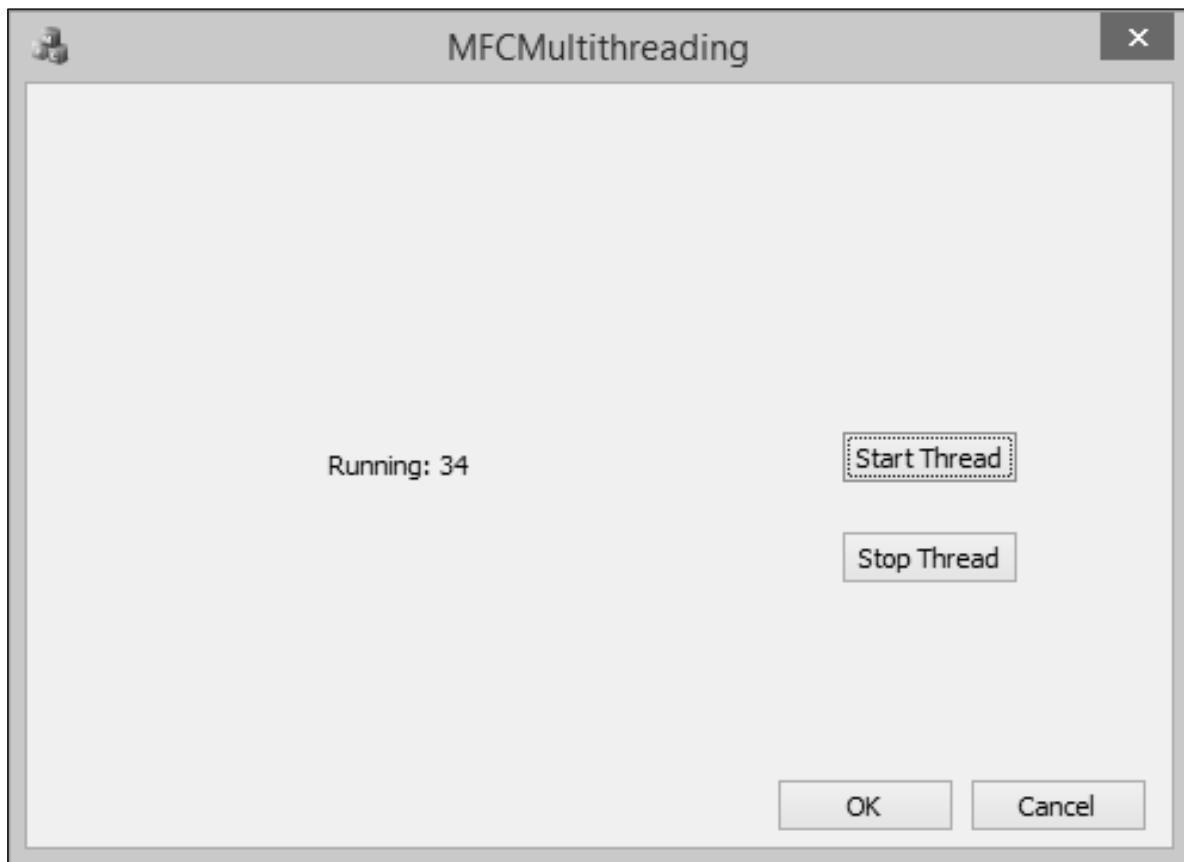
}

void CMFCMultithreadingDlg::OnBnClickedButtonStop()
{
    // TODO: Add your control notification handler code here
    stopNow = TRUE;
    KillTimer(1234);
    m_ctrlStatus.SetWindowText(L"Stopped");
}
```

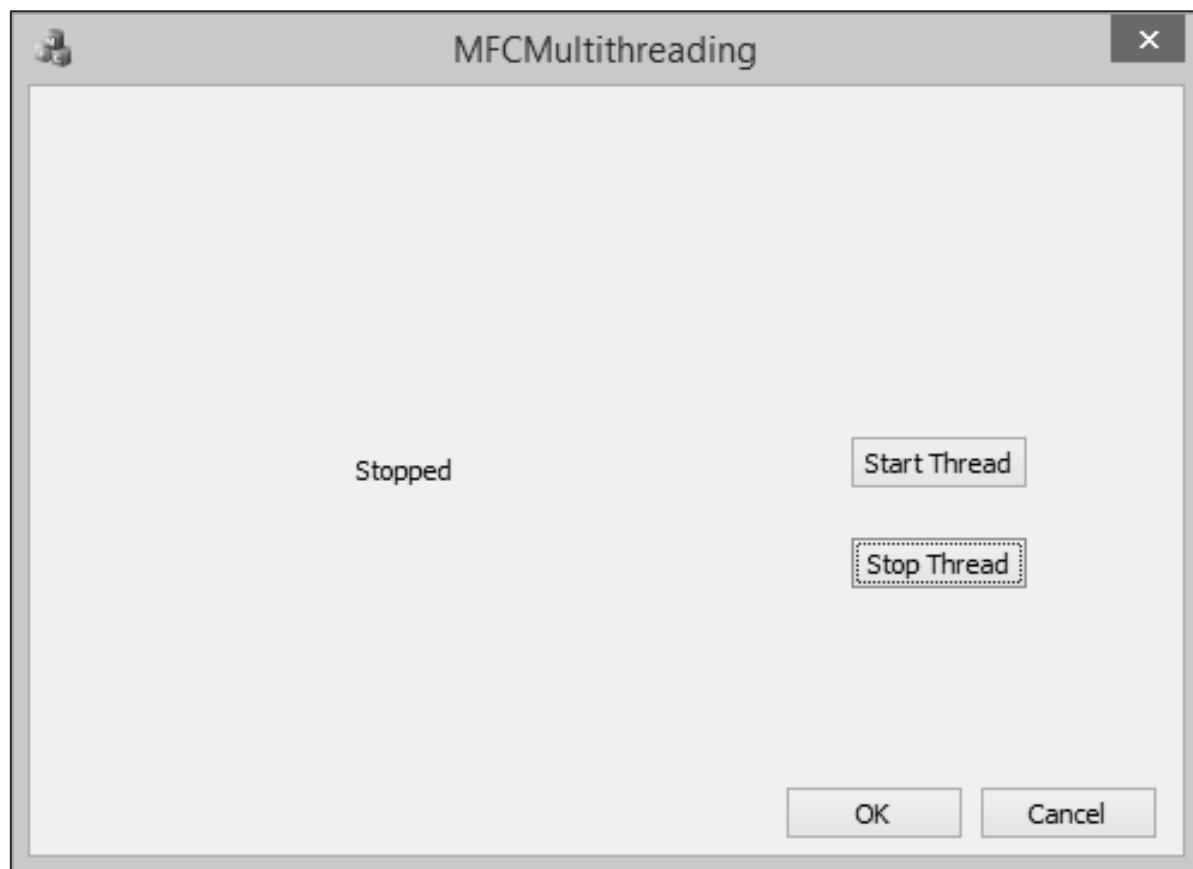
**Step 10:** When the above code is compiled and executed, you will see the following output.



**Step 11:** Now click on Start Thread button.



**Step 12:** Click the Stop Thread button. It will stop the thread.



# 23. MFC - Internet Programming

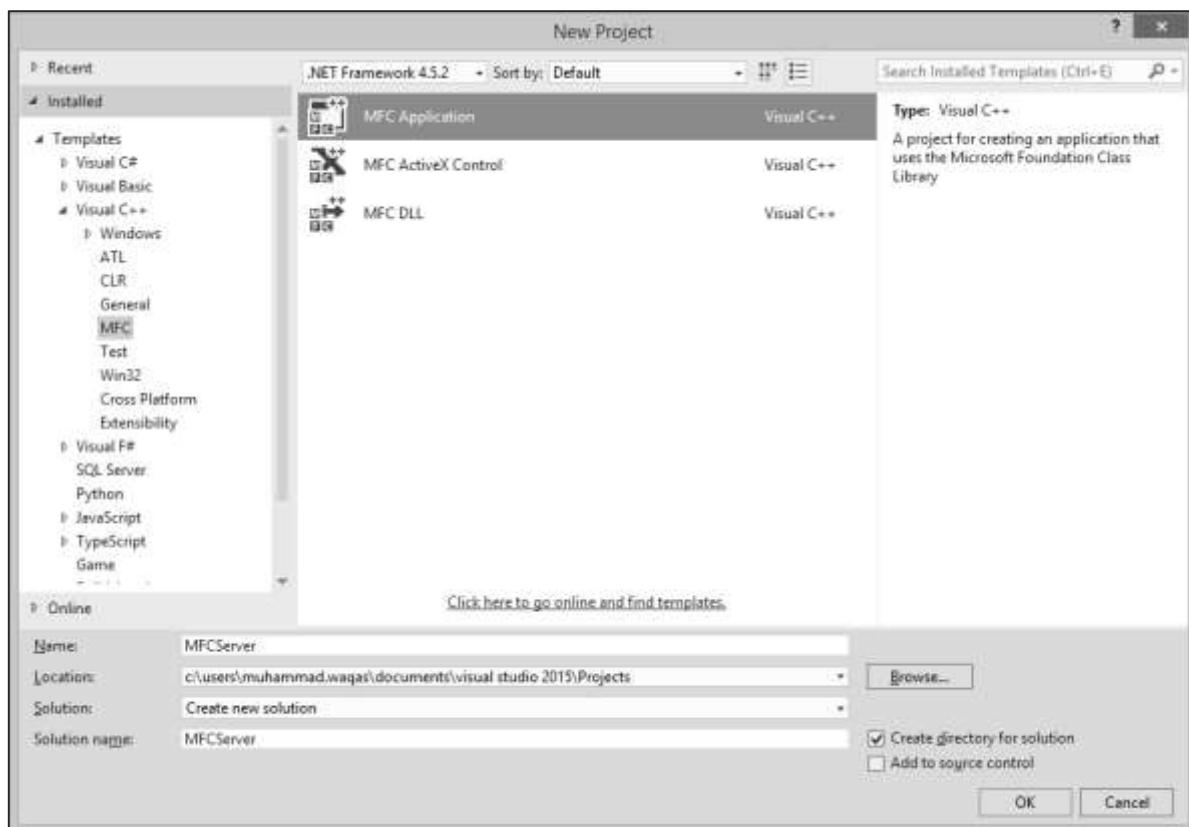
Microsoft provides many APIs for programming both client and server applications. Many new applications are being written for the Internet, and as technologies, browser capabilities, and security options change, new types of applications will be written. Your custom application can retrieve information and provide data on the Internet.

MFC provides a class **CSocket** for writing network communications programs with Windows Sockets.

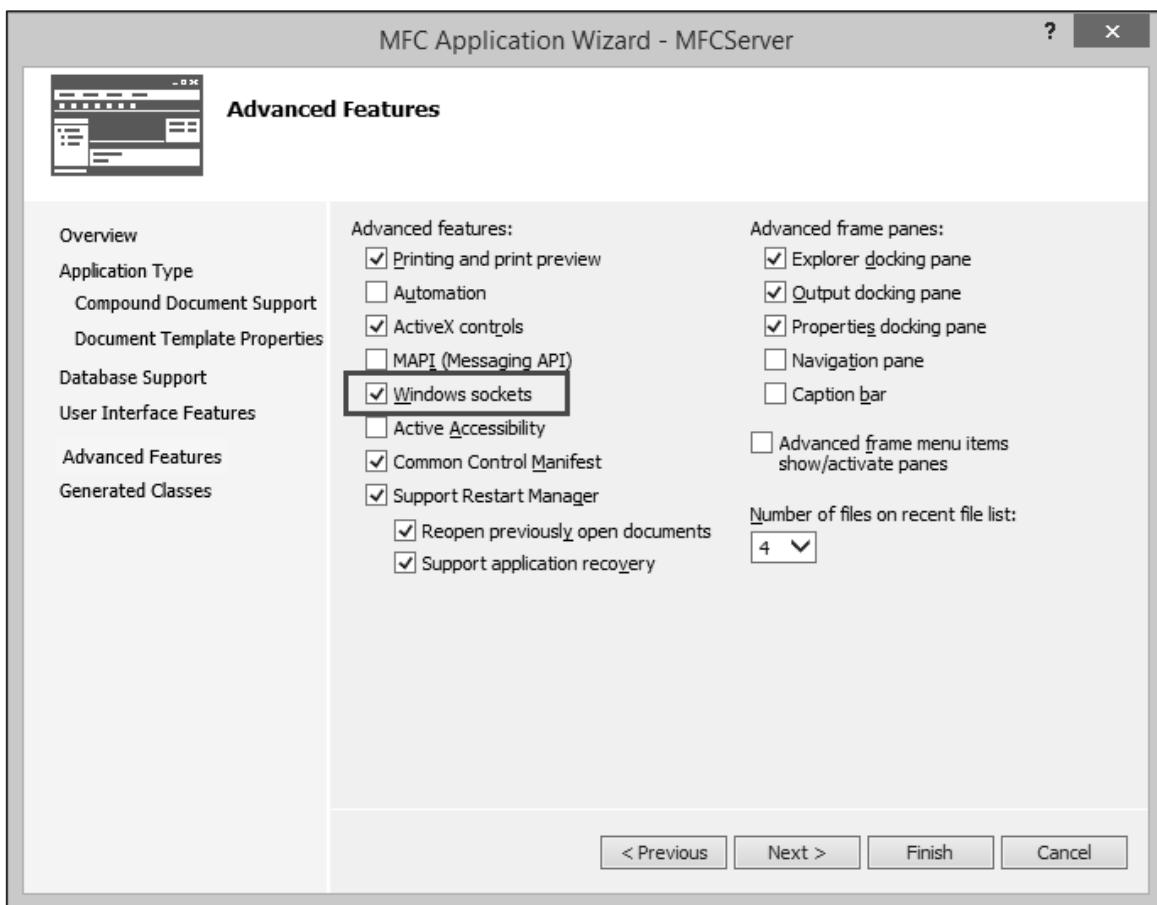
Here is a list of methods in CSocket class.

Name	Description
<b>Attach</b>	Attaches a SOCKET handle to a CSocket object.
<b>CancelBlockingCall</b>	Cancels a blocking call that is currently in progress.
<b>Create</b>	Creates a socket.
<b>FromHandle</b>	Returns a pointer to a CSocket object, given a SOCKET handle.
<b>IsBlocking</b>	Determines whether a blocking call is in progress.

Let us look into a simple example by creating a MFS SDI application.

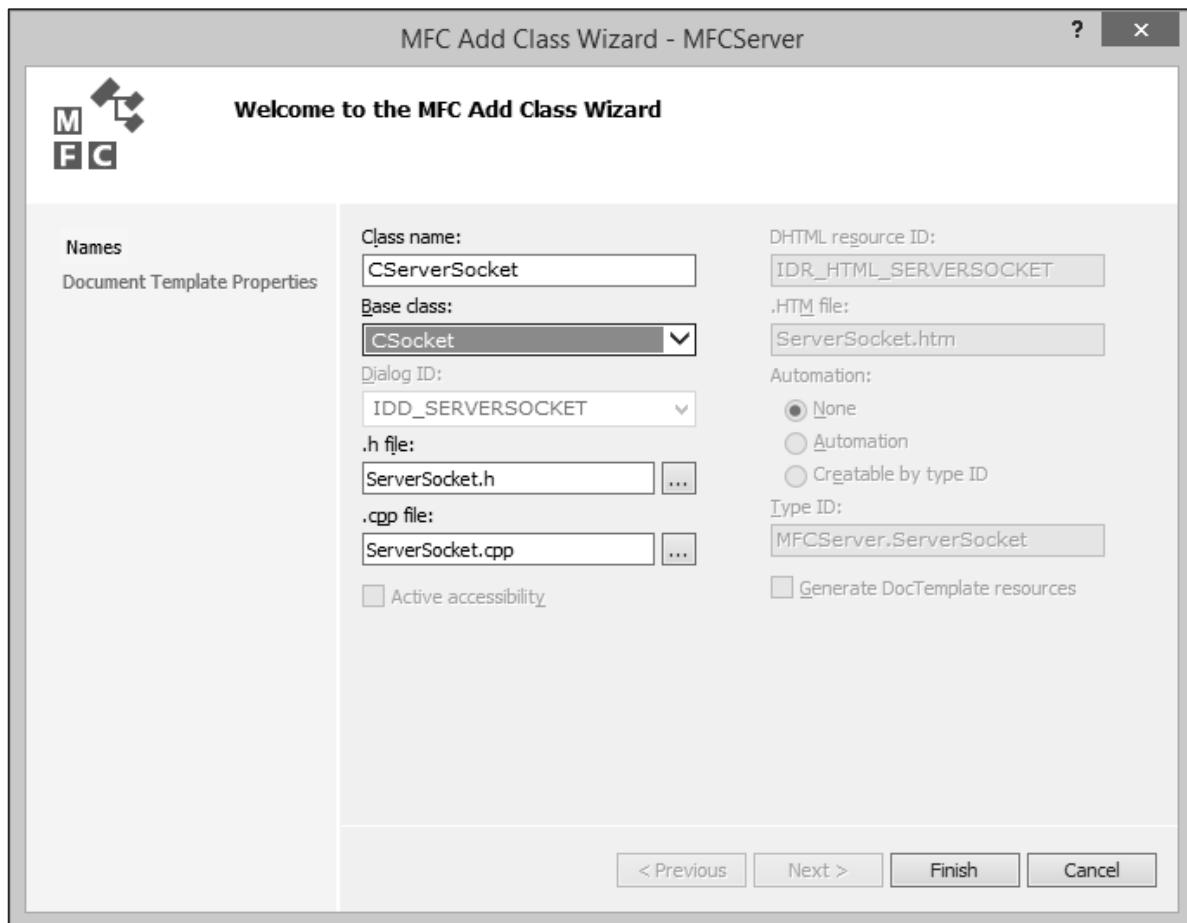


**Step 1:** Enter MFCServer in the name field and click OK.

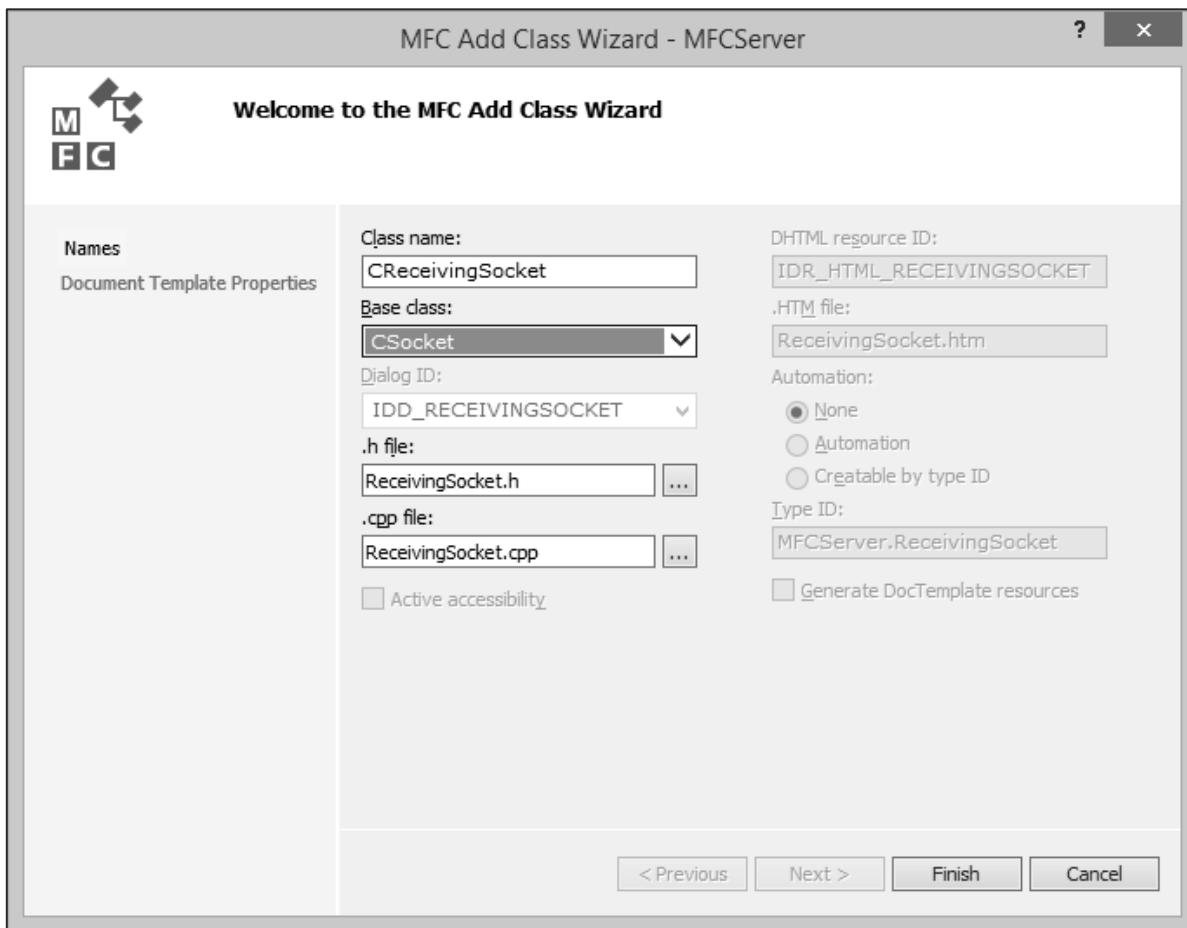


**Step 2:** On Advanced Features tab, check the Windows sockets option.

**Step 3:** Once the project is created, add a new MFC class CServerSocket.



**Step 4:** Select the CSocket as base class and click Finish.

**Step 5:** Add more MFC class CReceivingSocket.**Step 6:** CReceivingSocket will receive incoming messages from client.

In CMFCServerApp, the header file includes the following files:

```
#include "ServerSocket.h"
#include "MFCSERVERView.h"
```

**Step 7:** Add the following two class variables in CMFCServerApp class.

```
CServerSocket m_serverSocket;
CMFCSERVERView m_pServerView;
```

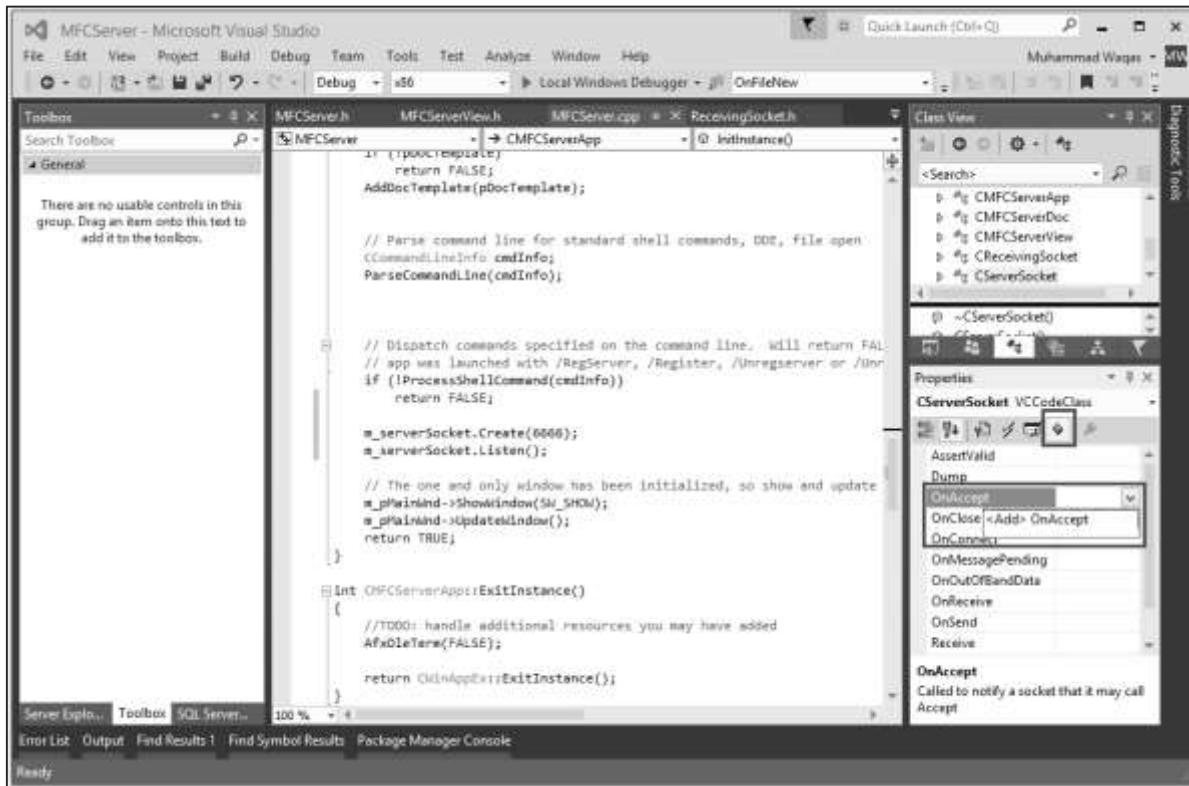
**Step 8:** In CMFCServerApp::InitInstance() method, create the socket and specify the port and then call the Listen method as shown below.

```
m_serverSocket.Create(6666);
m_serverSocket.Listen();
```

**Step 9:** Include the following header file in CMFCServerView header file.

```
#include "MFCSERVERDOC.H"
```

**Step 10:** Override the OnAccept function from Socket class.



**Step 11:** Select `CServerSocket` in class view and the highlighted icon in Properties window. Now, Add `OnAccept`. Here is the implementation of `OnAccept` function.

```
void CServerSocket::OnAccept(int nErrorCode)
{
    // TODO: Add your specialized code here and/or call the base class
    AfxMessageBox(L"Connection accepted");
    CSocket::OnAccept(nErrorCode);
}
```

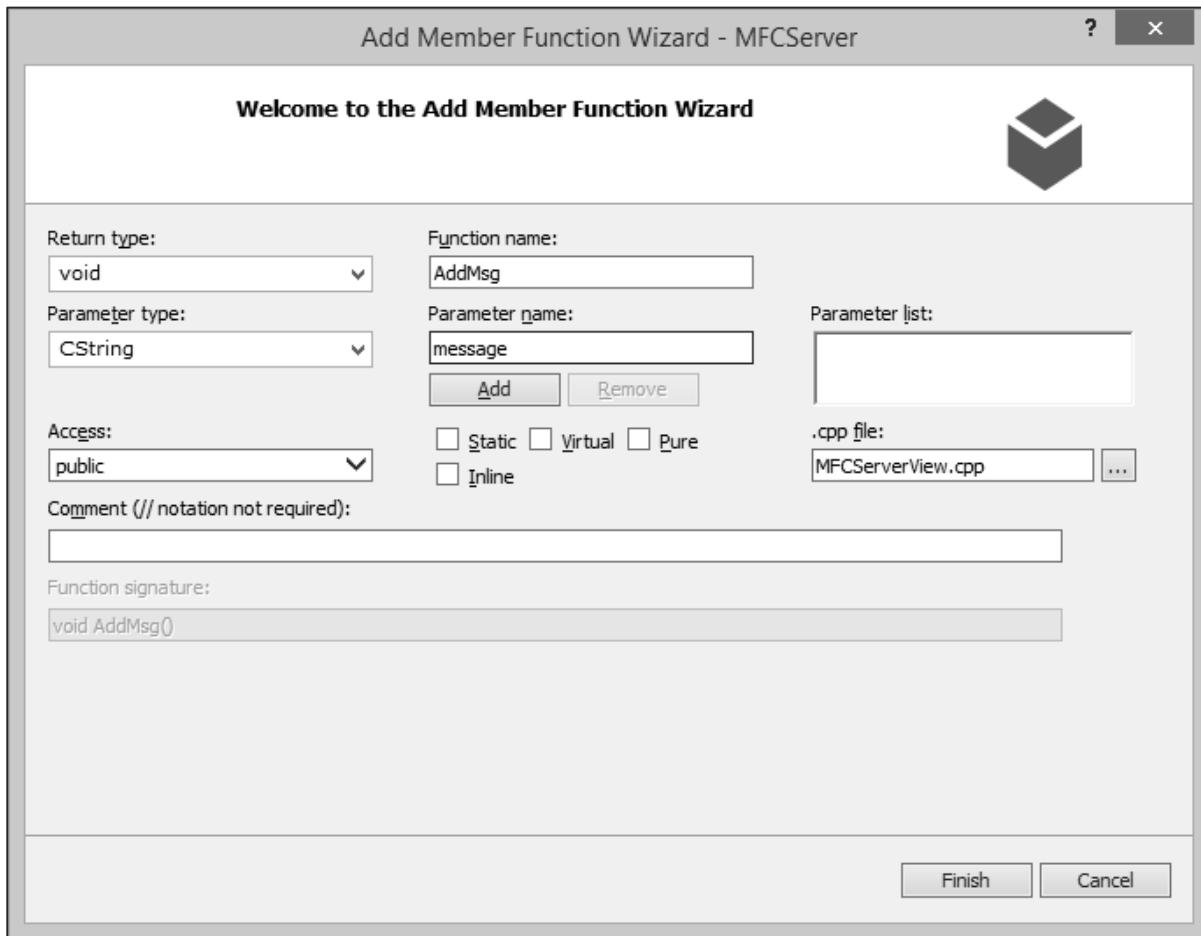
**Step 12:** Add `OnReceive()` function.

```
void CServerSocket::OnReceive(int nErrorCode)
```

```
{
    // TODO: Add your specialized code here and/or call the base class
    AfxMessageBox(L"Data Received");
    CSocket::OnReceive(nErrorCode);
}
```

**Step 13:** Add OnReceive() function in CReceivingSocket class.

Right-click on the CMFCSERVERVIEW class in solution explorer and select Add -> Add Function.



**Step 14:** Enter the above mentioned information and click finish.

**Step 15:** Add the following CStringArray variable in CMFCSERVERVIEW header file.

```
CStringArray m_msgArray;
```

**Step 16:** Here is the implementation of AddMsg() function.

```
void CMFCSERVERVIEW::AddMsg(CString message)
{
    m_msgArray.Add(message);
    Invalidate();
}
```

**Step 17:** Update the constructor as shown in the following code.

```
CMFCSERVERVIEW::CMFCSERVERVIEW()
{
    ((CMFCSERVERAPP*)AfxGetApp())->m_pServerView = this;
}
```

**Step 18:** Here is the implementation of OnDraw() function, which display messages.

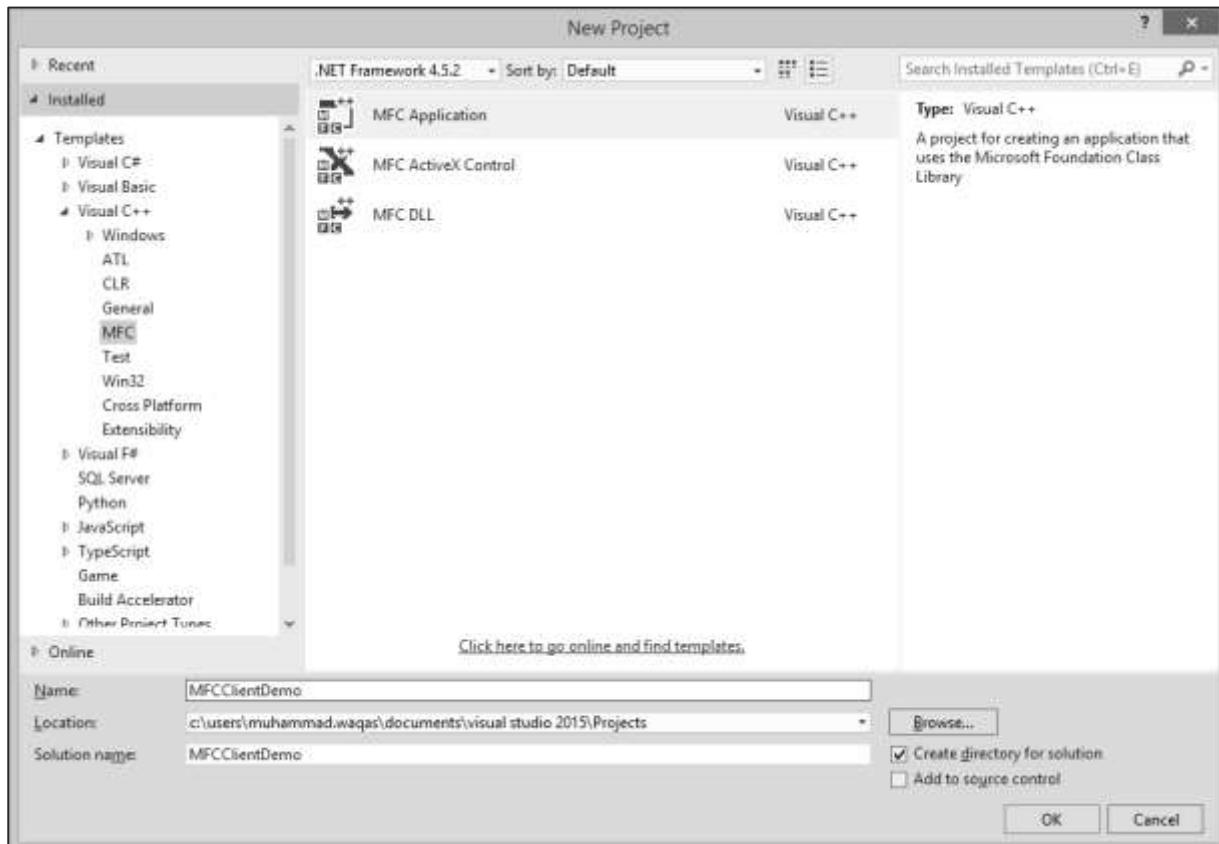
```
void CMFCSERVERVIEW::OnDraw(CDC* pDC)
{
    int y = 100;
    for (int i = 0; m_msgArray.GetSize(); i++)
    {
        pDC->TextOut(100, y, m_msgArray.GetAt(i));
        y += 50;
    }
    CMFCSERVERDOC* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

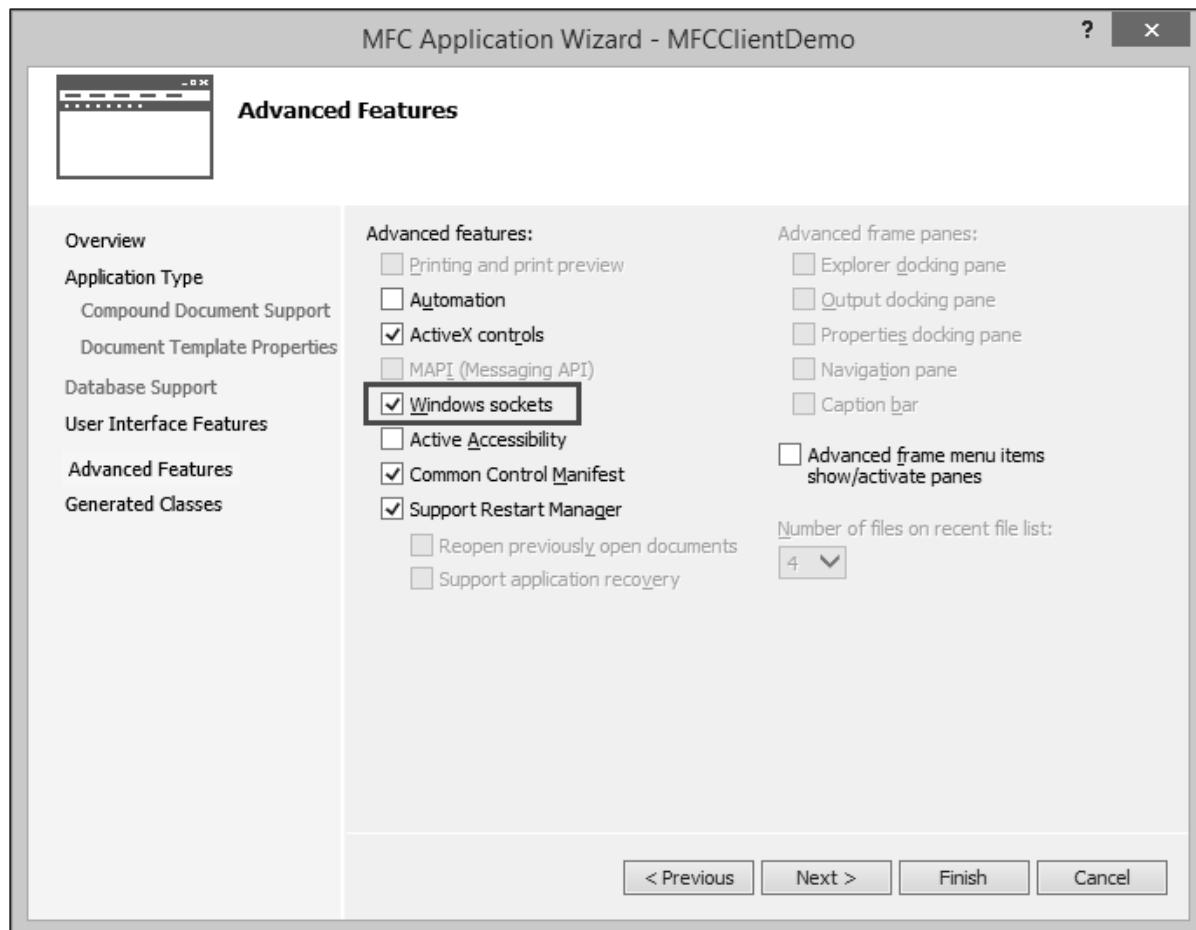
    // TODO: add draw code for native data here
}
```

**Step 19:** The server side is now complete. It will receive message from the client.

## Create Client Side Application

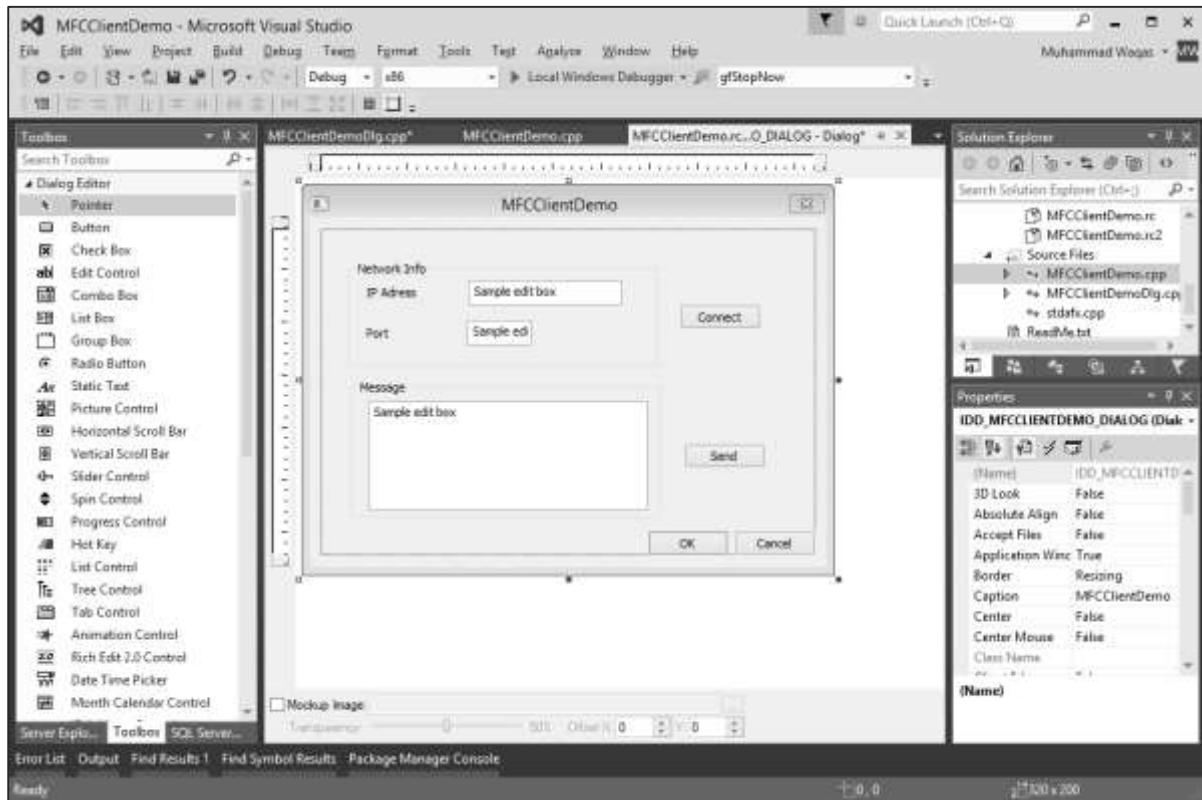
**Step 1:** Let us create a new MFC dialog based application for client side application.





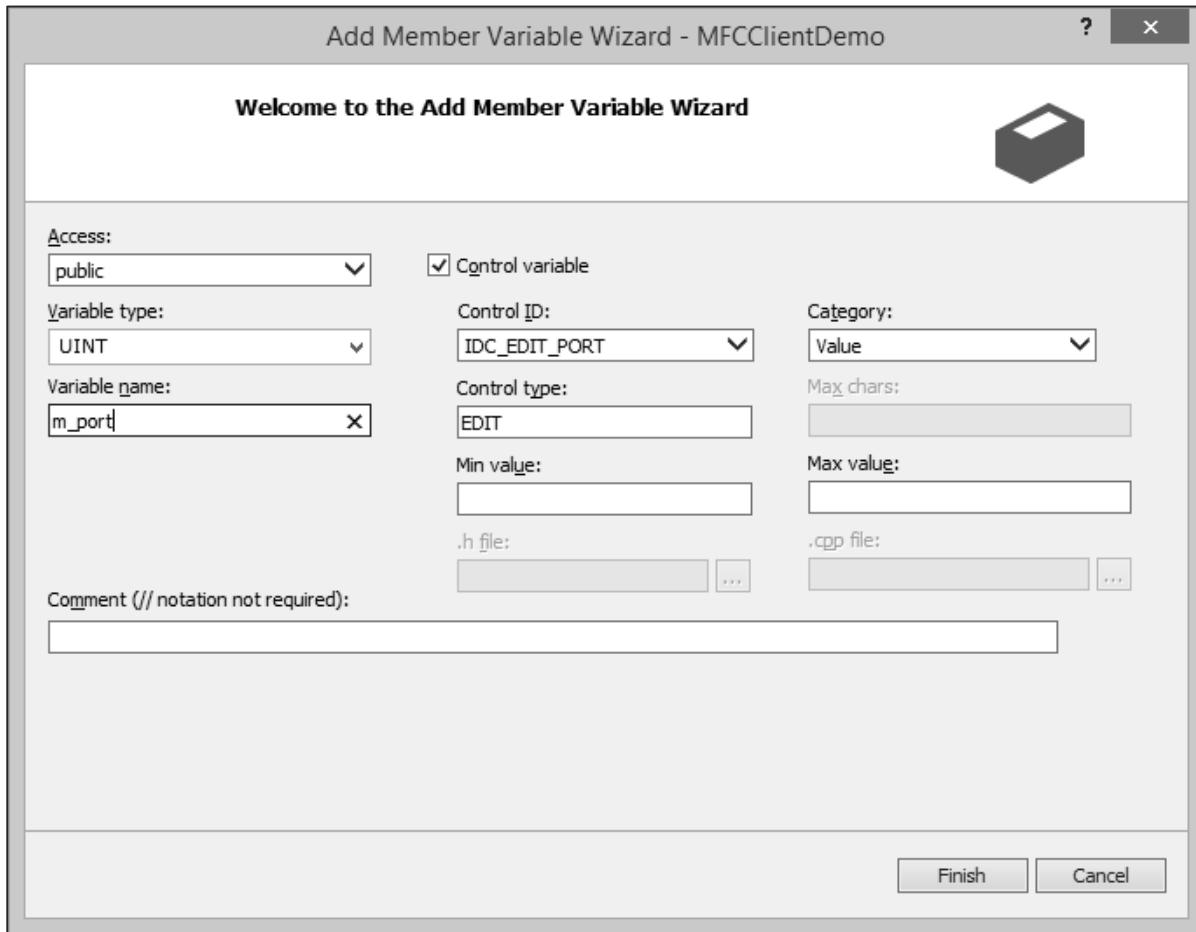
**Step 2:** On Advanced Features tab, check the Windows sockets option as shown above.

**Step 3:** Once the project is created, design your dialog box as shown in the following snapshot.

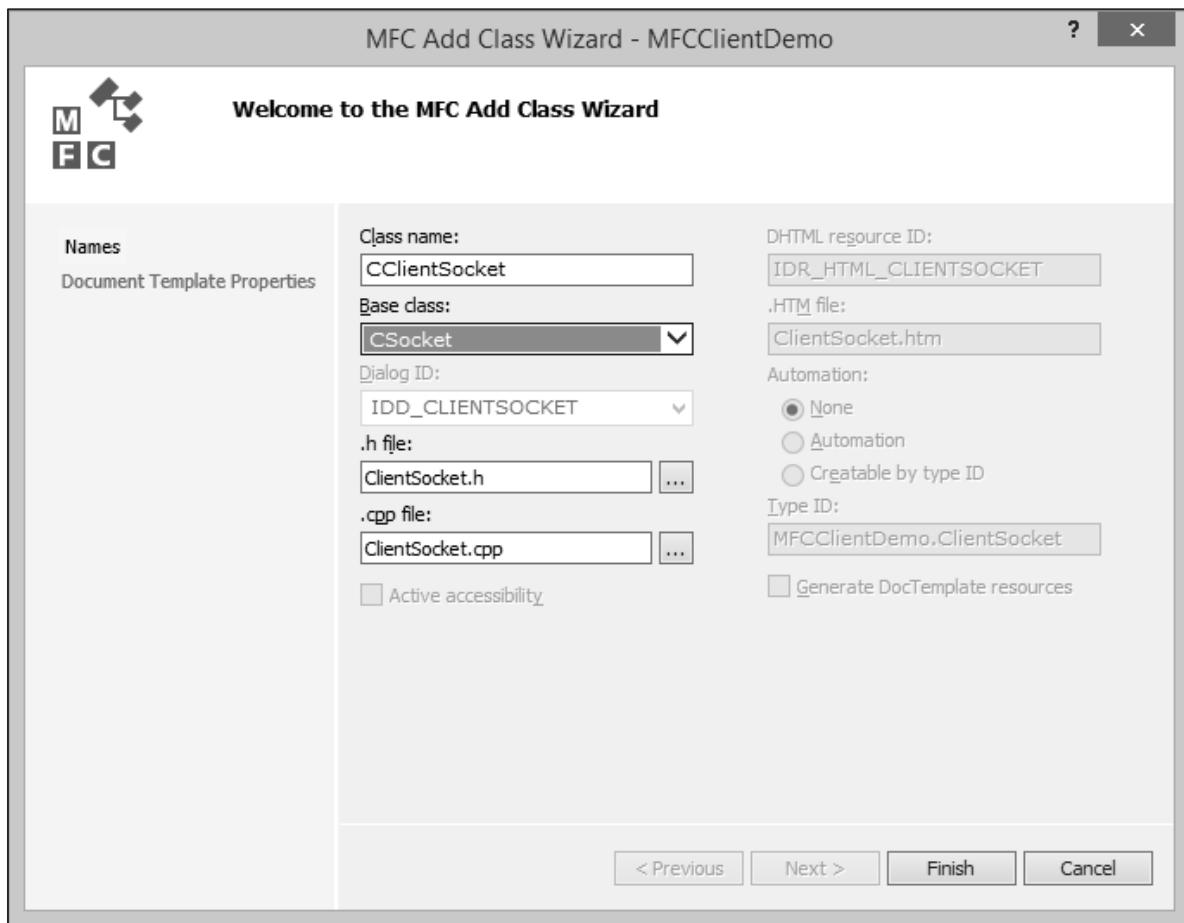


**Step 4:** Add event handlers for Connect and Send buttons.

**Step 5:** Add value variables for all the three edit controls. For port edit control, select the variable type UINT.



**Step 6:** Add MFC class for connecting and sending messages.



**Step 7:** Include the header file of CClientSocket class in the header file CMFCCClientDemoApp class and add the class variable. Similarly, add the class variable in CMFCCClientDemoDlg header file as well.

```
CClientSocket m_clientSocket;
```

**Step 8:** Here is the implementation of Connect button event handler.

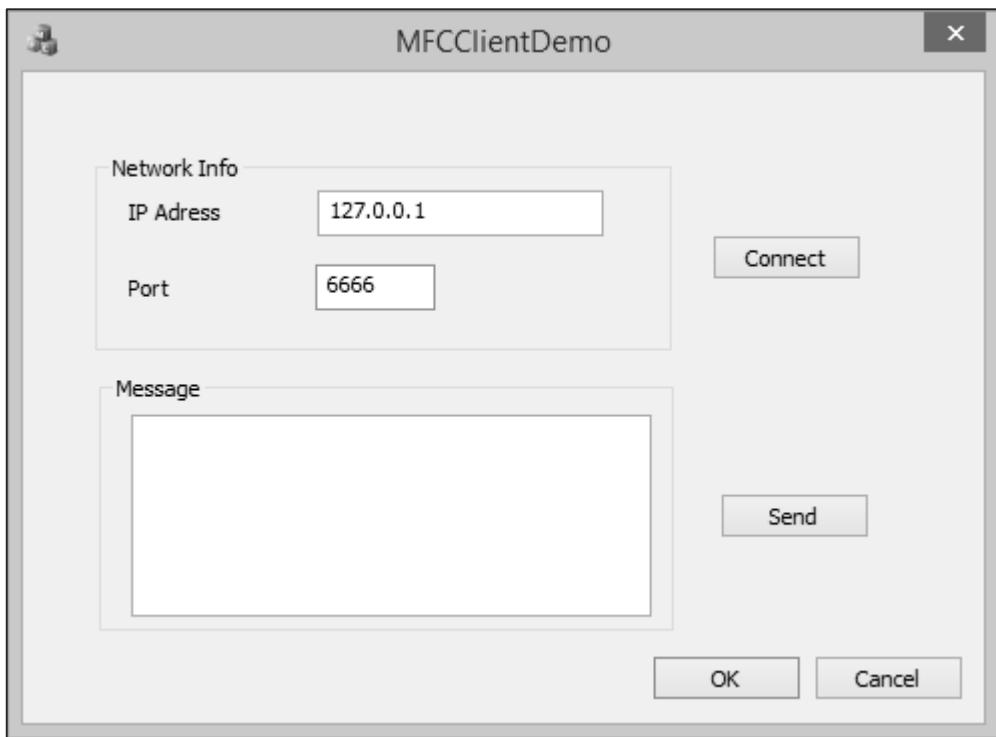
```
void CMFCCClientDemoDlg::OnBnClickedButtonConnect()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    m_clientSocket.Create();
    if (m_clientSocket.Connect(m_ipAddress, m_port))
    {
        AfxMessageBox(L"Connection Successfull");
    }
    else
```

```
{  
    AfxMessageBox(L"Connection Failed");  
}  
DWORD error = GetLastError();  
}
```

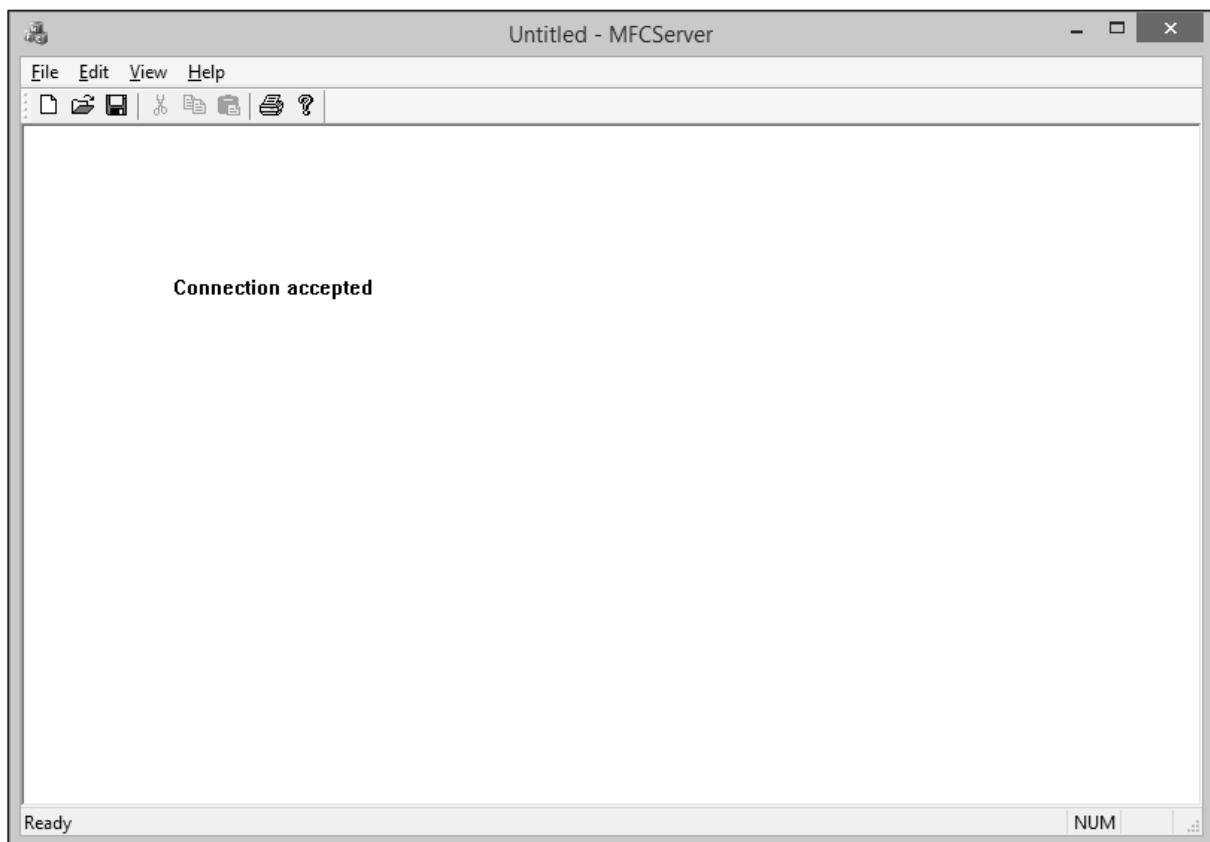
**Step 9:** Here is the implementation of Send button event handler.

```
void CMFCCClientDemoDlg::OnBnClickedButtonSend()  
{  
    // TODO: Add your control notification handler code here  
    UpdateData(TRUE);  
    if (m_clientSocket.Send(m_message.GetBuffer(m_message.GetLength()),  
    m_message.GetLength()))  
    {  
    }  
    else  
    {  
        AfxMessageBox(L"Failed to send message");  
    }  
}
```

**Step 10:** First run the Server application and then the client application. Enter the local host ip and port and click Connect.



**Step 11:** You will now see the message on Server side as shown in the following snapshot.



# 24. MFC - GDI

Windows provides a variety of drawing tools to use in device contexts. It provides pens to draw lines, brushes to fill interiors, and fonts to draw text. MFC provides graphic-object classes equivalent to the drawing tools in Windows.

## Drawing

A device context is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer. All drawing calls are made through a device-context object, which encapsulates the Windows APIs for drawing lines, shapes, and text.

Device contexts allow device-independent drawing in Windows. Device contexts can be used to draw to the screen, to the printer, or to a metafile.

**CDC** is the most fundamental class to draw in MFC. The CDC object provides member functions to perform the basic drawing steps, as well as members for working with a display context associated with the client area of a window.

Here is a list of methods in CDC class.

Name	Description
<b>AbortDoc</b>	Terminates the current print job, erasing everything the application has written to the device since the last call of the <b>StartDoc</b> member function.
<b>AbortPath</b>	Closes and discards any paths in the device context.
<b>AddMetaFileComment</b>	Copies the comment from a buffer into a specified enhanced-format metafile.
<b>AlphaBlend</b>	Displays bitmaps that have transparent or semitransparent pixels.
<b>AngleArc</b>	Draws a line segment and an arc, and moves the current position to the ending point of the arc.
<b>Arc</b>	Draws an elliptical arc.
<b>ArcTo</b>	Draws an elliptical arc. This function is similar to <b>Arc</b> , except that the current position is updated.
<b>Attach</b>	Attaches a Windows device context to this CDC object.
<b>BeginPath</b>	Opens a path bracket in the device context.
<b>BitBlt</b>	Copies a bitmap from a specified device context.
<b>Chord</b>	Draws a chord (a closed figure bounded by the intersection of an ellipse and a line segment).
<b>CloseFigure</b>	Closes an open figure in a path.
<b>CreateCompatibleDC</b>	Creates a memory-device context that is compatible with another device context. You can use it to prepare images in memory.
<b>CreateDC</b>	Creates a device context for a specific device.
<b>CreateIC</b>	Creates an information context for a specific device. This provides a fast way to get information about the device without creating a device context.

<b>DeleteDC</b>	Deletes the Windows device context associated with this CDC object.
<b>DeleteTempMap</b>	Called by the <b>CWinApp</b> idle-time handler to delete any temporary CDC object created by <b>FromHandle</b> . Also detaches the device context.
<b>Detach</b>	Detaches the Windows device context from this CDC object.
<b>DPtoHIMETRIC</b>	Converts device units into <b>HIMETRIC</b> units.
<b>DPtoLP</b>	Converts device units into logical units.
<b>Draw3dRect</b>	Draws a three-dimensional rectangle.
<b>DrawDragRect</b>	Erases and redraws a rectangle as it is dragged.
<b>DrawEdge</b>	Draws the edges of a rectangle.
<b>DrawEscape</b>	Accesses drawing capabilities of a video display that are not directly available through the graphics device interface (GDI).
<b>DrawFocusRect</b>	Draws a rectangle in the style used to indicate focus.
<b>DrawFrameControl</b>	Draw a frame control.
<b>DrawIcon</b>	Draws an icon.
<b>DrawState</b>	Displays an image and applies a visual effect to indicate a state.
<b>DrawText</b>	Draws formatted text in the specified rectangle.
<b>DrawTextEx</b>	Draws formatted text in the specified rectangle using additional formats.
<b>Ellipse</b>	Draws an ellipse.
<b>EndDoc</b>	Ends a print job started by the StartDoc member function.
<b>EndPage</b>	Informs the device driver that a page is ending.
<b>EndPath</b>	Closes a path bracket and selects the path defined by the bracket into the device context.
<b>EnumObjects</b>	Enumerates the pens and brushes available in a device context.
<b>Escape</b>	Allows applications to access facilities that are not directly available from a particular device through GDI. Also allows access to Windows escape functions. Escape calls made by an application are translated and sent to the device driver.
<b>ExcludeClipRect</b>	Creates a new clipping region that consists of the existing clipping region minus the specified rectangle.
<b>ExcludeUpdateRgn</b>	Prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.
<b>ExtFloodFill</b>	Fills an area with the current brush. Provides more flexibility than the <a href="#">FloodFill</a> member function.
<b>ExtTextOut</b>	Writes a character string within a rectangular region using the currently selected font.
<b>FillPath</b>	Closes any open figures in the current path and fills the path's interior by using the current brush and polygon-filling mode.
<b>FillRect</b>	Fills a given rectangle by using a specific brush.
<b>FillRgn</b>	Fills a specific region with the specified brush.
<b>FillSolidRect</b>	Fills a rectangle with a solid color.
<b>FlattenPath</b>	Transforms any curves in the path selected into the current device context, and turns each curve into a sequence of lines.
<b>FloodFill</b>	Fills an area with the current brush.

<b>FrameRect</b>	Draws a border around a rectangle.
<b>FrameRgn</b>	Draws a border around a specific region using a brush.
<b>FromHandle</b>	Returns a pointer to a CDC object when given a handle to a device context. If a CDC object is not attached to the handle, a temporary CDC object is created and attached.
<b>GetArcDirection</b>	Returns the current arc direction for the device context.
<b>GetAspectRatioFilter</b>	Retrieves the setting for the current aspect-ratio filter.
<b>GetBkColor</b>	Retrieves the current background color.
<b>GetBkMode</b>	Retrieves the background mode.
<b>GetBoundsRect</b>	Returns the current accumulated bounding rectangle for the specified device context.
<b>GetBrushOrg</b>	Retrieves the origin of the current brush.
<b>GetCharABCWidths</b>	Retrieves the widths, in logical units, of consecutive characters in a given range from the current font.
<b>GetCharABCWidthsI</b>	Retrieves the widths, in logical units, of consecutive glyph indices in a specified range from the current TrueType font.
<b>GetCharacterPlacement</b>	Retrieves various types of information on a character string.
<b>GetCharWidth</b>	Retrieves the fractional widths of consecutive characters in a given range from the current font.
<b>GetCharWidthI</b>	Retrieves the widths, in logical coordinates, of consecutive glyph indices in a specified range from the current font.
<b>GetClipBox</b>	Retrieves the dimensions of the tightest bounding rectangle around the current clipping boundary.
<b>GetColorAdjustment</b>	Retrieves the color adjustment values for the device context.
<b>GetCurrentBitmap</b>	Returns a pointer to the currently selected <b>CBitmap</b> object.
<b>GetCurrentBrush</b>	Returns a pointer to the currently selected <b>CBrush</b> object.
<b>GetCurrentFont</b>	Returns a pointer to the currently selected <b>CFont</b> object.
<b>GetCurrentPalette</b>	Returns a pointer to the currently selected <b>CPalette</b> object.
<b>GetCurrentPen</b>	Returns a pointer to the currently selected <b>CPen</b> object.
<b>GetCurrentPosition</b>	Retrieves the current position of the pen (in logical coordinates).
<b>GetDCBrushColor</b>	Retrieves the current brush color.
<b>GetDCPenColor</b>	Retrieves the current pen color.
<b>GetDeviceCaps</b>	Retrieves a specified kind of device-specific information about a given display device's capabilities.
<b>GetFontData</b>	Retrieves font metric information from a scalable font file. The information to retrieve is identified by specifying an offset into the font file and the length of the information to return.
<b>GetFontLanguageInfo</b>	Returns information about the currently selected font for the specified display context.
<b>GetGlyphOutline</b>	Retrieves the outline curve or bitmap for an outline character in the current font.
<b>GetGraphicsMode</b>	Retrieves the current graphics mode for the specified device context.
<b>GetHalftoneBrush</b>	Retrieves a halftone brush.
<b>GetKerningPairs</b>	Retrieves the character kerning pairs for the font that is currently selected in the specified device context.

<b>GetLayout</b>	Retrieves the layout of a device context (DC). The layout can be either left to right (default) or right to left (mirrored).
<b>GetMapMode</b>	Retrieves the current mapping mode.
<b>GetMiterLimit</b>	Returns the miter limit for the device context.
<b>GetNearestColor</b>	Retrieves the closest logical color to a specified logical color that the given device can represent.
<b>GetOutlineTextMetrics</b>	Retrieves font metric information for TrueType fonts.
<b>GetOutputCharWidth</b>	Retrieves the widths of individual characters in a consecutive group of characters from the current font using the output device context.
<b>GetOutputTabbedTextExtent</b>	Computes the width and height of a character string on the output device context.
<b>GetOutputTextExtent</b>	Computes the width and height of a line of text on the output device context using the current font to determine the dimensions.
<b>GetOutputTextMetrics</b>	Retrieves the metrics for the current font from the output device context.
<b>GetPath</b>	Retrieves the coordinates defining the endpoints of lines and the control points of curves found in the path that is selected into the device context.
<b>GetPixel</b>	Retrieves the RGB color value of the pixel at the specified point.
<b>GetPolyFillMode</b>	Retrieves the current polygon-filling mode.
<b>GetROP2</b>	Retrieves the current drawing mode.
<b>GetSafeHdc</b>	Returns <a href="#">m_hDC</a> , the output device context.
<b>GetStretchBltMode</b>	Retrieves the current bitmap-stretching mode.
<b>GetTabbedTextExtent</b>	Computes the width and height of a character string on the attribute device context.
<b>GetTextAlign</b>	Retrieves the text-alignment flags.
<b>GetTextCharacterExtra</b>	Retrieves the current setting for the amount of intercharacter spacing.
<b>GetTextColor</b>	Retrieves the current text color.
<b>GetTextExtent</b>	Computes the width and height of a line of text on the attribute device context using the current font to determine the dimensions.
<b>GetTextExtentExPointI</b>	Retrieves the number of characters in a specified string that will fit within a specified space and fills an array with the text extent for each of those characters.
<b>GetTextExtentPointI</b>	Retrieves the width and height of the specified array of glyph indices.
<b>GetTextFace</b>	Copies the typeface name of the current font into a buffer as a null-terminated string.
<b>GetTextMetrics</b>	Retrieves the metrics for the current font from the attribute device context.
<b>GetViewportExt</b>	Retrieves the x- and y-extents of the viewport.
<b>GetViewportOrg</b>	Retrieves the x- and y-coordinates of the viewport origin.
<b>GetWindow</b>	Returns the window associated with the display device context.
<b>GetWindowExt</b>	Retrieves the x- and y-extents of the associated window.
<b>GetWindowOrg</b>	Retrieves the x- and y-coordinates of the origin of the associated window.

<b>GetWorldTransform</b>	Retrieves the current world-space to page-space transformation.
<b>GradientFill</b>	Fills rectangle and triangle structures with a gradating color.
<b>GrayString</b>	Draws dimmed (grayed) text at the given location.
<b>HIMETRICtoDP</b>	Converts HIMETRIC units into device units.
<b>HIMETRICtoLP</b>	Converts HIMETRIC units into logical units.
<b>IntersectClipRect</b>	Creates a new clipping region by forming the intersection of the current region and a rectangle.
<b>InvertRect</b>	Inverts the contents of a rectangle.
<b>InvertRgn</b>	Inverts the colors in a region.
<b>IsPrinting</b>	Determines whether the device context is being used for printing.
<b>LineTo</b>	Draws a line from the current position up to, but not including, a point.
<b>LPtoDP</b>	Converts logical units into device units.
<b>LPtoHIMETRIC</b>	Converts logical units into HIMETRIC units.
<b>MaskBlt</b>	Combines the color data for the source and destination bitmaps using the given mask and raster operation.
<b>ModifyWorldTransform</b>	Changes the world transformation for a device context using the specified mode.
<b>MoveTo</b>	Moves the current position.
<b>OffsetClipRgn</b>	Moves the clipping region of the given device.
<b>OffsetViewportOrg</b>	Modifies the viewport origin relative to the coordinates of the current viewport origin.
<b>OffsetWindowOrg</b>	Modifies the window origin relative to the coordinates of the current window origin.
<b>PaintRgn</b>	Fills a region with the selected brush.
<b>PatBlt</b>	Creates a bit pattern.
<b>Pie</b>	Draws a pie-shaped wedge.
<b>PlayMetaFile</b>	Plays the contents of the specified metafile on the given device. The enhanced version of <b>PlayMetaFile</b> displays the picture stored in the given enhanced-format metafile. The metafile can be played any number of times.
<b>PigBlt</b>	Performs a bit-block transfer of the bits of color data from the specified rectangle in the source device context to the specified parallelogram in the given device context.
<b>PolyBezier</b>	Draws one or more Bzier splines. The current position is neither used nor updated.
<b>PolyBezierTo</b>	Draws one or more Bzier splines, and moves the current position to the ending point of the last Bzier spline.
<b>PolyDraw</b>	Draws a set of line segments and Bzier splines. This function updates the current position.
<b>Polygon</b>	Draws a polygon consisting of two or more points (vertices) connected by lines.
<b>Polyline</b>	Draws a set of line segments connecting the specified points.
<b>PolylineTo</b>	Draws one or more straight lines and moves the current position to the ending point of the last line.
<b>PolyPolygon</b>	Creates two or more polygons that are filled using the current polygon-filling mode. The polygons may be disjoint or they may overlap.

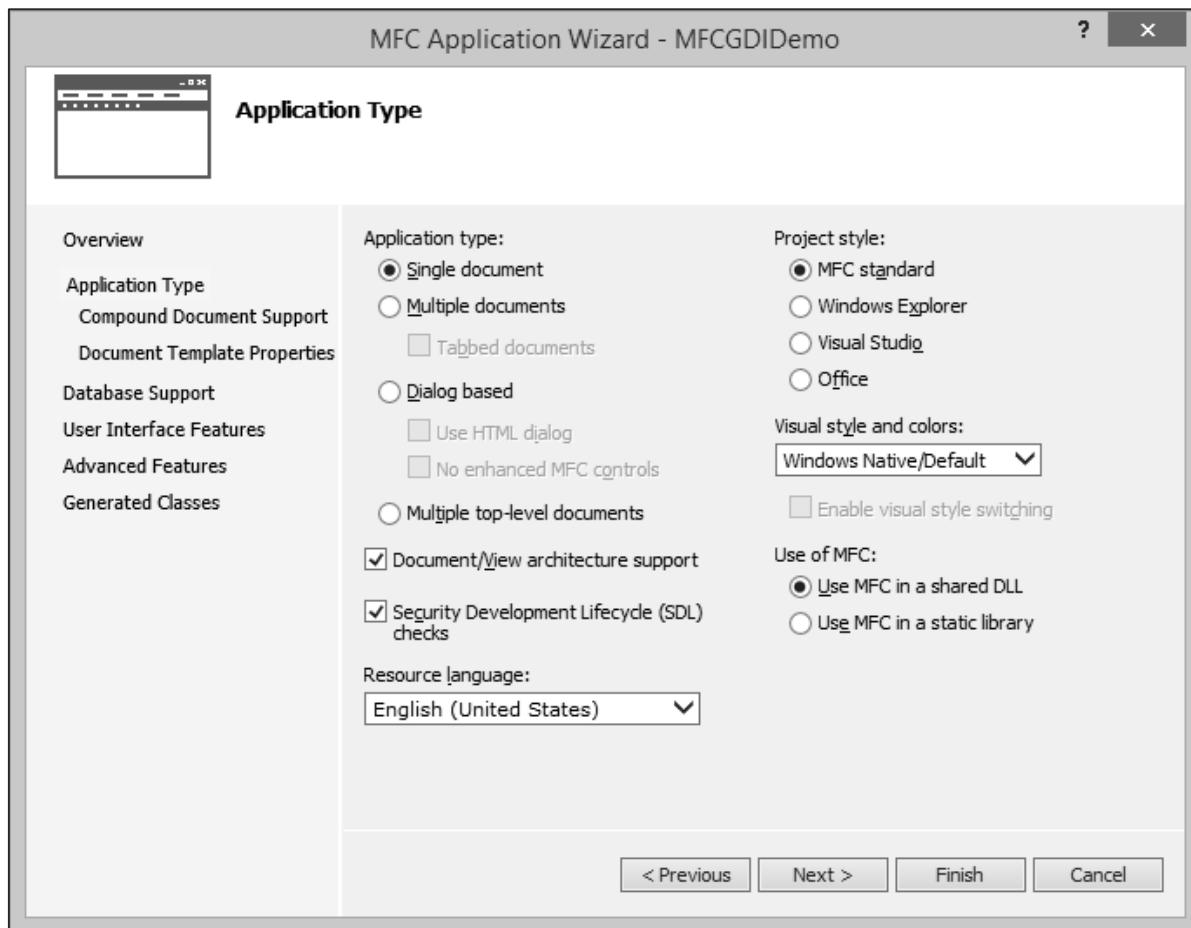
<b>PolyPolyline</b>	Draws multiple series of connected line segments. The current position is neither used nor updated by this function.
<b>PtVisible</b>	Specifies whether the given point is within the clipping region.
<b>RealizePalette</b>	Maps palette entries in the current logical palette to the system palette.
<b>Rectangle</b>	Draws a rectangle using the current pen and fills it using the current brush.
<b>RectVisible</b>	Determines whether any part of the given rectangle lies within the clipping region.
<b>ReleaseAttribDC</b>	Releases <b>m_hAttribDC</b> , the attribute device context.
<b>ReleaseOutputDC</b>	Releases <b>m_hDC</b> , the output device context.
<b>ResetDC</b>	Updates the <b>m_hAttribDC</b> device context.
<b>RestoreDC</b>	Restores the device context to a previous state saved with <b>SaveDC</b> .
<b>RoundRect</b>	Draws a rectangle with rounded corners using the current pen and filled using the current brush.
<b>SaveDC</b>	Saves the current state of the device context.
<b>ScaleViewportExt</b>	Modifies the viewport extent relative to the current values.
<b>ScaleWindowExt</b>	Modifies the window extents relative to the current values.
<b>ScrollDC</b>	Scrolls a rectangle of bits horizontally and vertically.
<b>SelectClipPath</b>	Selects the current path as a clipping region for the device context, combining the new region with any existing clipping region by using the specified mode.
<b>SelectClipRgn</b>	Combines the given region with the current clipping region by using the specified mode.
<b>SelectObject</b>	Selects a GDI drawing object such as a pen.
<b>SelectPalette</b>	Selects the logical palette.
<b>SelectStockObject</b>	Selects one of the predefined stock pens, brushes, or fonts provided by Windows.
<b>SetAbortProc</b>	Sets a programmer-supplied callback function that Windows calls if a print job must be aborted.
<b>SetArcDirection</b>	Sets the drawing direction to be used for arc and rectangle functions.
<b>SetAttribDC</b>	Sets <b>m_hAttribDC</b> , the attribute device context.
<b>SetBkColor</b>	Sets the current background color.
<b>SetBkMode</b>	Sets the background mode.
<b>SetBoundsRect</b>	Controls the accumulation of bounding-rectangle information for the specified device context.
<b>SetBrushOrg</b>	Specifies the origin for the next brush selected into a device context.
<b>SetColorAdjustment</b>	Sets the color adjustment values for the device context using the specified values.
<b>SetDCBrushColor</b>	Sets the current brush color.
<b>SetDCPenColor</b>	Sets the current pen color.
<b>SetGraphicsMode</b>	Sets the current graphics mode for the specified device context.
<b>SetLayout</b>	Changes the layout of a device context (DC).
<b>SetMapMode</b>	Sets the current mapping mode.
<b>SetMapperFlags</b>	Alters the algorithm that the font mapper uses when it maps logical fonts to physical fonts.

<b>SetMiterLimit</b>	Sets the limit for the length of miter joins for the device context.
<b>SetOutputDC</b>	Sets m_hDC, the output device context.
<b>SetPixel</b>	Sets the pixel at the specified point to the closest approximation of the specified color.
<b>SetPixelV</b>	Sets the pixel at the specified coordinates to the closest approximation of the specified color. <b>SetPixelV</b> is faster than <b>SetPixel</b> because it does not need to return the color value of the point actually painted.
<b>SetPolyFillMode</b>	Sets the polygon-filling mode.
<b>SetROP2</b>	Sets the current drawing mode.
<b>SetStretchBltMode</b>	Sets the bitmap-stretching mode.
<b>SetTextAlign</b>	Sets the text-alignment flags.
<b>SetTextCharacterExtra</b>	Sets the amount of intercharacter spacing.
<b>SetTextColor</b>	Sets the text color.
<b>SetTextJustification</b>	Adds space to the break characters in a string.
<b>SetViewportExt</b>	Sets the x- and y-extents of the viewport.
<b>SetViewportOrg</b>	Sets the viewport origin.
<b>SetWindowExt</b>	Sets the x- and y-extents of the associated window.
<b>SetWindowOrg</b>	Sets the window origin of the device context.
<b>SetWorldTransform</b>	Sets the current world-space to page-space transformation.
<b>StartDoc</b>	Informs the device driver that a new print job is starting.
<b>StartPage</b>	Informs the device driver that a new page is starting.
<b>StretchBlt</b>	Moves a bitmap from a source rectangle and device into a destination rectangle, stretching or compressing the bitmap if necessary to fit the dimensions of the destination rectangle.
<b>StrokeAndFillPath</b>	Closes any open figures in a path, strikes the outline of the path by using the current pen, and fills its interior by using the current brush.
<b>StrokePath</b>	Renders the specified path by using the current pen.
<b>TabbedTextOut</b>	Writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions.
<b>TextOut</b>	Writes a character string at a specified location using the currently selected font.
<b>TransparentBlt</b>	Transfers a bit-block of color data from the specified source device context into a destination device context, rendering a specified color transparent in the transfer.
<b>UpdateColors</b>	Updates the client area of the device context by matching the current colors in the client area to the system palette on a pixel-by-pixel basis.
<b>WidenPath</b>	Redefines the current path as the area that would be painted if the path were stroked using the pen currently selected into the device context.

## Lines

---

**Step 1:** Let us look into a simple example by creating a new MFC based single document project with **MFCGDI Demo** name.



**Step 2:** Once the project is created, go the Solution Explorer and double click on the **MFCGDDIDemoView.cpp** file under the Source Files folder.

**Step 3:** Draw the line as shown below in **CMFCGDDIDemoView::OnDraw()** method.

```
void CMFCGDDIDemoView::OnDraw(CDC* pDC)
{
    pDC->MoveTo(95, 125);
    pDC->LineTo(230, 125);
    CMFCGDDIDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

**Step 4:** Run this application. You will see the following output.



**Step 5:** The CDC::MoveTo() method is used to set the starting position of a line.

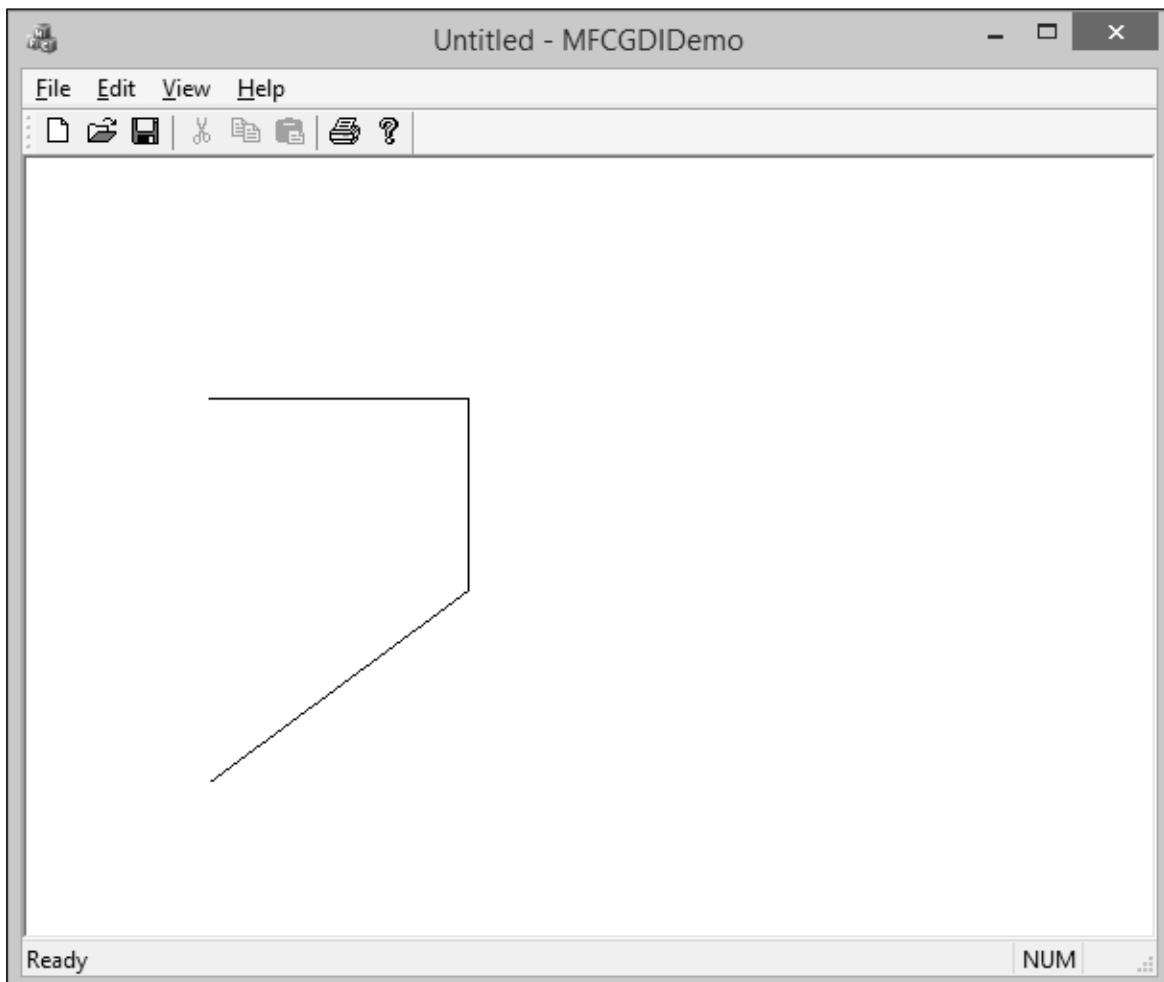
When using LineTo(), the program starts from the MoveTo() point to the LineTo() end.

After LineTo() when you do not call MoveTo(), and call again LineTo() with other point value, the program will draw a line from the previous LineTo() to the new LineTo() point.

**Step 6:** To draw different lines, you can use this property as shown in the following code.

```
void CMFCGDDIDemoView::OnDraw(CDC* pDC)
{
    pDC->MoveTo(95, 125);
    pDC->LineTo(230, 125);
    pDC->LineTo(230, 225);
    pDC->LineTo(95, 325);
    CMFCGDDIDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: add draw code for native data here }
```

**Step 7:** Run this application. You will see the following output.



## Polyline

A **polyline** is a series of connected lines. The lines are stored in an array of POINT or CPoint values. To draw a polyline, you use the CDC::Polyline() method. To draw a polyline, at least two points are required. If you define more than two points, each line after the first would be drawn from the previous point to the next point until all points have been included.

**Step 1:** Let us look into a simple example.

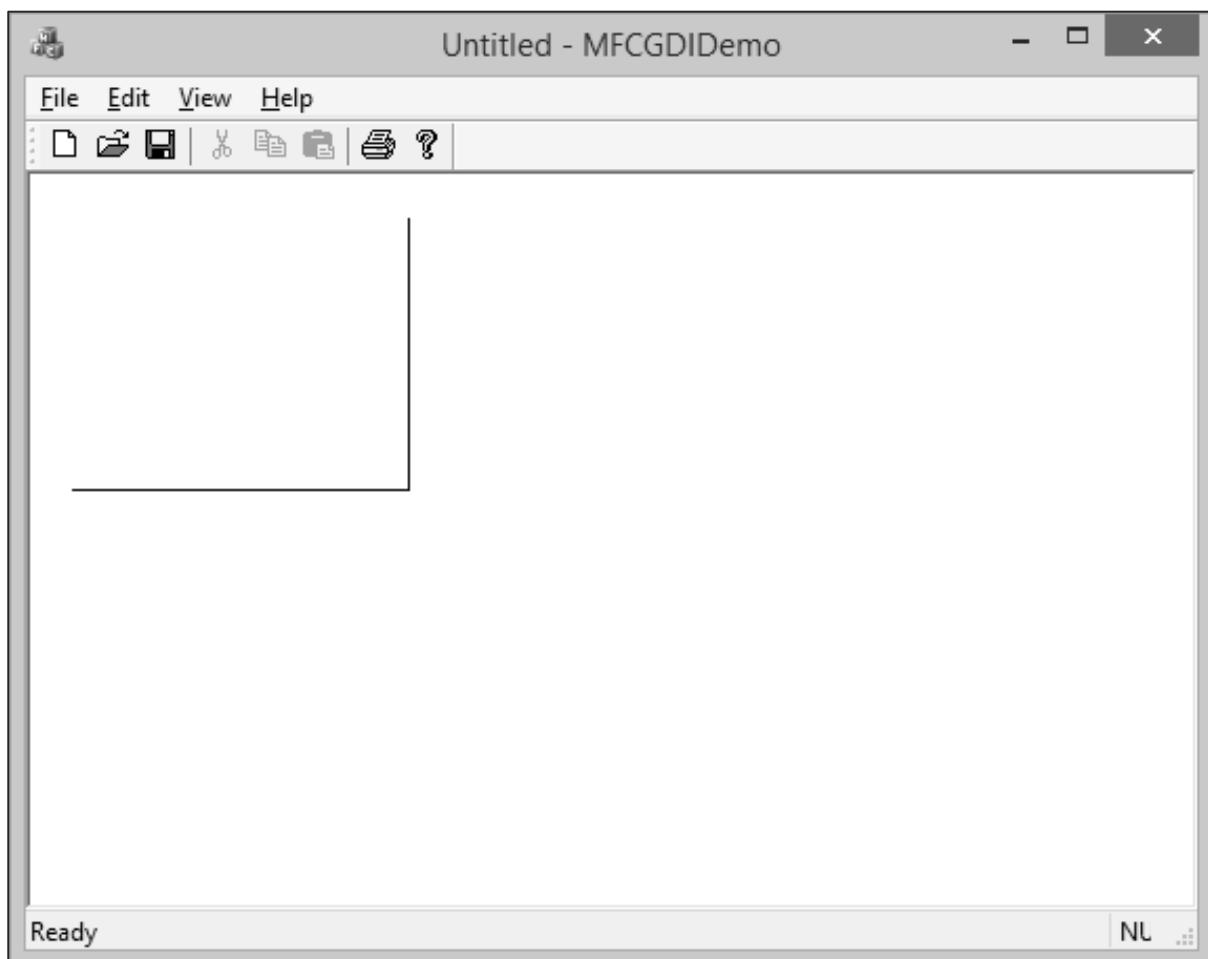
```
void CMFCGDDIDemoView::OnDraw(CDC* pDC)
{
    CPoint Pt[7];
    Pt[0] = CPoint(20, 150);
    Pt[1] = CPoint(180, 150);
    Pt[2] = CPoint(180, 20);
```

```
pDC->Polyline(Pt, 3);

CMFCGDI DemoDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
if (!pDoc)
    return;

// TODO: add draw code for native data here
```

**Step 2:** When you run this application, you will see the following output.



## Rectangles

A **rectangle** is a geometric figure made of four sides that compose four right angles. Like the line, to draw a rectangle, you must define where it starts and where it ends. To draw a rectangle, you can use the `CDC::Rectangle()` method.

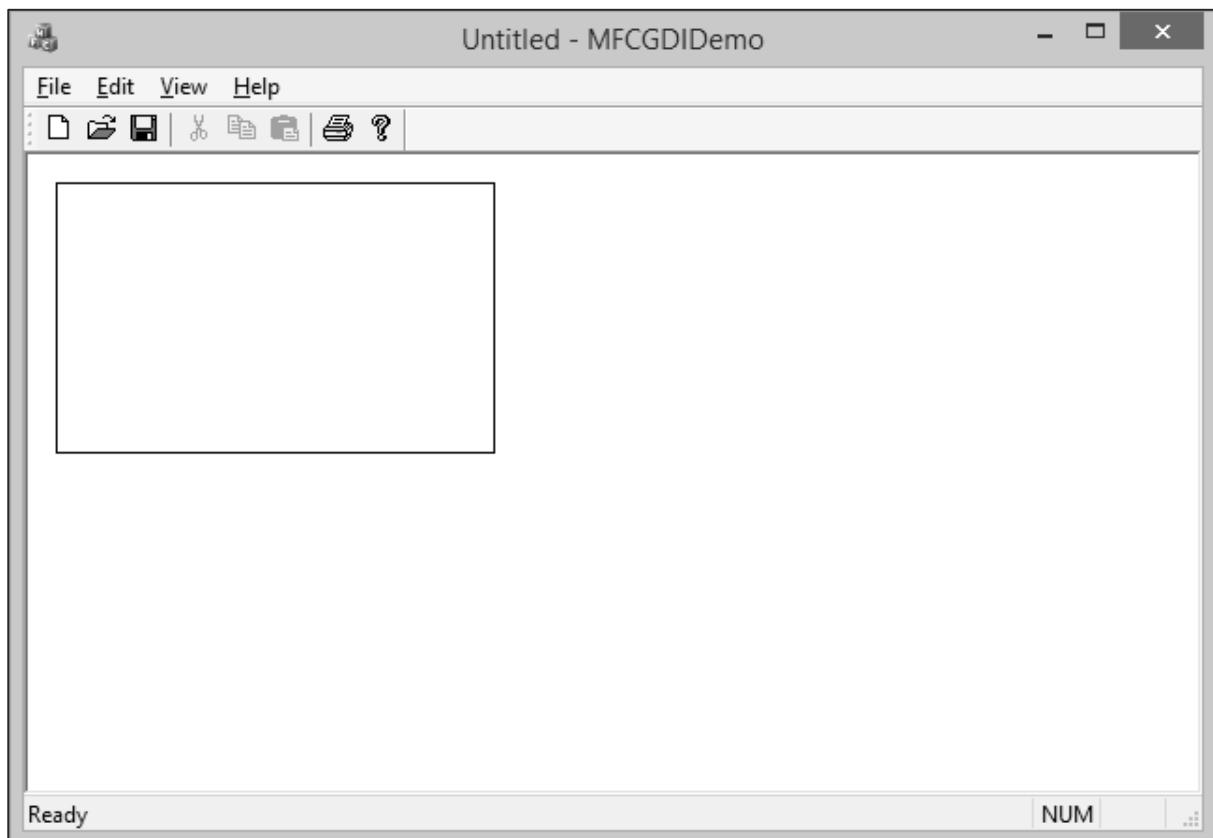
**Step 1:** Let us look into a simple example.

```
void CMFCGDDemoView::OnDraw(CDC* pDC)
{
    pDC->Rectangle(15, 15, 250, 160);

    CMFCGDDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

**Step 2:** When you run this application, you will see the following output.



## Squares

A **square** is a geometric figure made of four sides that compose four right angles, but each side must be equal in length.

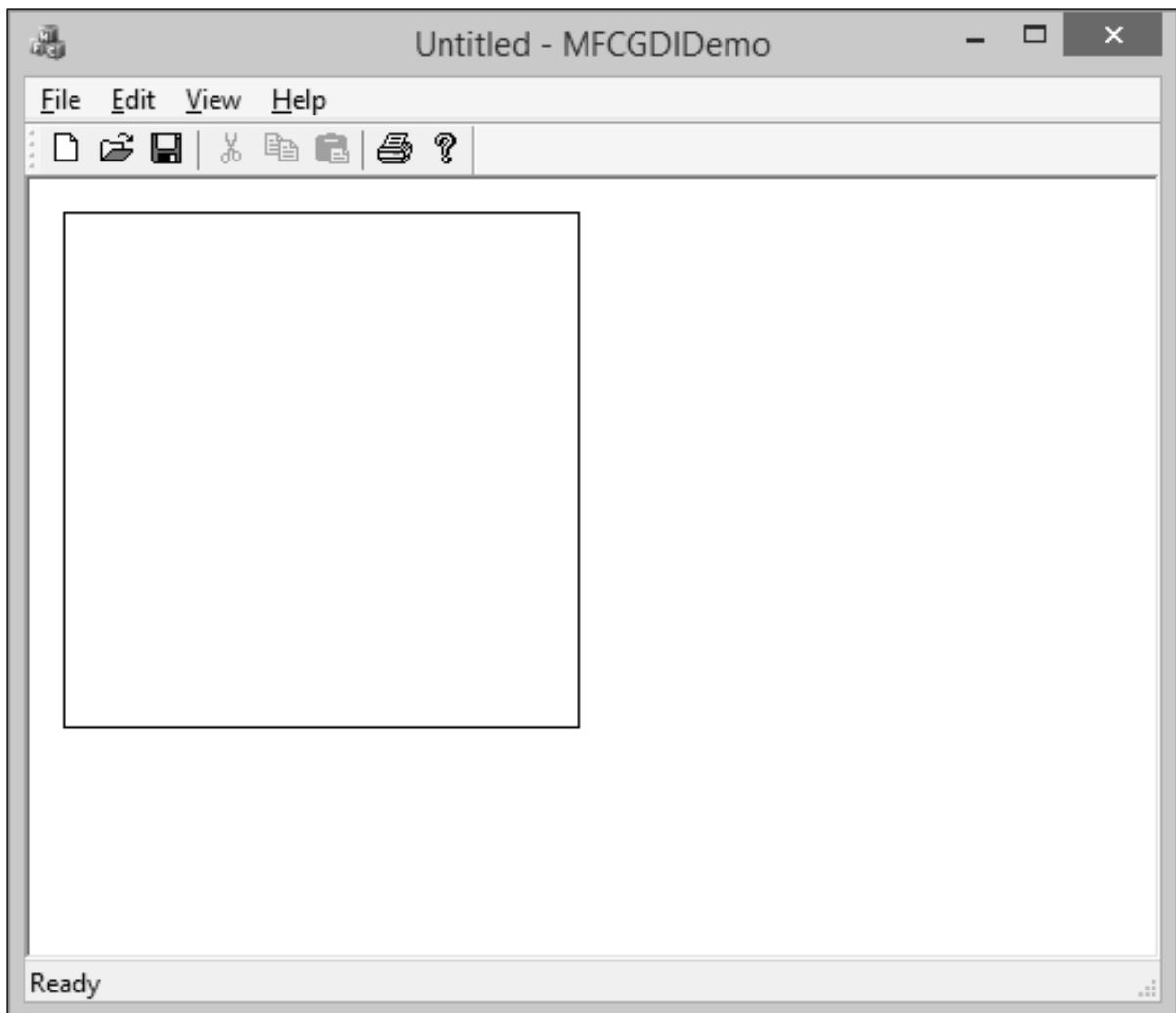
Let us look into a simple example.

```
void CMFCGDIView::OnDraw(CDC* pDC)
{
    pDC->Rectangle(15, 15, 250, 250);

    CMFCGDIViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

When you run this application, you will see the following output.



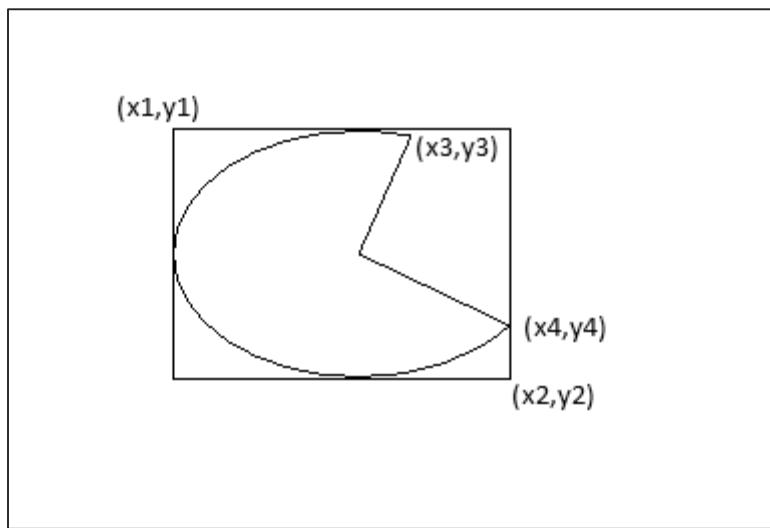
## Pies

---

A **pie** is a fraction of an ellipse delimited by two lines that span from the center of the ellipse to one side each. To draw a pie, you can use the `CDC::Pie()` method as shown below:

```
BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
```

- The  $(x_1, y_1)$  point determines the upper-left corner of the rectangle in which the ellipse that represents the pie fits. The  $(x_2, y_2)$  point is the bottom-right corner of the rectangle.



- The  $(x_3, y_3)$  point specifies the starting corner of the pie in a default counterclockwise direction.
- The  $(x_4, y_4)$  point specifies the end point of the pie.

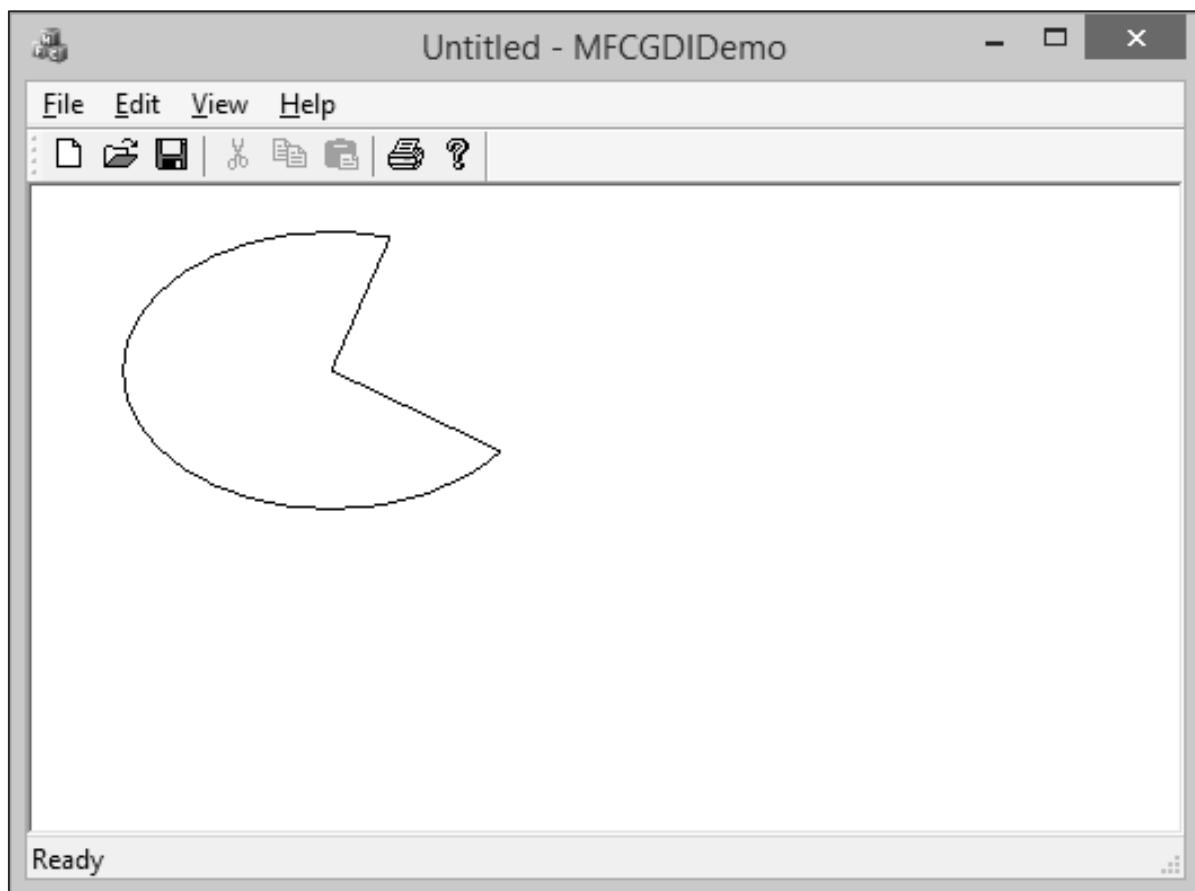
Let us look into a simple example.

```
void CMFCGDDemoView::OnDraw(CDC* pDC)
{
    pDC->Pie(40, 20, 226, 144, 155, 32, 202, 115);

    CMFCGDDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

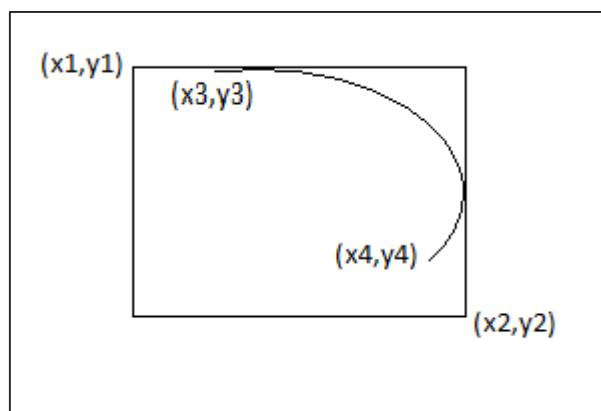
**Step 2:** When you run this application, you will see the following output.



## Arcs

An arc is a portion or segment of an ellipse, meaning an arc is a non-complete ellipse. To draw an arc, you can use the `CDC::Arc()` method.

```
BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
```



The `CDC` class is equipped with the `SetArcDirection()` method.

Here is the syntax:

```
int SetArcDirection(int nArcDirection)
```

Value	Orientation
AD_CLOCKWISE	The figure is drawn clockwise
AD_COUNTERCLOCKWISE	The figure is drawn counterclockwise

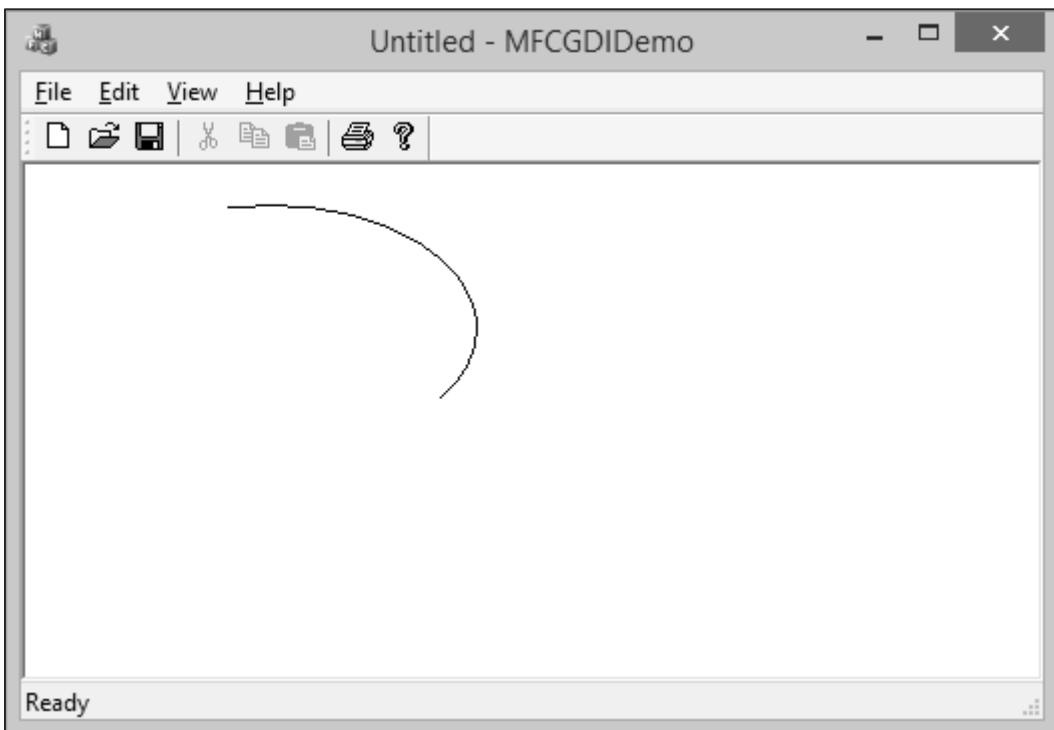
**Step 1:** Let us look into a simple example.

```
void CMFCGDIView::OnDraw(CDC* pDC)
{
    pDC->SetArcDirection(AD_COUNTERCLOCKWISE);
    pDC->Arc(20, 20, 226, 144, 202, 115, 105, 32);

    CMFCGDIView* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

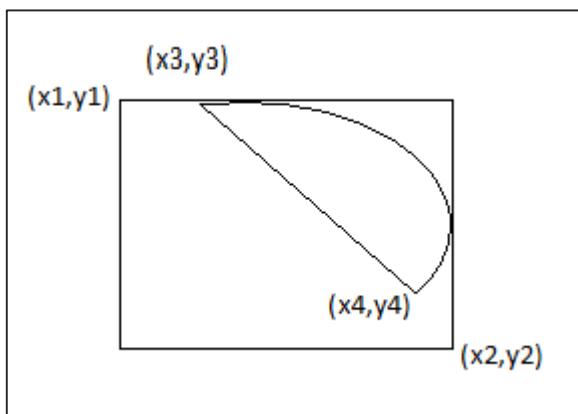
    // TODO: add draw code for native data here
}
```

**Step 2:** When you run this application, you will see the following output.



## Chords

The arcs we have drawn so far are considered open figures because they are made of a line that has a beginning and an end (unlike a circle or a rectangle that do not). A **chord** is an arc whose two ends are connected by a straight line.



To draw a chord, you can use the `CDC::Chord()` method.

```
BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
```

Let us look into a simple example.

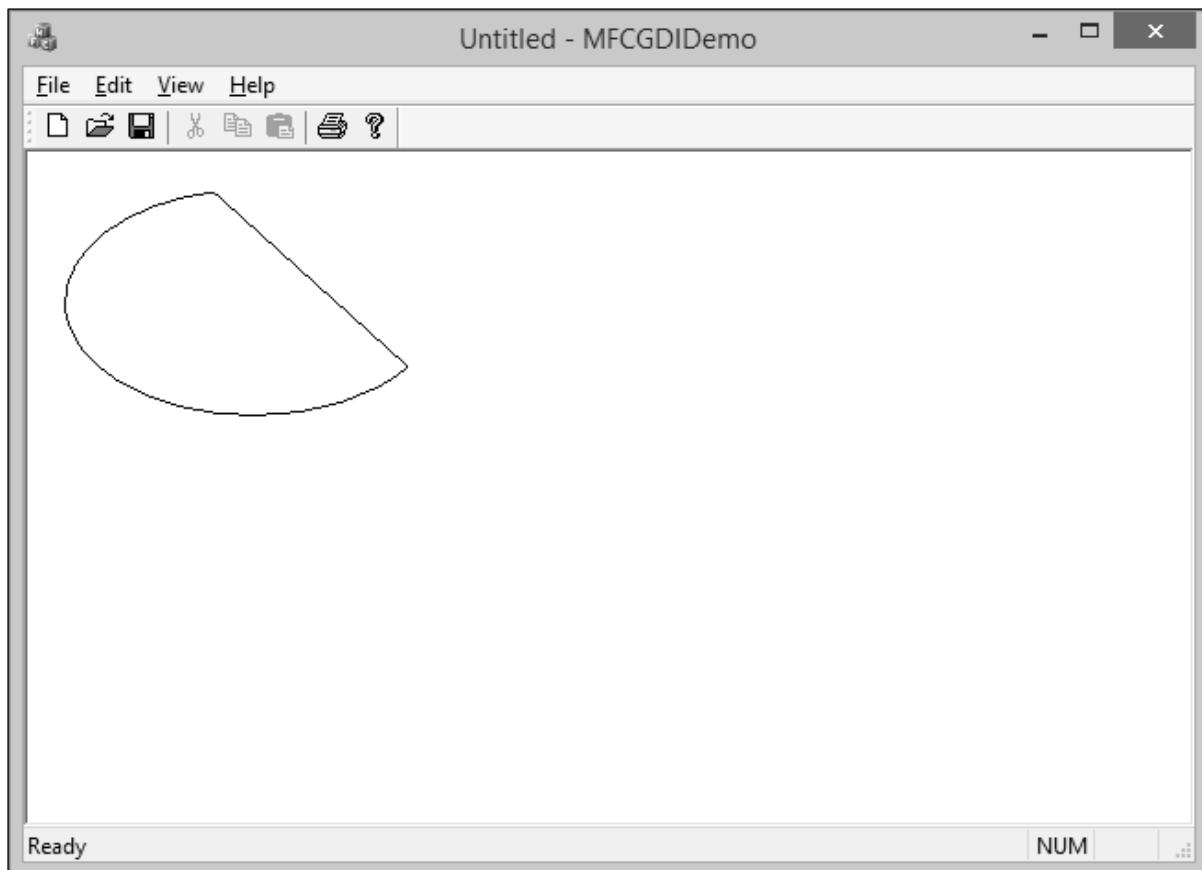
```
void CMFCGDDIDemoView::OnDraw(CDC* pDC)
{
    pDC->SetArcDirection(AD_CLOCKWISE);
    pDC->Chord(20, 20, 226, 144, 202, 115, 105, 32);
```

398

```
CMFCGDDemoDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
if (!pDoc)
    return;

// TODO: add draw code for native data here
}
```

When you run the above application, you will see the following output.



The arc direction in this example is set clockwise.

## Colors

The **color** is one the most fundamental objects that enhances the aesthetic appearance of an object. The color is a non-spatial object that is added to an object to modify some of its visual aspects. The MFC library, combined with the Win32 API, provides various actions you can use to take advantage of the various aspects of colors.

The RGB macro behaves like a function and allows you to pass three numeric values separated by a comma. Each value must be between 0 and 255 as shown in the following code.

```
void CMFCGDIView::OnDraw(CDC* pDC)
{
    COLORREF color = RGB(239, 15, 225);
}
```

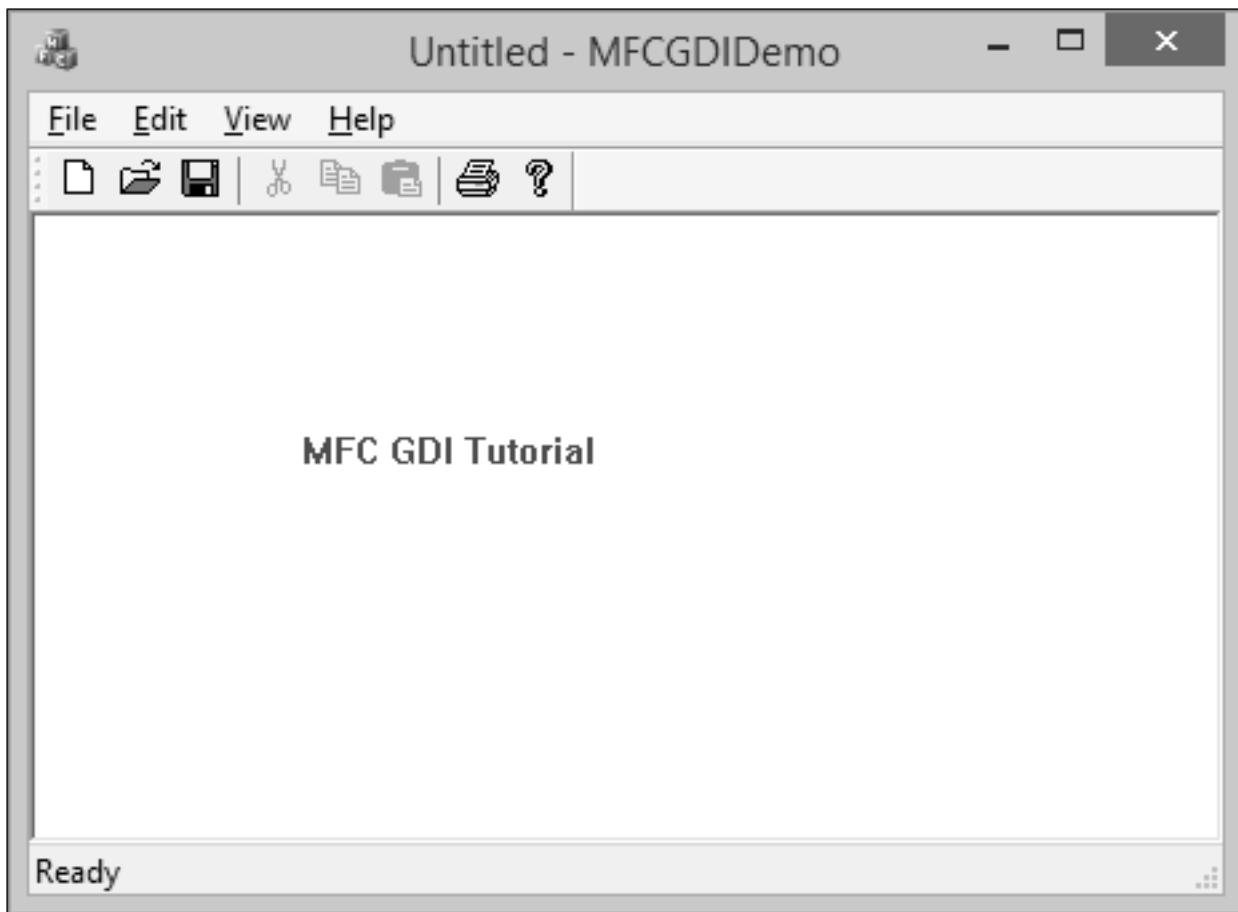
Let us look into a simple example.

```
void CMFCGDIView::OnDraw(CDC* pDC)
{
    COLORREF color = RGB(239, 15, 225);
    pDC->SetTextColor(color);
    pDC->TextOut(100, 80, L"MFC GDI Tutorial", 16);

    CMFCGDIViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

When you run this application, you will see the following output.



## Fonts

**CFont** encapsulates a Windows graphics device interface (GDI) font and provides member functions for manipulating the font. To use a CFont object, construct a CFont object and attach a Windows font to it, and then use the object's member functions to manipulate the font.

Here is a list of methods in CFont class.

Name	Description
<b>CreateFont</b>	Initializes a CFont with the specified characteristics.
<b>CreateFontIndirect</b>	Initializes a CFont object with the characteristics given in a <b>LOGFONT</b> structure.
<b>CreatePointFont</b>	Initializes a CFont with the specified height, measured in tenths of a point, and typeface.
<b>CreatePointFontIndirect</b>	Same as <b>CreateFontIndirect</b> except that the font height is measured in tenths of a point rather than logical units.
<b>FromHandle</b>	Returns a pointer to a CFont object when given a Windows <b>HFONT</b> .
<b>GetLogFont</b>	Fills a LOGFONT with information about the logical font attached to the CFont object.

Let us look into a simple example.

```
void CMFCGDIView::OnDraw(CDC* pDC)
{
    CFont font;
    font.CreatePointFont(920, L"Garamond");
    CFont *pFont = pDC->SelectObject(&font);
    COLORREF color = RGB(239, 15, 225);
    pDC->SetTextColor(color);
    pDC->TextOut(100, 80, L"MFC GDI Tutorial", 16);
    pDC->SelectObject(pFont);
    font.DeleteObject();

    CMFCGDIViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

When you run the above application, you will see the following output.



## Pens

A **pen** is a tool used to draw lines and curves on a device context. In the graphics programming, a pen is also used to draw the borders of a geometric closed shape such as a rectangle or a polygon. Microsoft Windows considers two types of pens — **cosmetic** and **geometric**.

A pen is referred to as cosmetic when it can be used to draw only simple lines of a fixed width, less than or equal to 1 pixel. A pen is geometric when it can assume different widths and various ends. MFC provides a class **CPen** which encapsulates a Windows graphics device interface (GDI) pen.

Here is a list of methods in CPen class.

Name	Description
<b>CreatePen</b>	Creates a logical cosmetic or geometric pen with the specified style, width, and brush attributes, and attaches it to the CPen object.
<b>CreatePenIndirect</b>	Creates a pen with the style, width, and color given in a LOGPEN structure, and attaches it to the CPen object.
<b>FromHandle</b>	Returns a pointer to a CPen object when given a Windows <b>HPEN</b> .
<b>GetExtLogPen</b>	Gets an EXTLOGOPEN underlying structure.
<b>GetLogPen</b>	Gets a <a href="#">LOGOPEN</a> underlying structure.

Here the different style values for pen:

Value	Description
<b>PS_SOLID</b>	A continuous solid line.
<b>PS_DASH</b>	A continuous line with dashed interruptions.
<b>PS_DOT</b>	A line with a dot interruption at every other pixel.
<b>PS_DASHDOT</b>	A combination of alternating dashed and dotted points.
<b>PS_DASHDOTDOT</b>	A combination of dash and double dotted interruptions.
<b>PS_NULL</b>	No visible line.
<b>PS_INSIDEFRAME</b>	A line drawn just inside of the border of a closed shape.

Let us look into a simple example.

```
void CMFCGDDemoView::OnDraw(CDC* pDC)
{
    CPen pen;
    pen.CreatePen(PS_DASHDOTDOT, 1, RGB(160, 75, 90));
    pDC->SelectObject(&pen);
    pDC->Rectangle(25, 35, 250, 125);
```

```
CMFCGDI DemoDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
if (!pDoc)
    return;

// TODO: add draw code for native data here
}
```

When you run the above application, you will see the following output.



## Brushes

A **brush** is a drawing tool used to fill out closed shaped or the interior of lines. A brush behaves like picking up a bucket of paint and pouring it somewhere. MFC provides a class **CBrush** which encapsulates a Windows graphics device interface (GDI) brush.

Here is a list of methods in CBrush class.

Name	Description
<b>CreateBrushIndirect</b>	Initializes a brush with the style, color, and pattern specified in a LOGBRUSH structure.
<b>CreateDIBPatternBrush</b>	Initializes a brush with a pattern specified by a device-independent bitmap (DIB).
<b>CreateHatchBrush</b>	Initializes a brush with the specified hatched pattern and color.
<b>CreatePatternBrush</b>	Initializes a brush with a pattern specified by a bitmap.
<b>CreateSolidBrush</b>	Initializes a brush with the specified solid color.
<b>CreateSysColorBrush</b>	Creates a brush that is the default system color.
<b>FromHandle</b>	Returns a pointer to a CBrush object when given a handle to a Windows HBRUSH object.
<b>GetLogBrush</b>	Gets a LOGBRUSH structure.

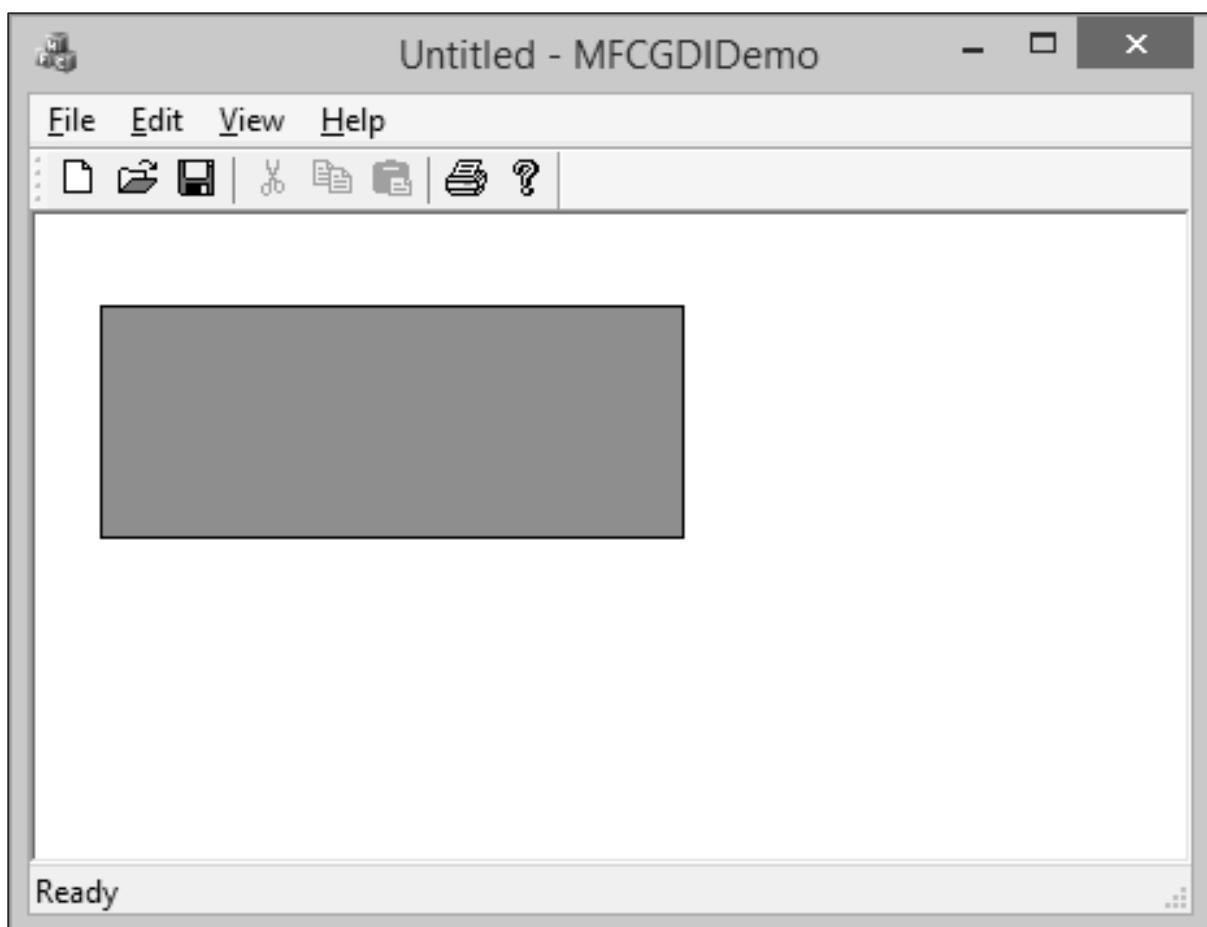
Let us look into a simple example.

```
void CMFCGDDemoView::OnDraw(CDC* pDC)
{
    CBrush brush(RGB(100, 150, 200));
    CBrush *pBrush = pDC->SelectObject(&brush);
    pDC->Rectangle(25, 35, 250, 125);
    pDC->SelectObject(pBrush);

    CMFCGDDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

When you run this application, you will see the following output.



# 25. MFC - Libraries

A **library** is a group of functions, classes, or other resources that can be made available to programs that need already implemented entities without the need to know how these functions, classes, or resources were created or how they function. A library makes it easy for a programmer to use functions, classes, and resources etc. created by another person or company and trust that this external source is reliable and efficient. Some unique features related to libraries are:

- A library is created and functions like a normal regular program, using functions or other resources and communicating with other programs.
- To implement its functionality, a library contains functions that other programs would need to complete their functionality.
- At the same time, a library may use some functions that other programs would not need.
- The program that uses the library, are also called the clients of the library.

There are two types of functions you will create or include in your libraries:

- An internal function is one used only by the library itself and clients of the library will not need access to these functions.
- External functions are those that can be accessed by the clients of the library.

There are two broad categories of libraries you will deal with in your programs:

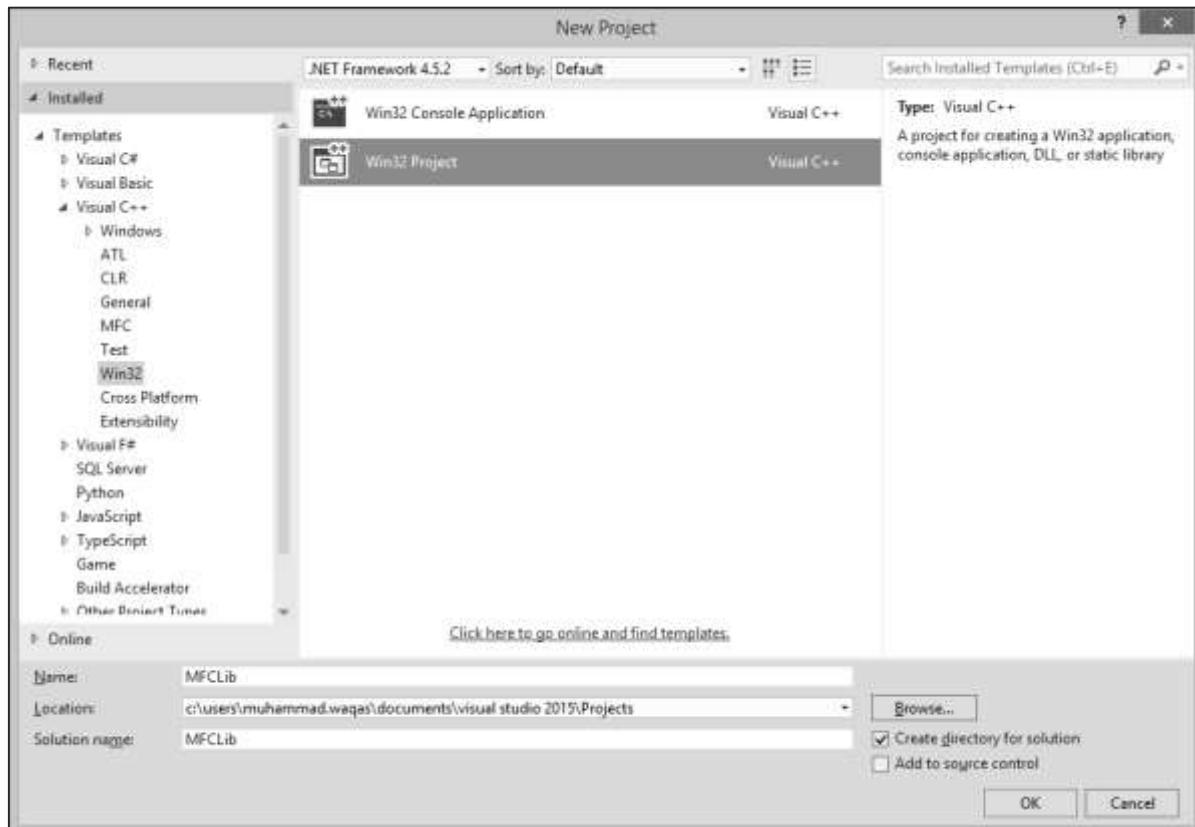
- Static libraries
- Dynamic libraries

## Static Library

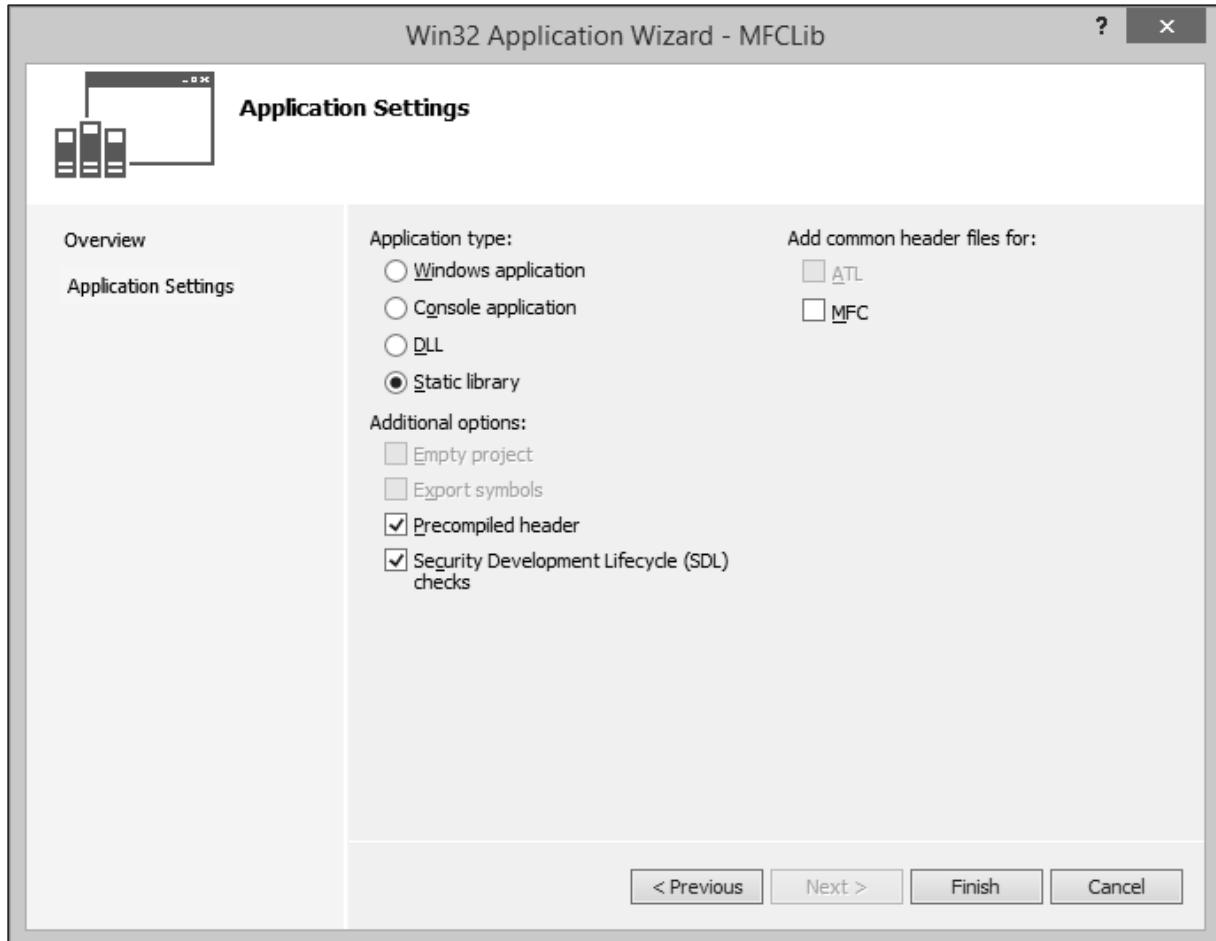
---

A **static library** is a file that contains functions, classes, or resources that an external program can use to complement its functionality. To use a library, the programmer has to create a link to it. The project can be a console application, a Win32 or an MFC application. The library file has the lib extension.

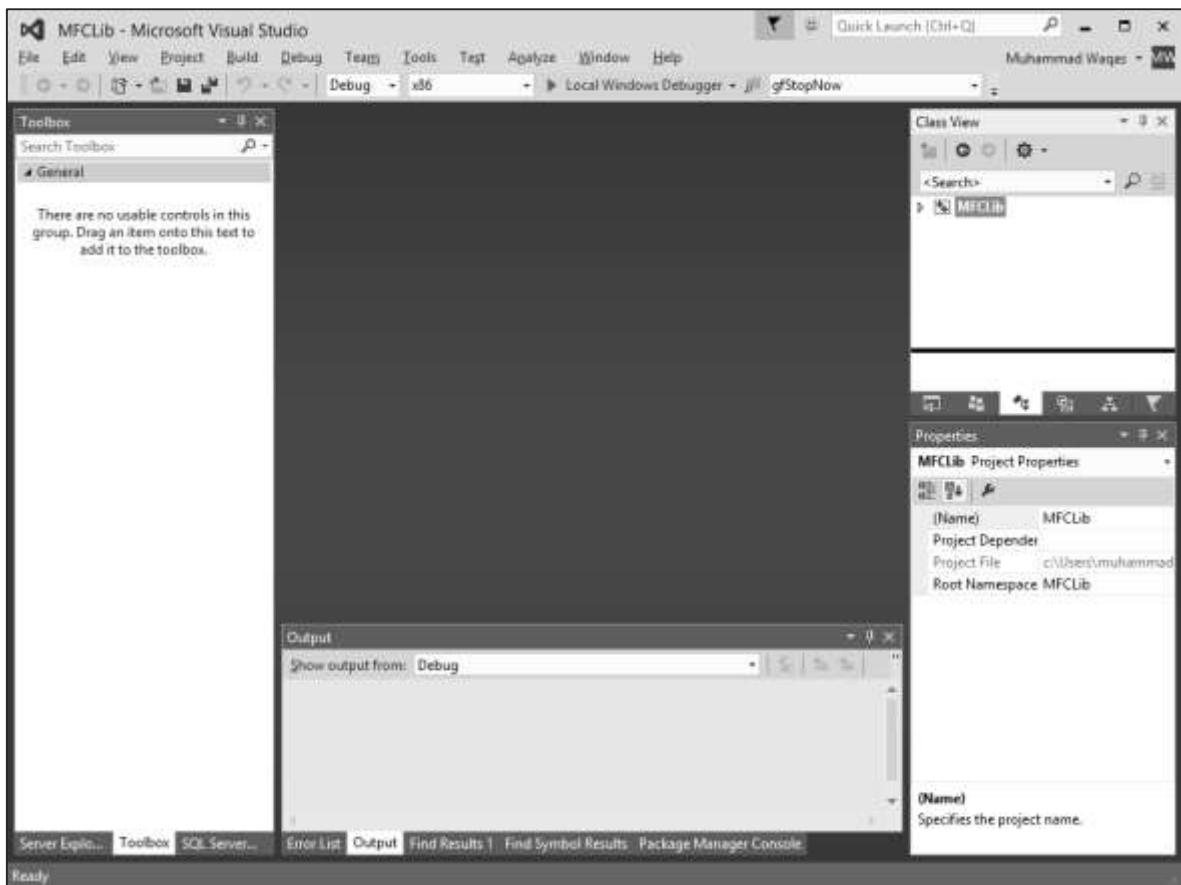
**Step 1:** Let us look into a simple example of static library by creating a new Win32 Project.



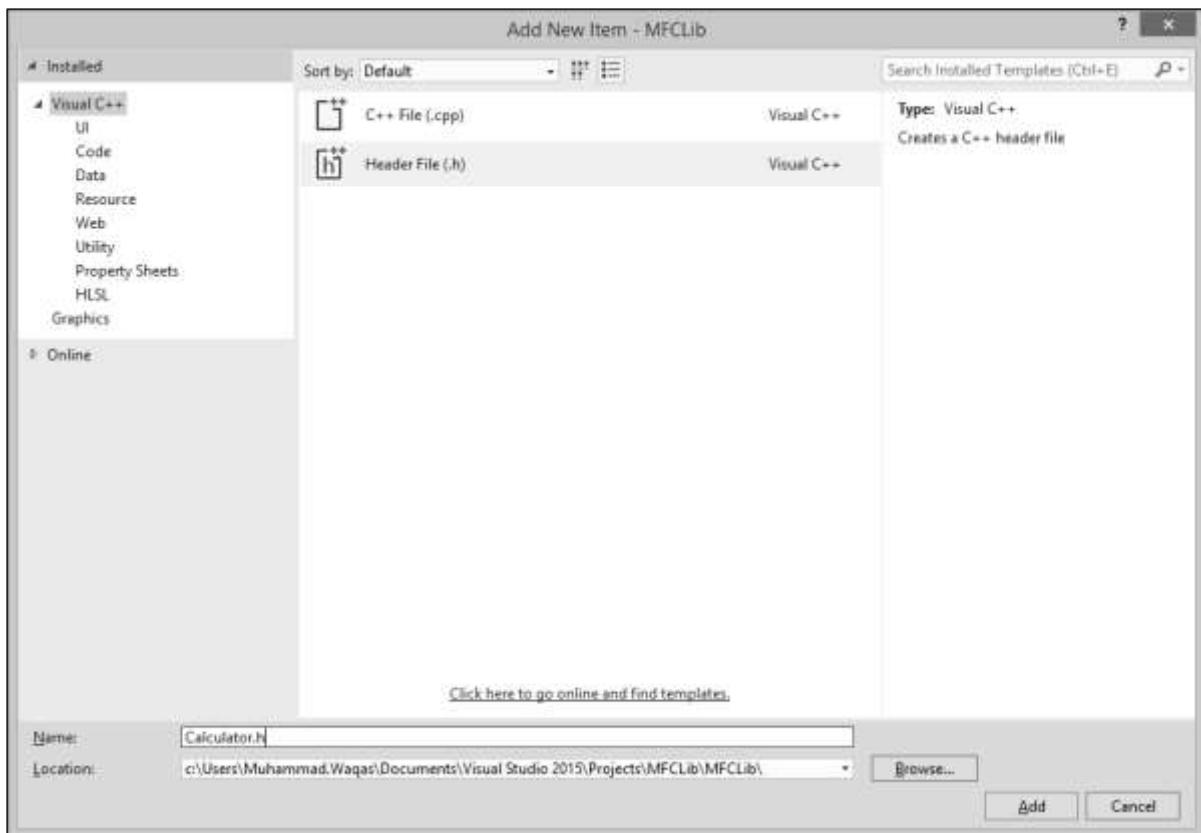
**Step 2:** On Application Wizard dialog box, choose the Static Library option.



**Step 3:** Click Finish to continue.



**Step 4:** Right-click on the project in solution explorer and add a header file from Add -> New Item...menu option.



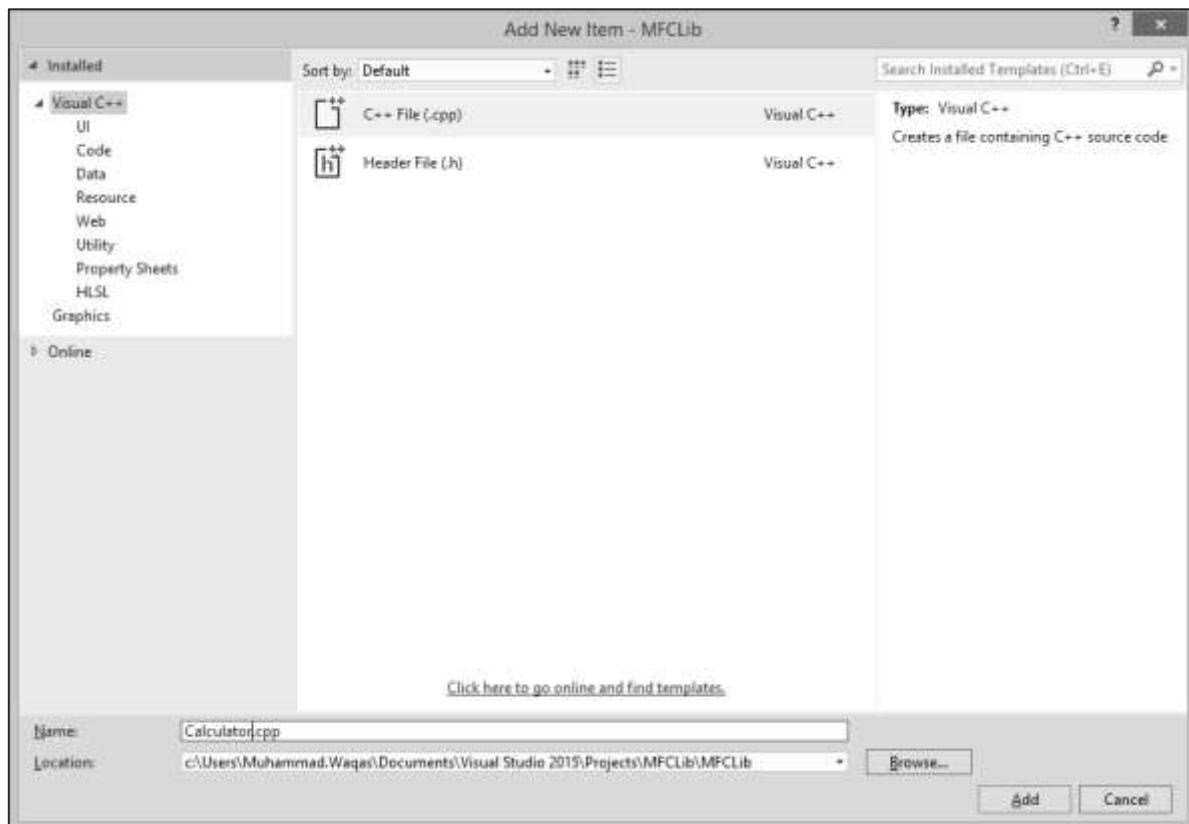
**Step 5:** Enter Calculator.h in the Name field and click Add.

Add the following code in the header file:

```
#pragma once
#ifndef _CALCULATOR_H_
#define _CALCULATOR_H_

double Min(const double *Numbers, const int Count);
double Max(const double *Numbers, const int Count);
double Sum(const double *Numbers, const int Count);
double Average(const double *Numbers, const int Count);
long GreatestCommonDivisor(long Nbr1, long Nbr2);
#endif // _CALCULATOR_H_
```

**Step 6:** Add a source (\*.cpp) file in the project.



**Step 7:** Enter Calculator.cpp in the Name field and click Add.

**Step 8:** Add the following code in the \*.cpp file:

```
#include "StdAfx.h"
#include "Calculator.h"

double Min(const double *Nbr, const int Total)
{
    double Minimum = Nbr[0];
    for (int i = 0; i < Total; i++)
        if (Minimum > Nbr[i])
            Minimum = Nbr[i];
    return Minimum;
}

double Max(const double *Nbr, const int Total)
{
    double Maximum = Nbr[0];
    for (int i = 0; i < Total; i++)
        if (Maximum < Nbr[i])
            Maximum = Nbr[i];
    return Maximum;
}
```

```
}

double Sum(const double *Nbr, const int Total)
{
    double S = 0;
    for (int i = 0; i < Total; i++)
        S += Nbr[i];
    return S;
}

double Average(const double *Nbr, const int Total)
{
    double avg, S = 0;
    for (int i = 0; i < Total; i++)
        S += Nbr[i];
    avg = S / Total;
    return avg;
}

long GreatestCommonDivisor(long Nbr1, long Nbr2)
{
    while (true)
    {
        Nbr1 = Nbr1 % Nbr2;
        if (Nbr1 == 0)
            return Nbr2;
        Nbr2 = Nbr2 % Nbr1;
        if (Nbr2 == 0)
            return Nbr1;
    }
}
```

**Step 9:** Build this library from the main menu, by clicking **Build -> Build MFCLib.**

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** MFCLib - Microsoft Visual Studio
- Toolbox:** General
- Code Editor:** Calculator.cpp (Calculator.h) - Global Scope
 

```

double Average(const double *Nbr, const int Total)
{
    double avg, S = 0;
    for (int i = 0; i < Total; i++)
        S += Nbr[i];
    avg = S / Total;
    return avg;
}

long GreatestCommonDivisor(long Nbr1, long Nbr2)
{
    while (true)
    {
        Nbr1 = Nbr1 % Nbr2;
        if (Nbr1 == 0)
            return Nbr2;
        Nbr2 = Nbr2 % Nbr1;
        if (Nbr2 == 0)
            return Nbr1;
    }
}
      
```
- Solution Explorer:** Solution 'MFCLib' (1 proj)
  - MFCLib
    - External Dependencies
    - Header Files
      - Calculator.h
      - stdafx.h
      - targetver.h
    - References
- Properties:** Max VCCodeFunction
 

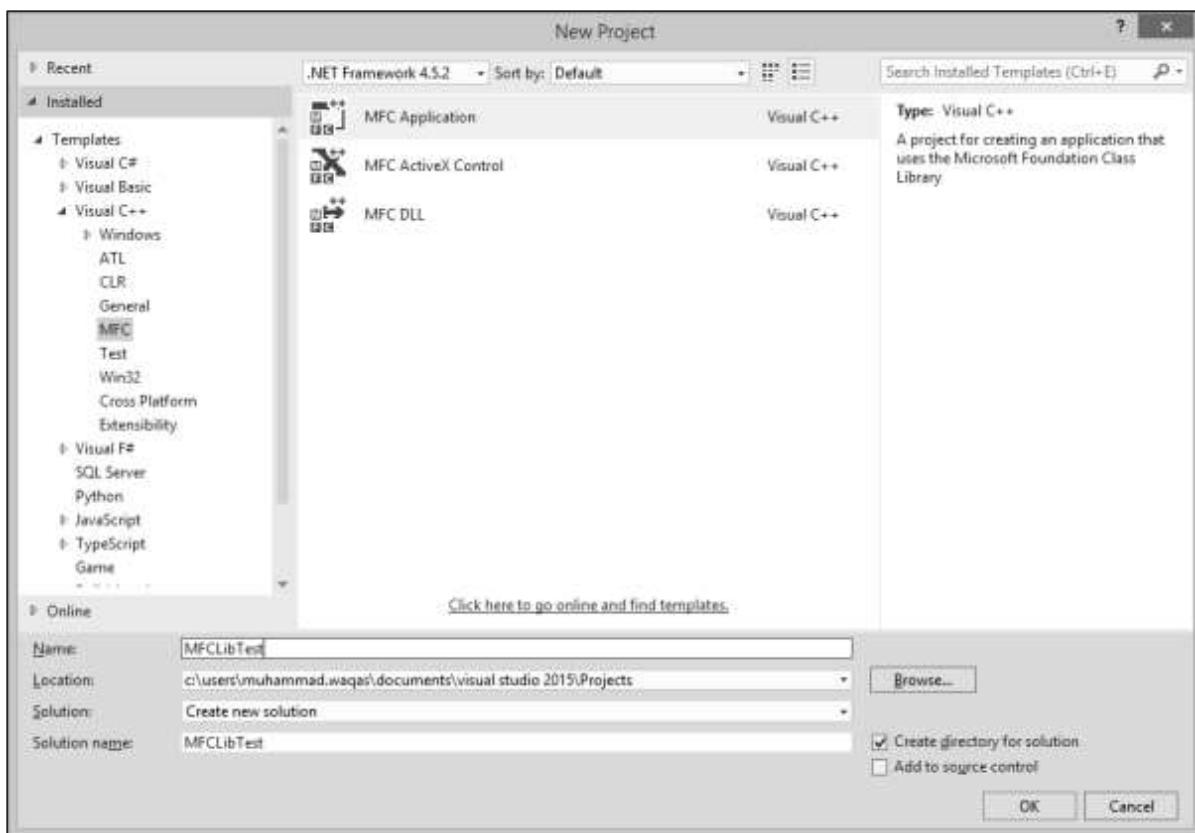
Name	Max
File	c:\Users\Muhan...
FullName	Max
IsDefault	False
IsDelete	False
IsFinal	False
IsInjected	False
IsInline	False
IsOverloader	False
IsSealed	False
IsTemplate	False
TypeString	double
- Output Window:**

```

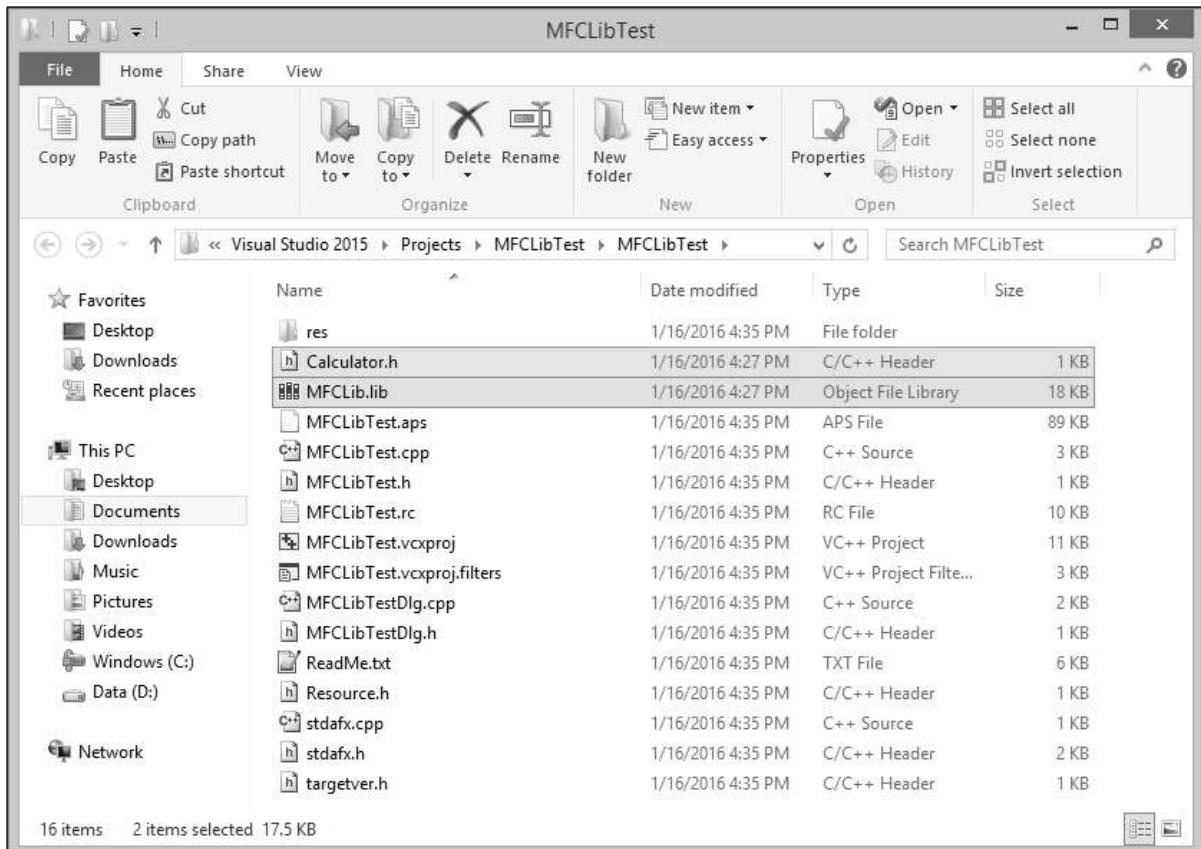
Show output from: Build
3>----- Build started: Project: MFCLib, Configuration: Debug Win32 -----
3>>> Calculator.cpp
3>>> MFCLib.vcxproj -> c:\Users\Muhammad.Waqas\Documents\Visual Studio 2015\Proj...
----- Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped -----
      
```

**Step 10:** When library is built successfully, it will display the above message.

**Step 11:** To use these functions from the library, let us add another MFC dialog application based from File -> New -> Project.

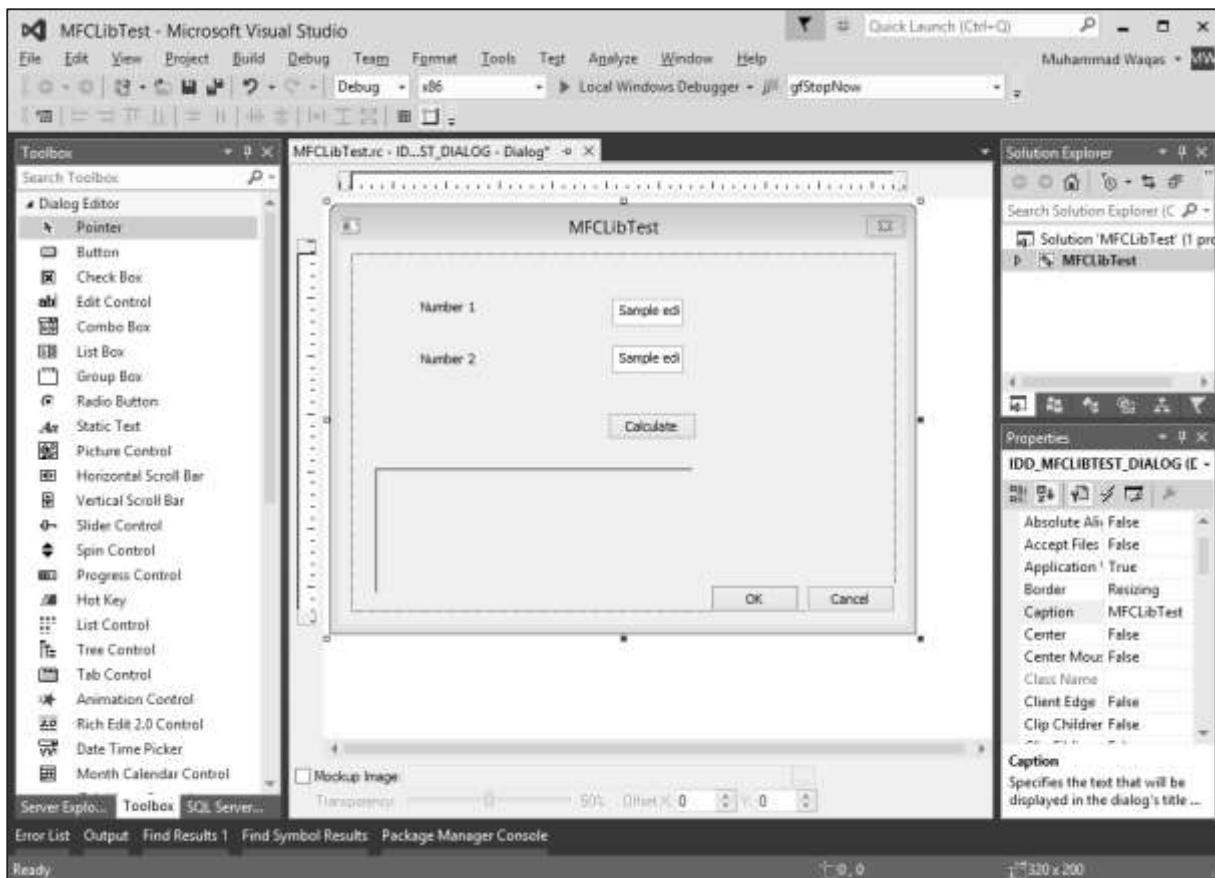


**Step 12:** Go to the MFCLib\Debug folder and copy the header file and \*.lib files to the MFCLibTest project as shown in the following snapshot.



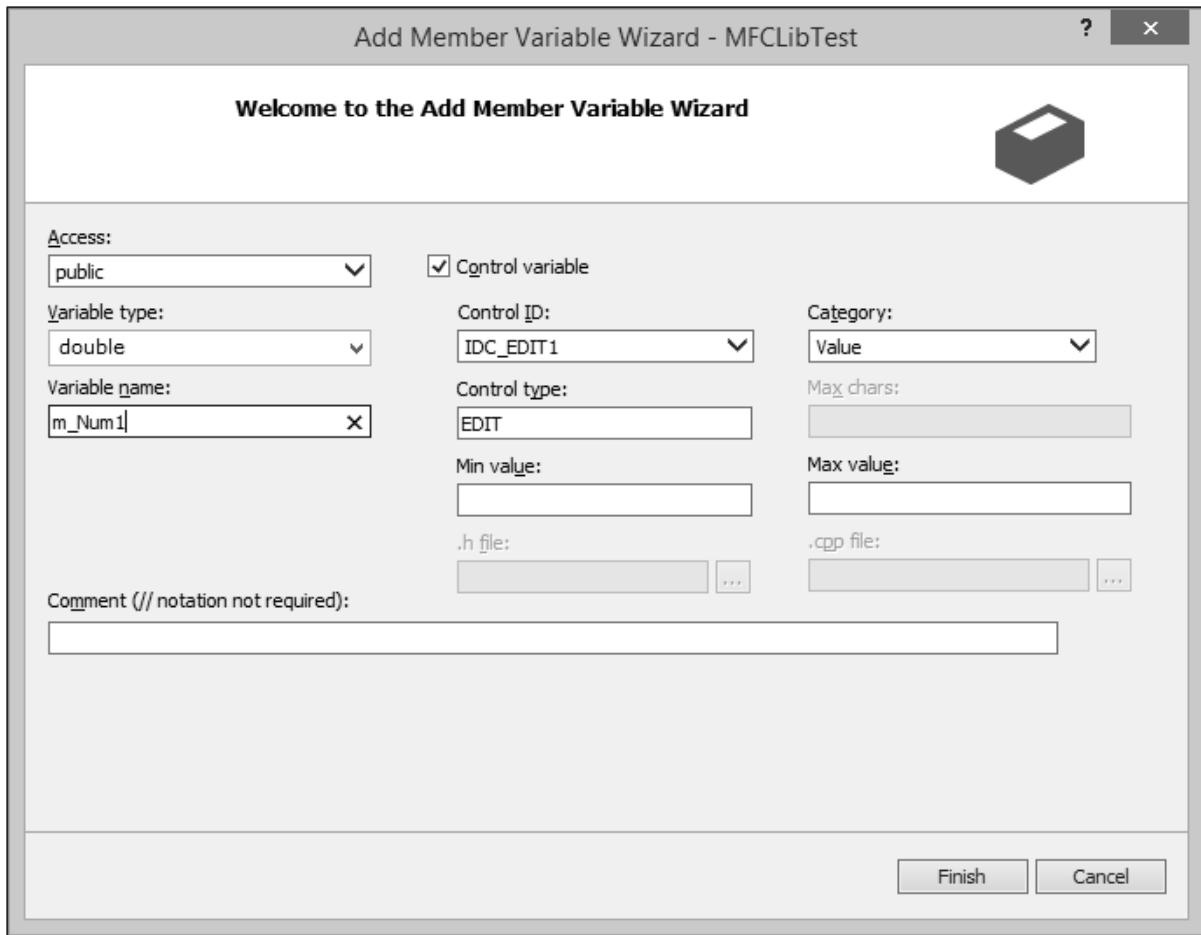
**Step 13:** To add the library to the current project, on the main menu, click Project -> Add Existing Item and select MFCLib.lib.

**Step 14:** Design your dialog box as shown in the following snapshot.

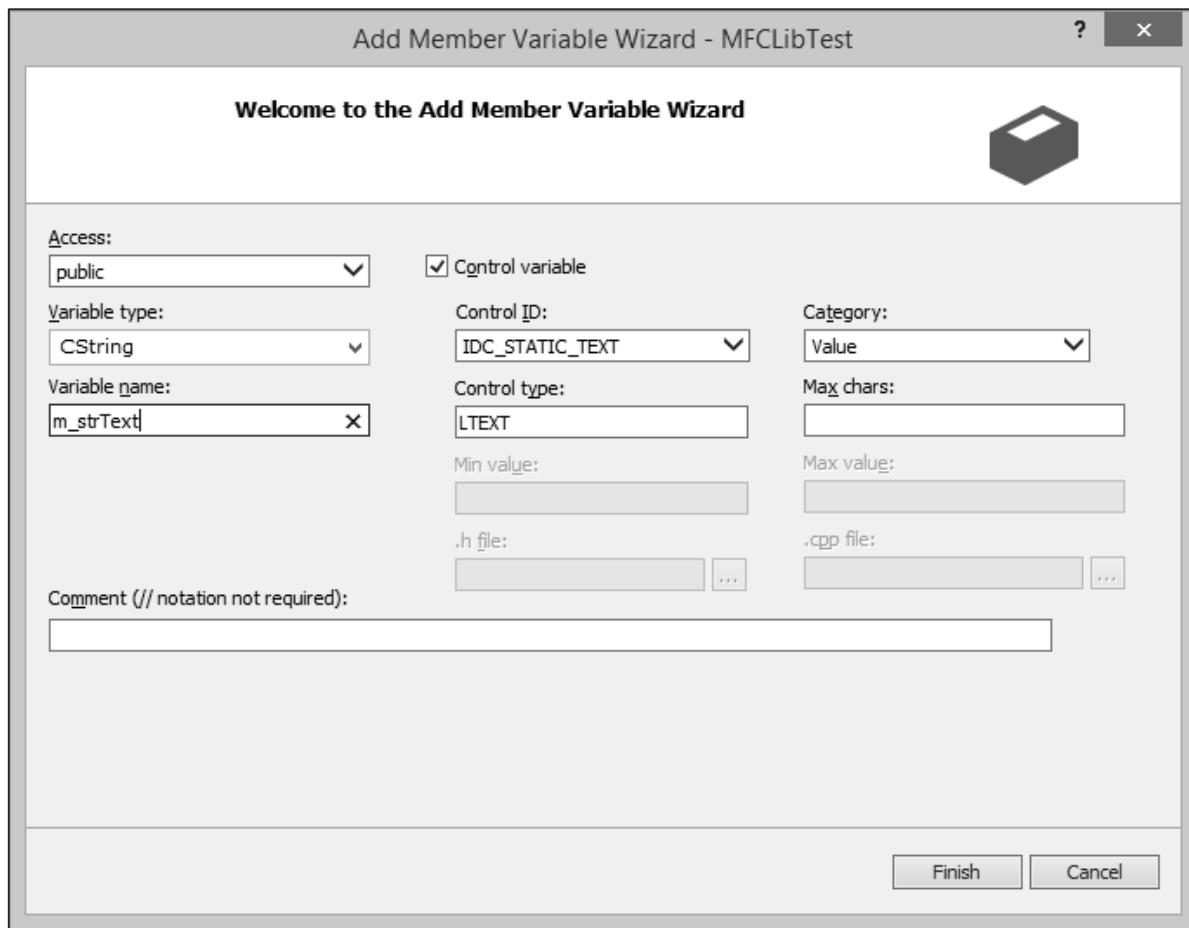


**Step 15:** Add value variable for both edit controls of value type double.

417



**Step 16:** Add value variable for Static text control, which is at the end of the dialog box.



**Step 17:** Add the event handler for Calculate button.

To add functionality from the library, we need to include the header file in CMFCLibTestDlg.cpp file.

```
#include "stdafx.h"
#include "MFCLibTest.h"
#include "MFCLibTestDlg.h"
#include "afxdialogex.h"
#include "Calculator.h"
```

**Step 18:** Here is the implementation of button event handler.

```
void CMFCLibTestDlg::OnBnClickedButtonCal()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    CString strTemp;
```

```

double numbers[2];
numbers[0] = m_Num1;
numbers[1] = m_Num2;
strTemp.Format(L"%,.2f", Max(numbers,2));
m_strText.Append(L"Max is:\t" + strTemp);

strTemp.Format(L"%,.2f", Min(numbers, 2));
m_strText.Append(L"\nMin is:\t" + strTemp);

strTemp.Format(L"%,.2f", Sum(numbers, 2));
m_strText.Append(L"\nSum is:\t" + strTemp);

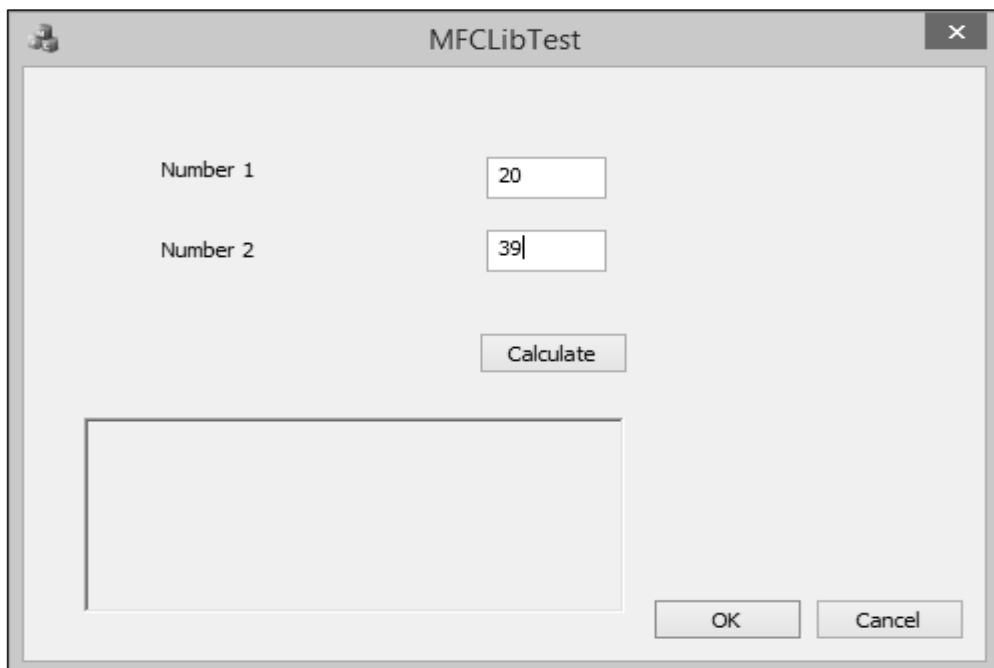
strTemp.Format(L"%,.2f", Average(numbers, 2));
m_strText.Append(L"\nAverage is:\t" + strTemp);

strTemp.Format(L"%d", GreatestCommonDivisor(m_Num1, m_Num2));
m_strText.Append(L"\nGDC is:\t" + strTemp);

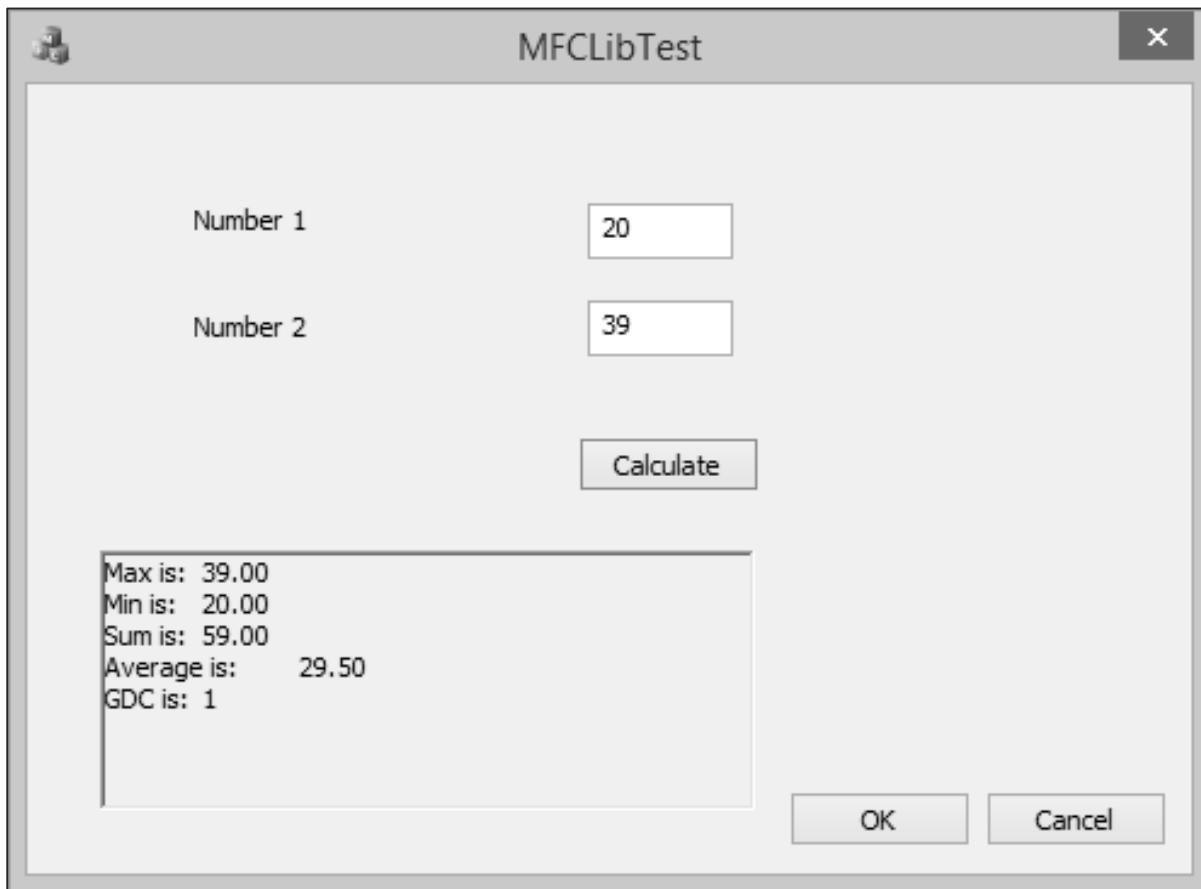
UpdateData(FALSE);
}

```

**Step 19:** When the above code is compiled and executed, you will see the following output.



**Step 20:** Enter two values in the edit field and click Calculate. You will now see the result after calculating from the library.



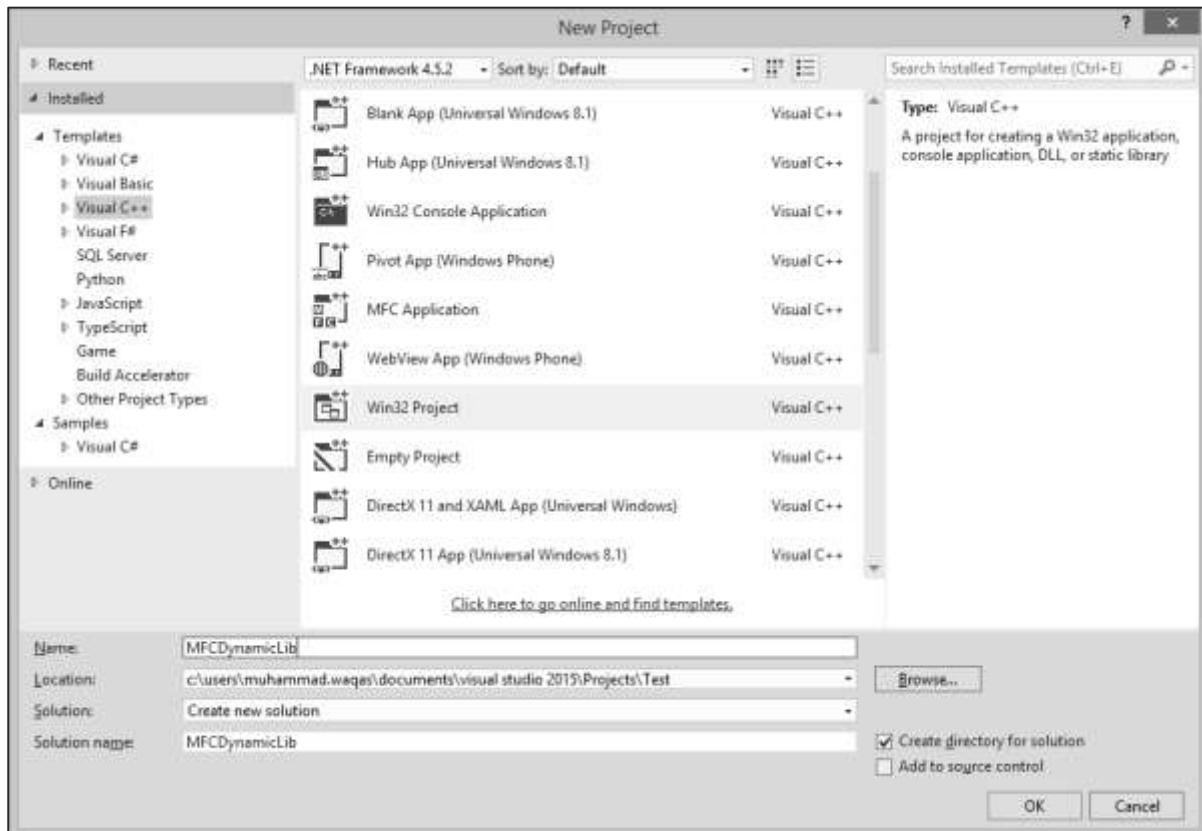
## Dynamic Library

A Win32 DLL is a library that can be made available to programs that run on a Microsoft Windows computer. As a normal library, it is made of functions and/or other resources grouped in a file.

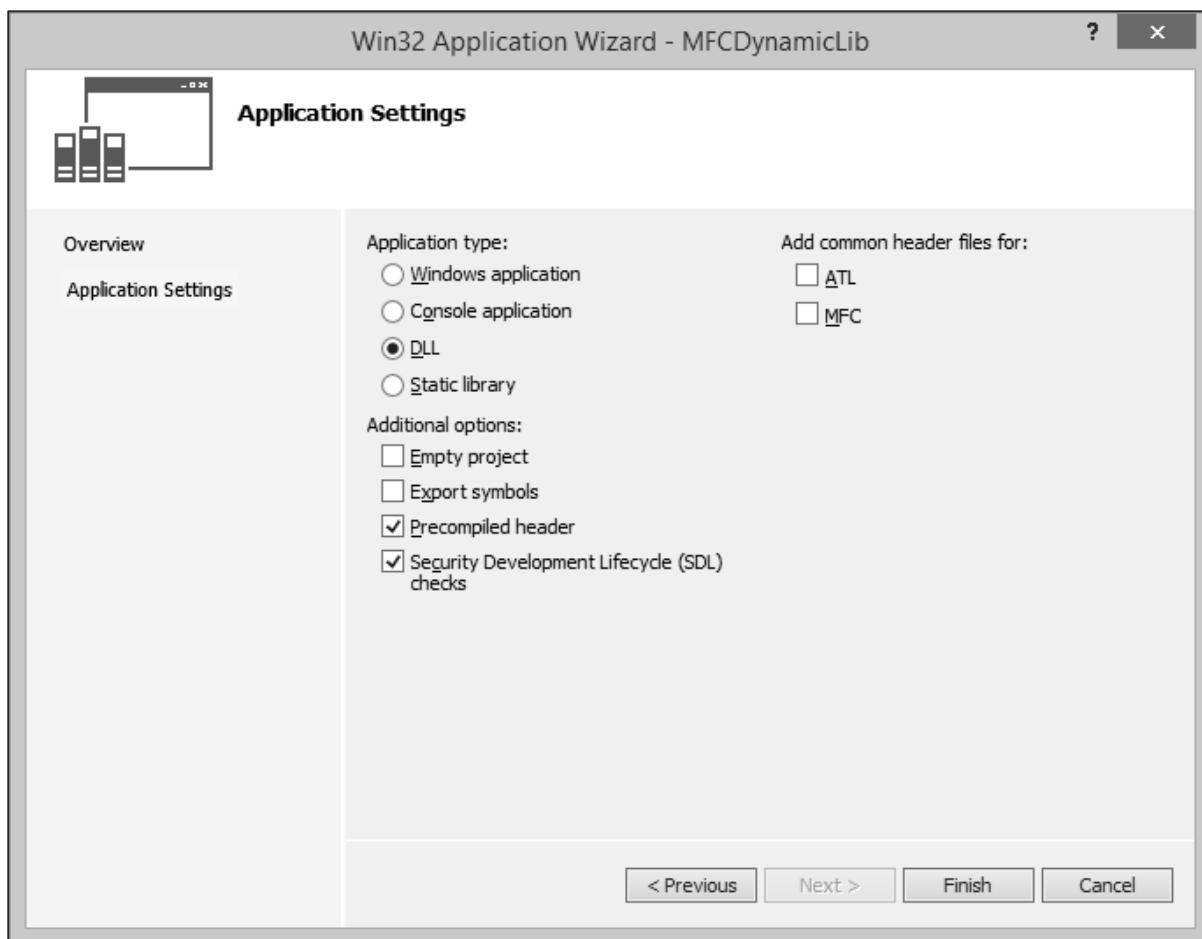
The DLL abbreviation stands for Dynamic Link Library. This means that, as opposed to a static library, a DLL allows the programmer to decide on when and how other applications will be linked to this type of library.

For example, a DLL allows different applications to use its library as they see fit and as necessary. In fact, applications created on different programming environments can use functions or resources stored in one particular DLL. For this reason, an application dynamically links to the library.

**Step 1:** Let us look into a simple example by creating a new Win32 Project.



**Step 2:** In the Application Type section, click the DLL radio button.



**Step 3:** Click Finish to continue.

**Step 4:** Add the following functions in MFCDynamicLib.cpp file and expose its definitions by using:

```
extern "C" __declspec(dllexport)
```

**Step 5:** Use the `_declspec(dllexport)` modifier for each function that will be accessed outside the DLL.

```
// MFCDynamicLib.cpp : Defines the exported functions for the DLL application.
//

#include "stdafx.h"

extern "C" __declspec(dllexport) double Min(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) double Max(const double *Numbers, const int Count);
```

```

extern "C" __declspec(dllexport) double Sum(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) double Average(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) long GreatestCommonDivisor(long Nbr1, long Nbr2);

double Min(const double *Nbr, const int Total)
{
    double Minimum = Nbr[0];
    for (int i = 0; i < Total; i++)
        if (Minimum > Nbr[i])
            Minimum = Nbr[i];
    return Minimum;
}

double Max(const double *Nbr, const int Total)
{
    double Maximum = Nbr[0];
    for (int i = 0; i < Total; i++)
        if (Maximum < Nbr[i])
            Maximum = Nbr[i];
    return Maximum;
}

double Sum(const double *Nbr, const int Total)
{
    double S = 0;
    for (int i = 0; i < Total; i++)
        S += Nbr[i];
    return S;
}

double Average(const double *Nbr, const int Total)
{
    double avg, S = 0;
    for (int i = 0; i < Total; i++)
        S += Nbr[i];
    avg = S / Total;
    return avg;
}

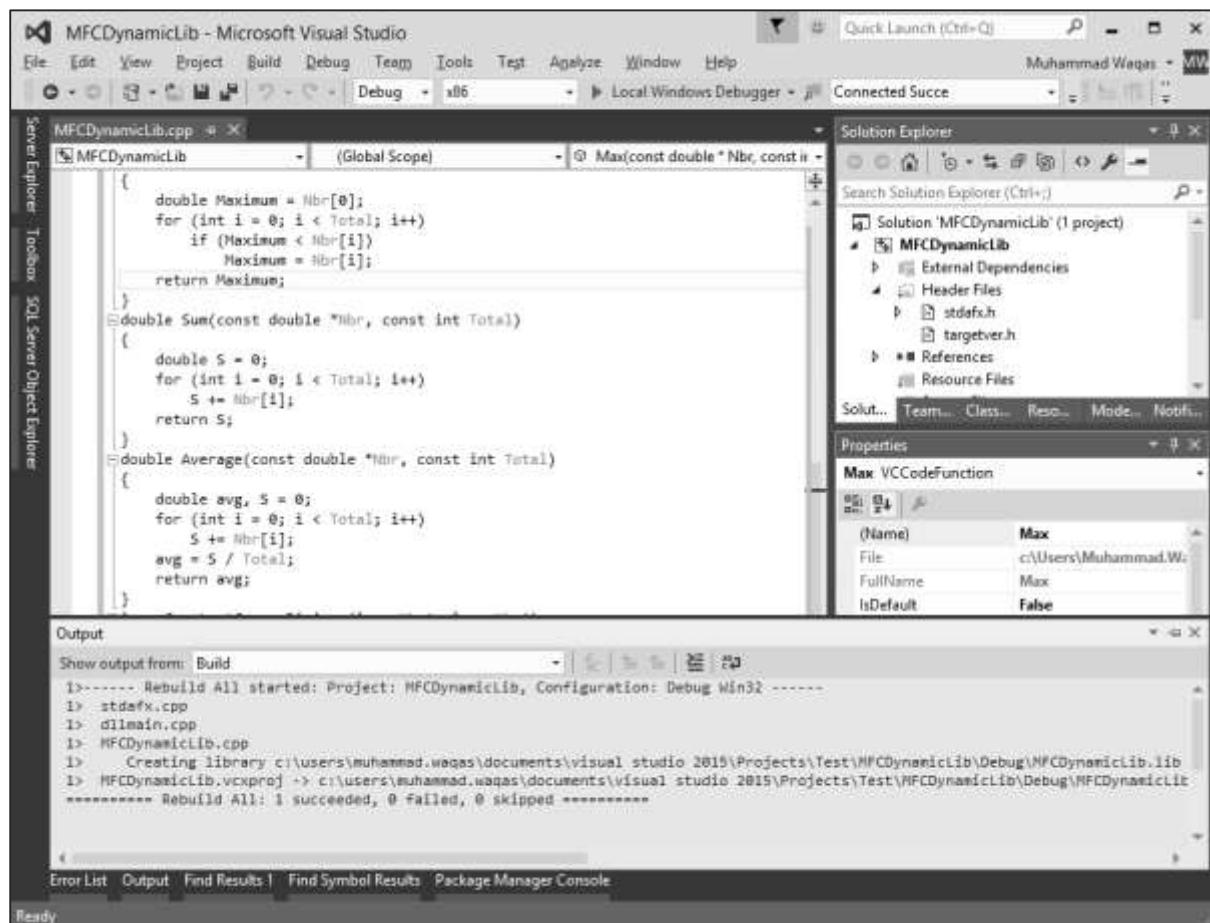
```

```

long GreatestCommonDivisor(long Nbr1, long Nbr2)
{
    while (true)
    {
        Nbr1 = Nbr1 % Nbr2;
        if (Nbr1 == 0)
            return Nbr2;
        Nbr2 = Nbr2 % Nbr1;
        if (Nbr2 == 0)
            return Nbr1;
    }
}

```

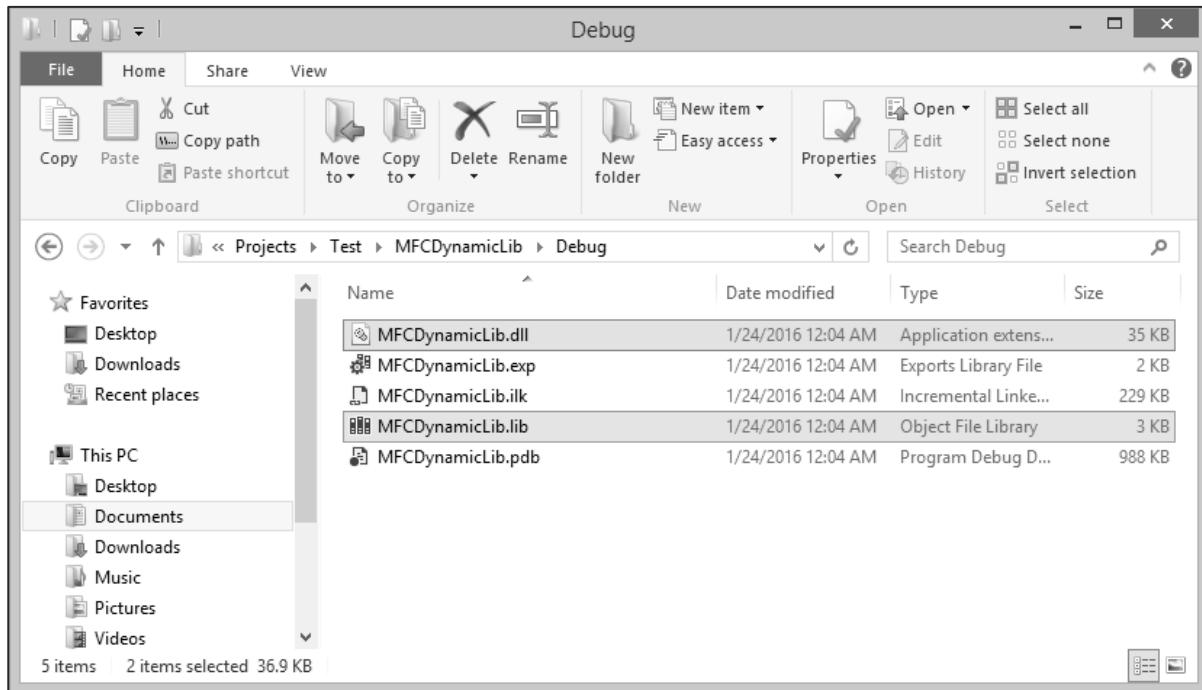
**Step 6:** To create the DLL, on the main menu, click **Build > Build MFCDynamicLib** from the main menu.



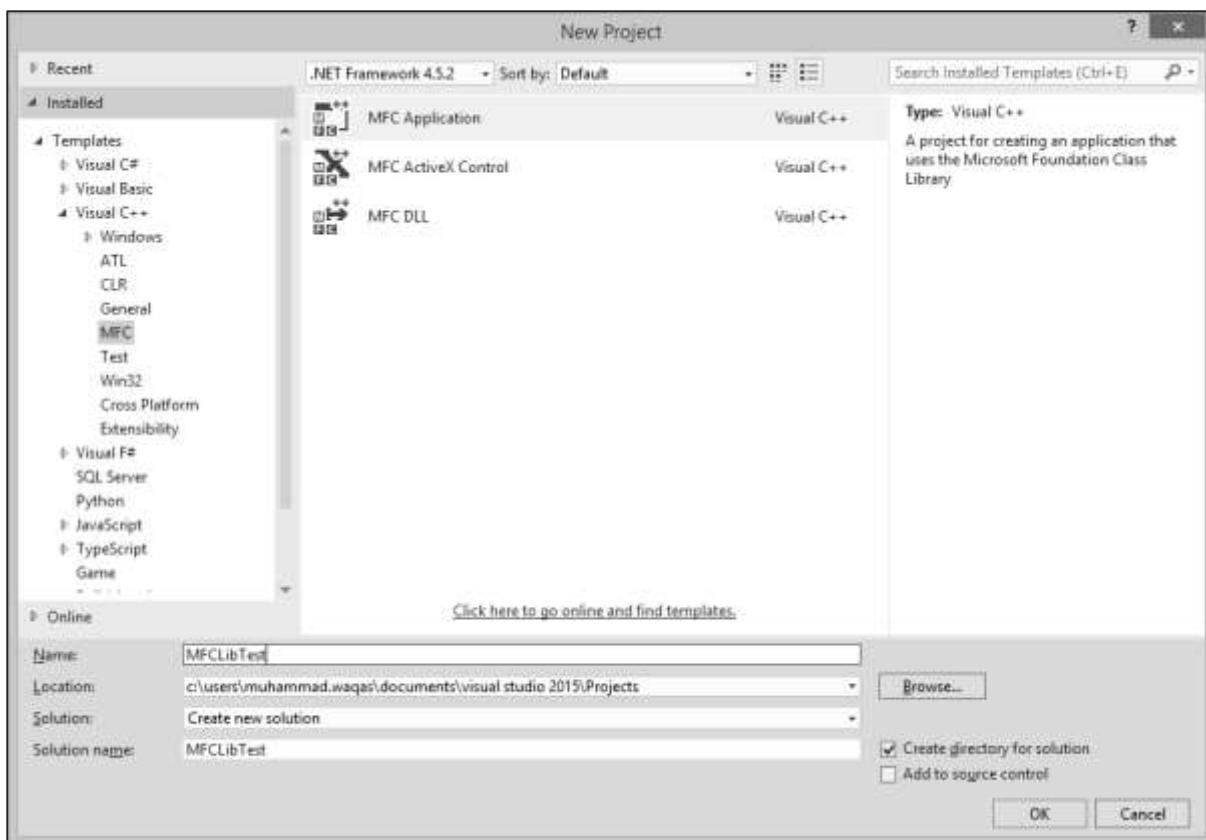
**Step 7:** Once the DLL is successfully created, you will see a message display in output window.

**Step 8:** Open Windows Explorer and then the Debug folder of the current project.

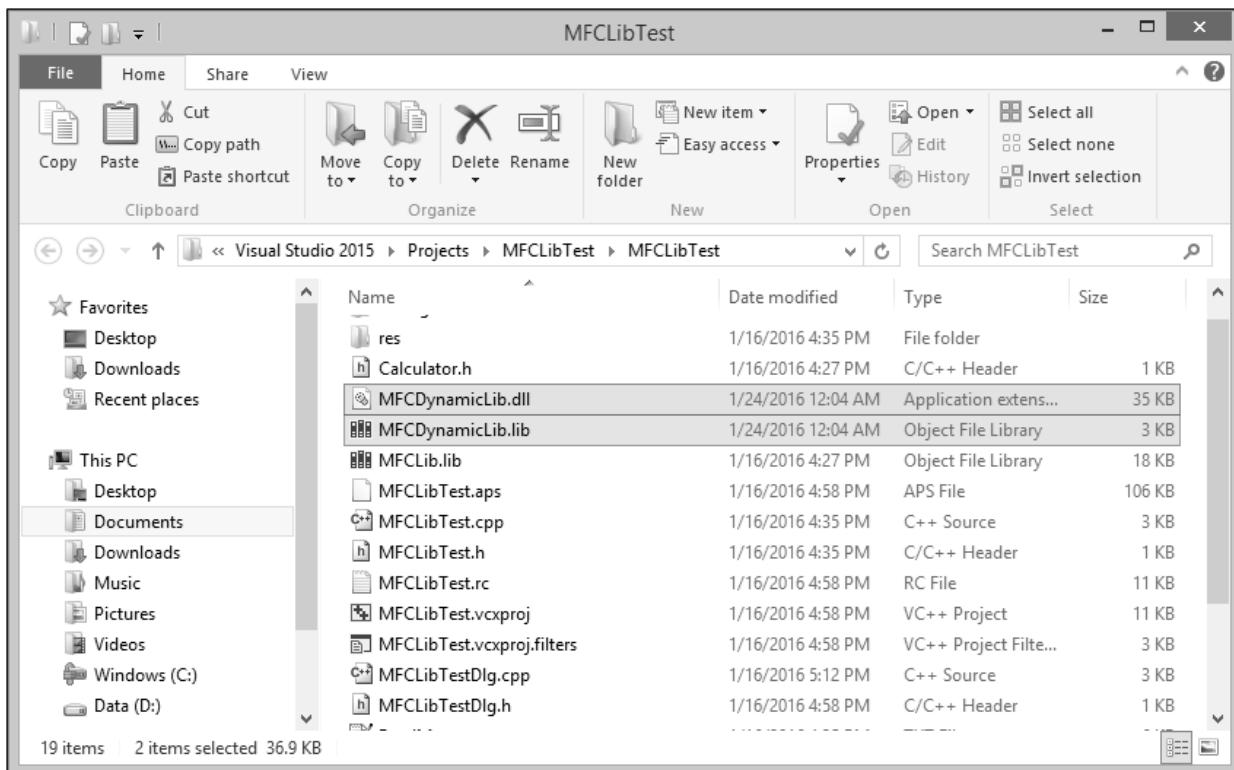
**Step 9:** Notice that a file with dll extension and another file with lib extension has been created.



**Step 10:** To test this file with dll extension, we need to create a new MFC dialog based application from File -> New --> Project.

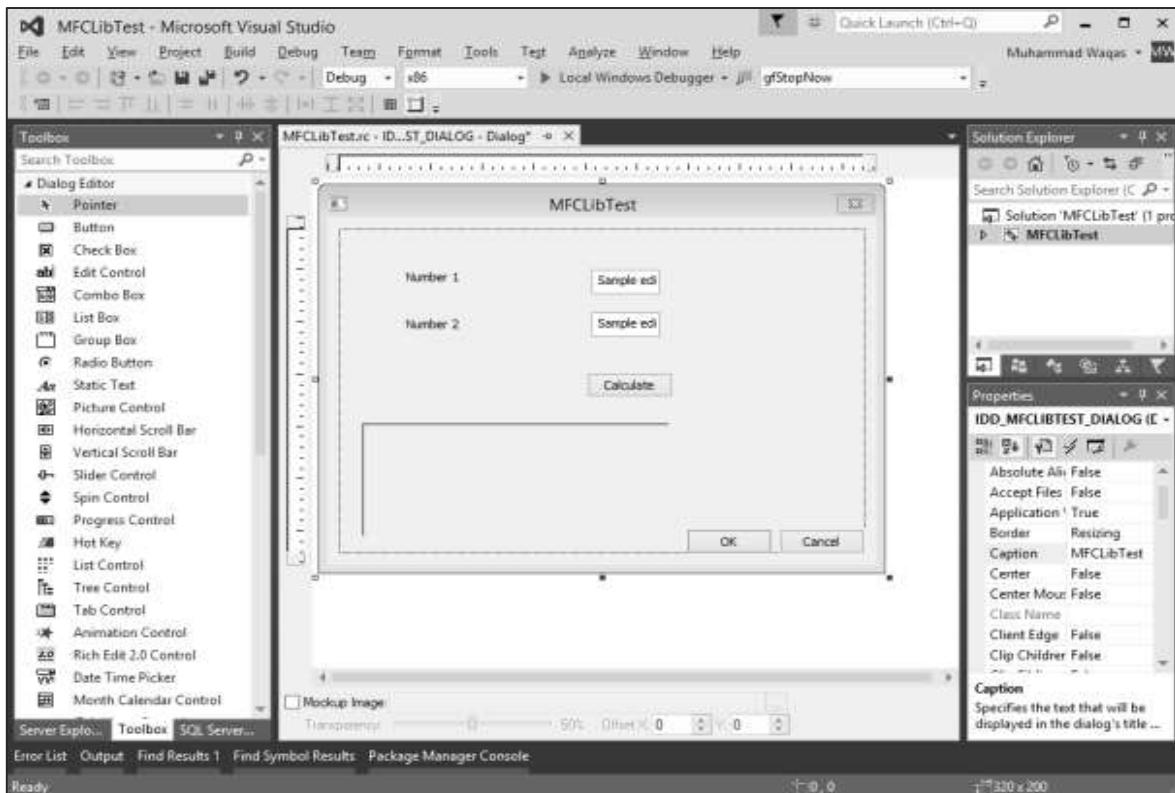


**Step 11:** Go to the MFCDynamicLib\Debug folder and copy the \*.dll and \*.lib files to the MFCLibTest project as shown in the following snapshot.

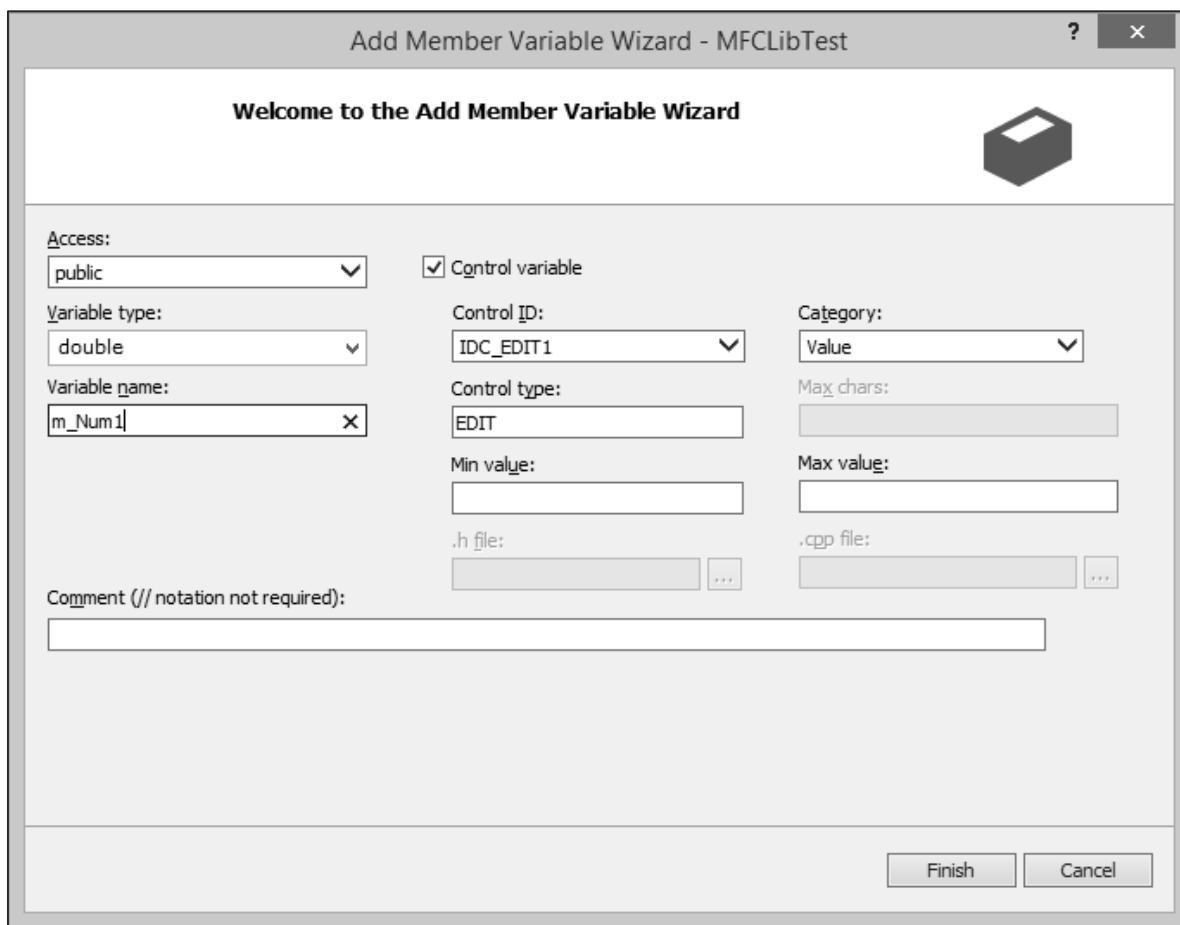


**Step 12:** To add the DLL to the current project, on the main menu, click Project -> Add Existing Item and then, select MFCDynamicLib.lib file.

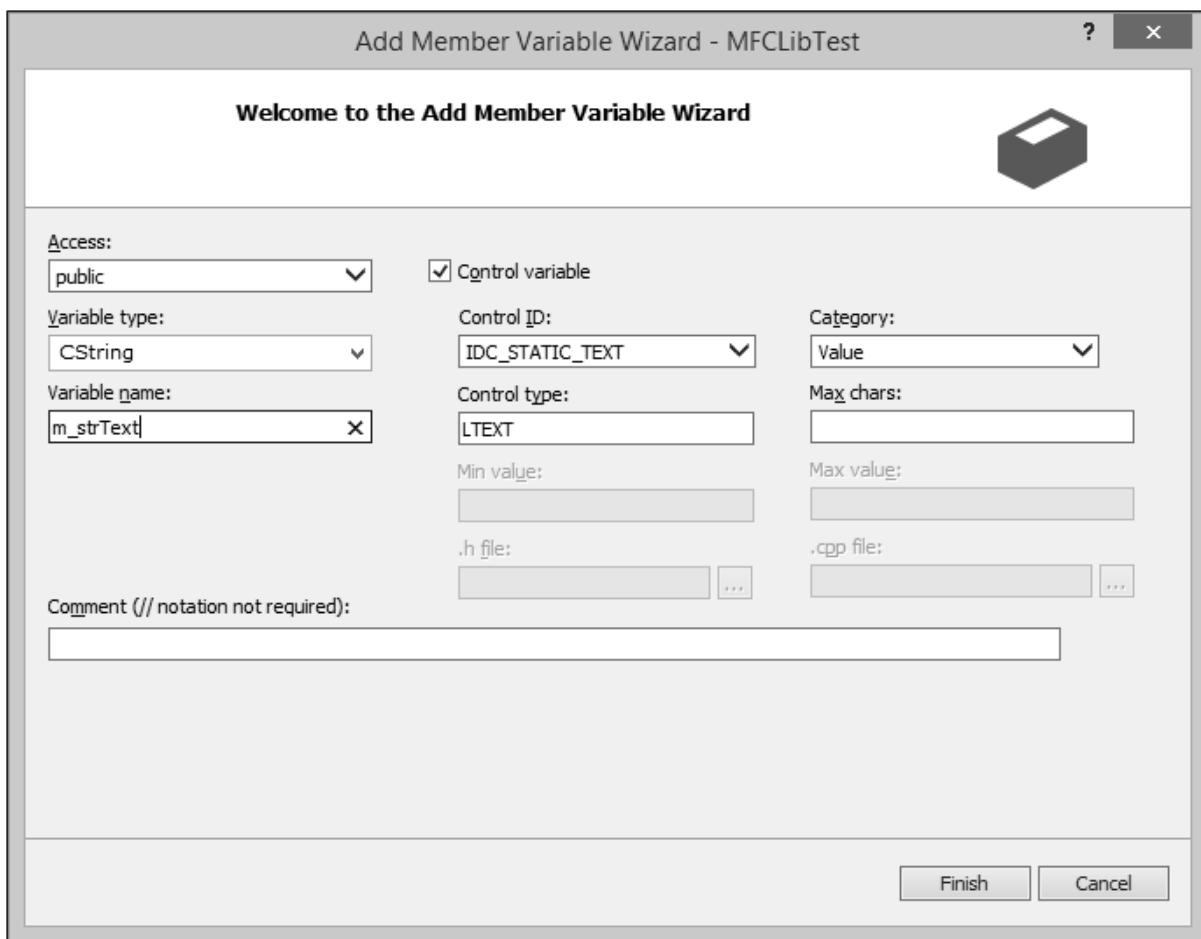
**Step 13:** Design your dialog box as shown in the following snapshot.



**Step 14:** Add value variable for both edit controls of value type double.



**Step 15:** Add value variable for Static text control, which is at the end of the dialog box.



**Step 16:** Add the event handler for Calculate button.

**Step 17:** In the project that is using the DLL, each function that will be accessed must be declared using the `_declspec(dllexport)` modifier.

**Step 18:** Add the following function declaration in MFCLibTestDlg.cpp file.

```
extern "C" __declspec(dllexport) double Min(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) double Max(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) double Sum(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) double Average(const double *Numbers, const int Count);
extern "C" __declspec(dllexport) long GreatestCommonDivisor(long Nbr1, long Nbr2);
```

**Step 19:** Here is the implementation of button event handler.

```
void CMFCLibTestDlg::OnBnClickedButtonCal()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    CString strTemp;
    double numbers[2];
    numbers[0] = m_Num1;
    numbers[1] = m_Num2;
    strTemp.Format(L"%.2f", Max(numbers, 2));
    m_strText.Append(L"Max is:\t" + strTemp);

    strTemp.Format(L"%.2f", Min(numbers, 2));
    m_strText.Append(L"\nMin is:\t" + strTemp);

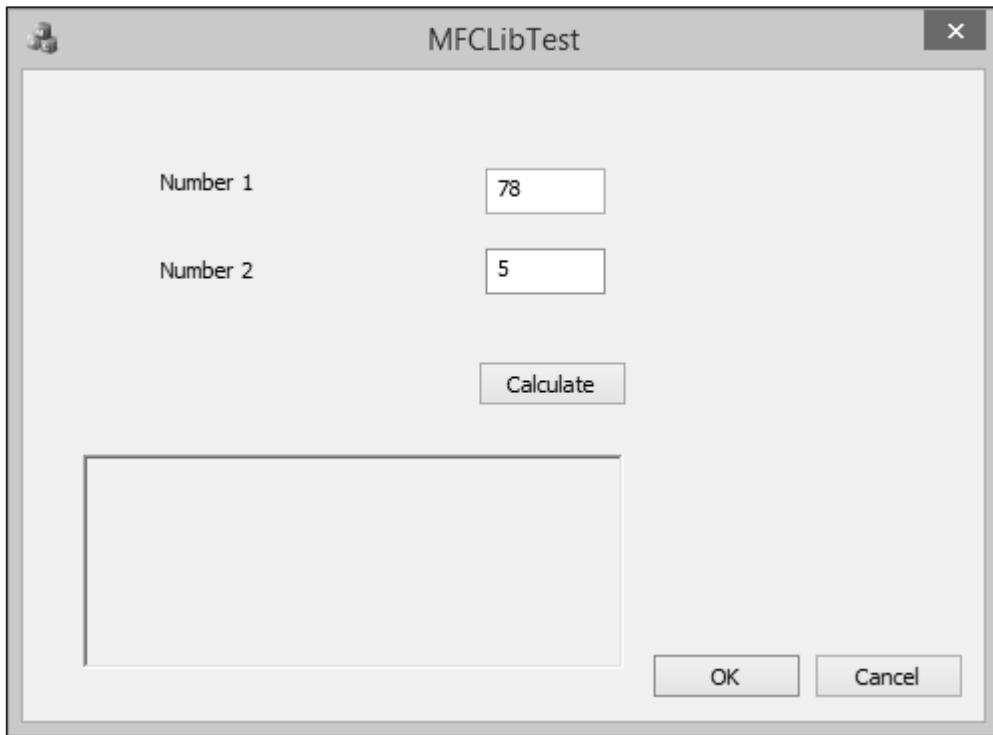
    strTemp.Format(L"%.2f", Sum(numbers, 2));
    m_strText.Append(L"\nSum is:\t" + strTemp);

    strTemp.Format(L"%.2f", Average(numbers, 2));
    m_strText.Append(L"\nAverage is:\t" + strTemp);

    strTemp.Format(L"%d", GreatestCommonDivisor(m_Num1, m_Num2));
    m_strText.Append(L"\nGDC is:\t" + strTemp);

    UpdateData(FALSE);
}
```

**Step 20:** When the above code is compiled and executed, you will see the following output.



**Step 21:** Enter two values in the edit field and click Calculate. You will now see the result after calculating from the DLL.

