# MFC

tutorialspoint

SIMPLYEASYLEARNING

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 2018 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.



tutorialspoint
SIMPLYEASYLEARNING

# Table of Contents





tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

# 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 you 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.



tutorialspoint
SIMPLYEASYLEARNING

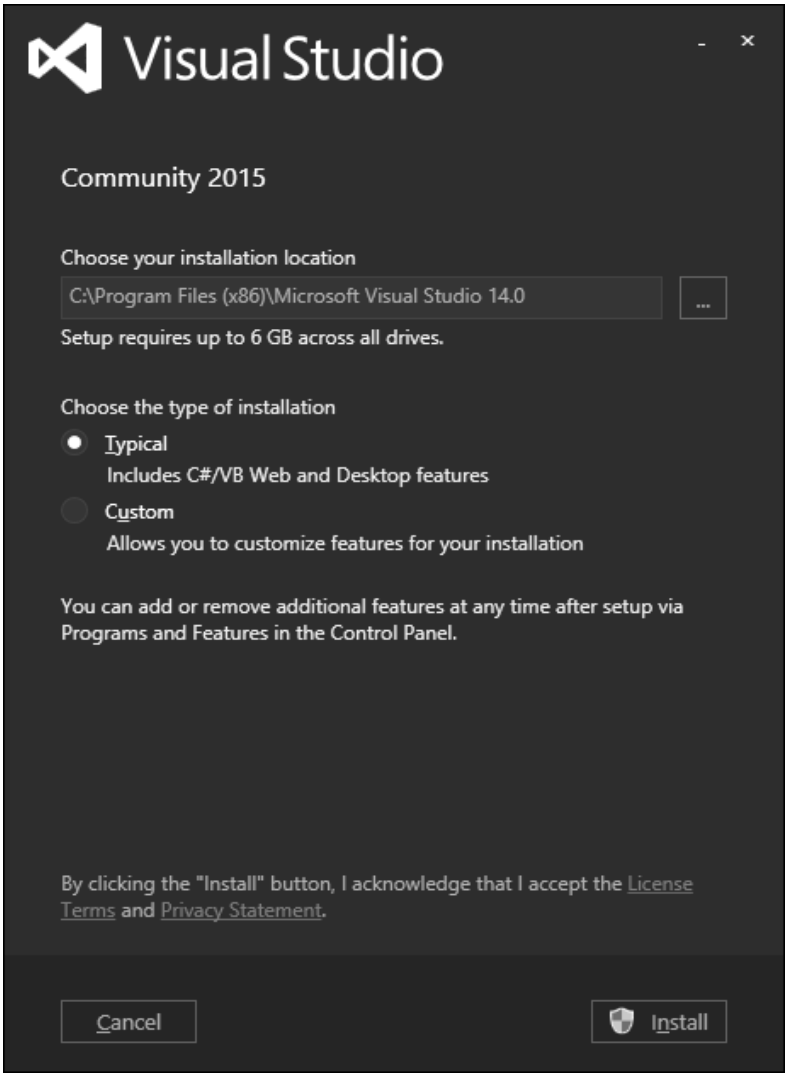
# 2. MFC — Environment Setup

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.

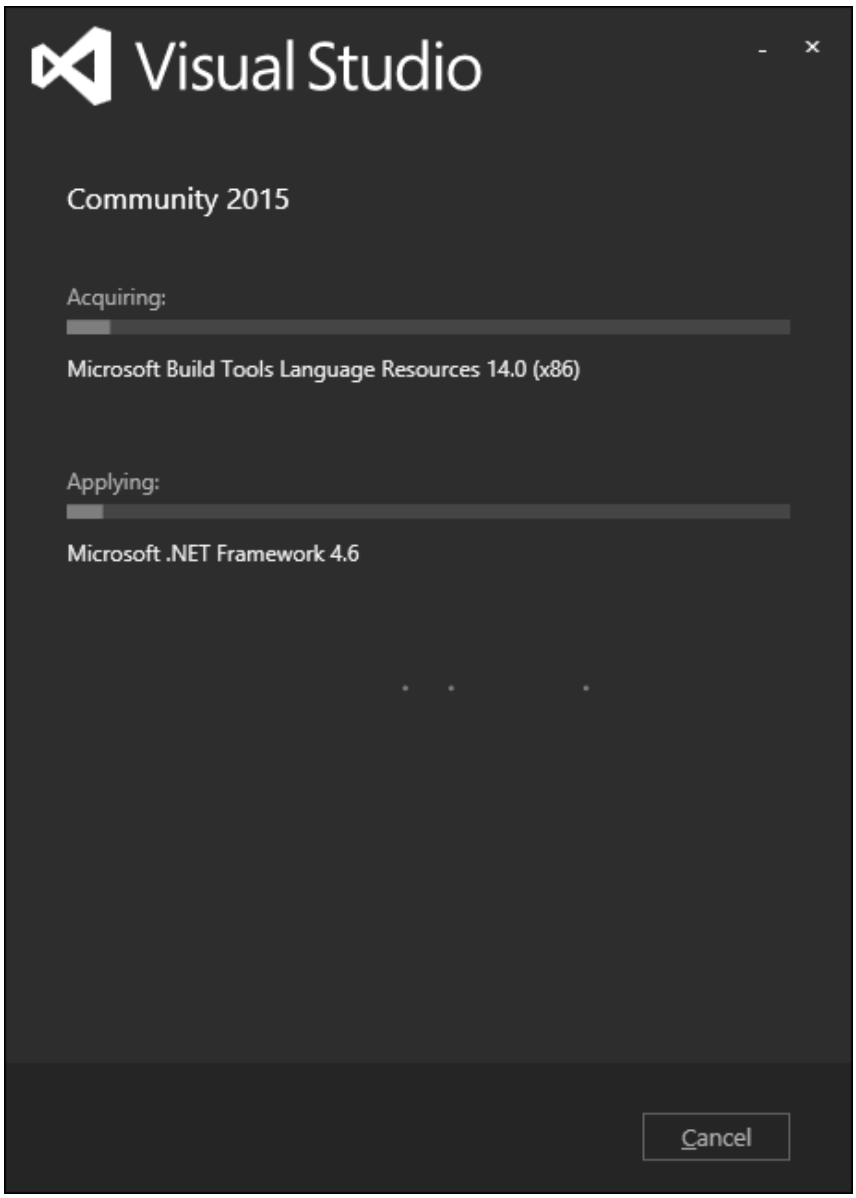
**Step 1:** Once Visual Studio is downloaded, run the installer. The following dialog box will be displayed.




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



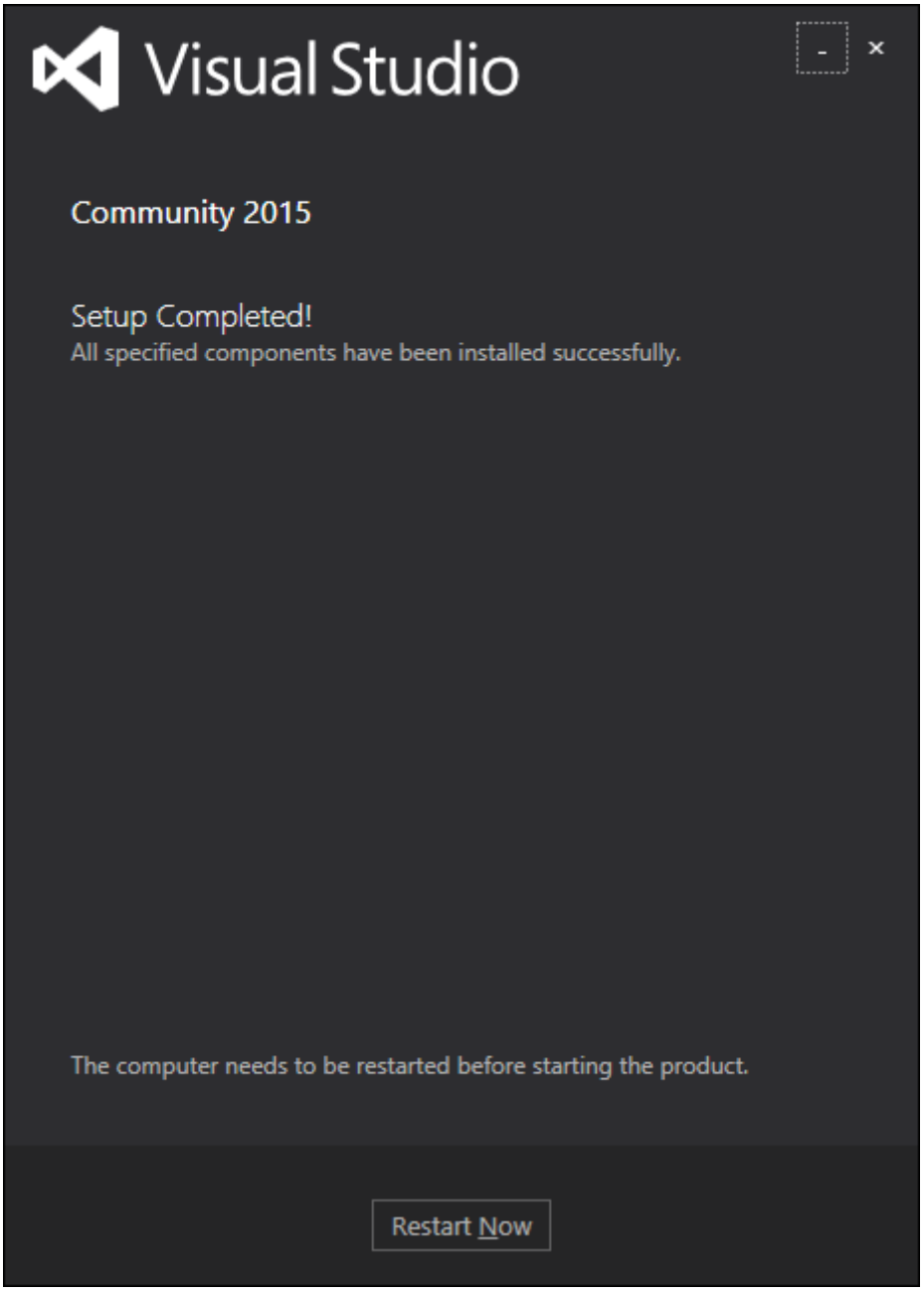
tutorialspoint
SIMPLYEASYLEARNING






tutorialspoint
SIMPLYEASYLEARNING

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



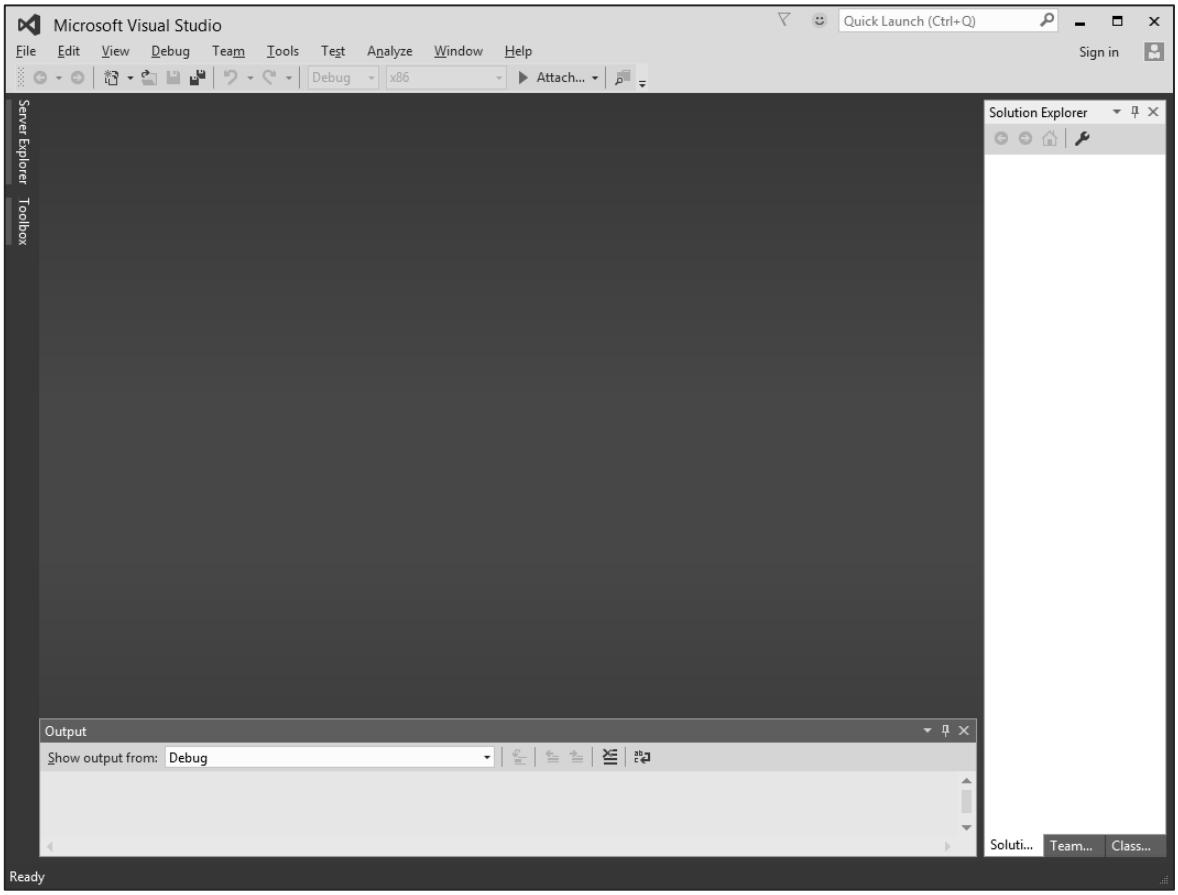

**Step 4:** Close this dialog box and restart your computer if required.

**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.




tutorialspoint
SIMPLYEASYLEARNING

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




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



tutorialspoint
SIMPLYEASYLEARNING

# 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 |
|---|---|
| **MFC Application** | 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. |
| **MFC ActiveX Control** | 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. |
| **MFC DLL** | 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 |
|---|---|
| **Empty Project** | 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. |
| **Custom Wizard** | 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

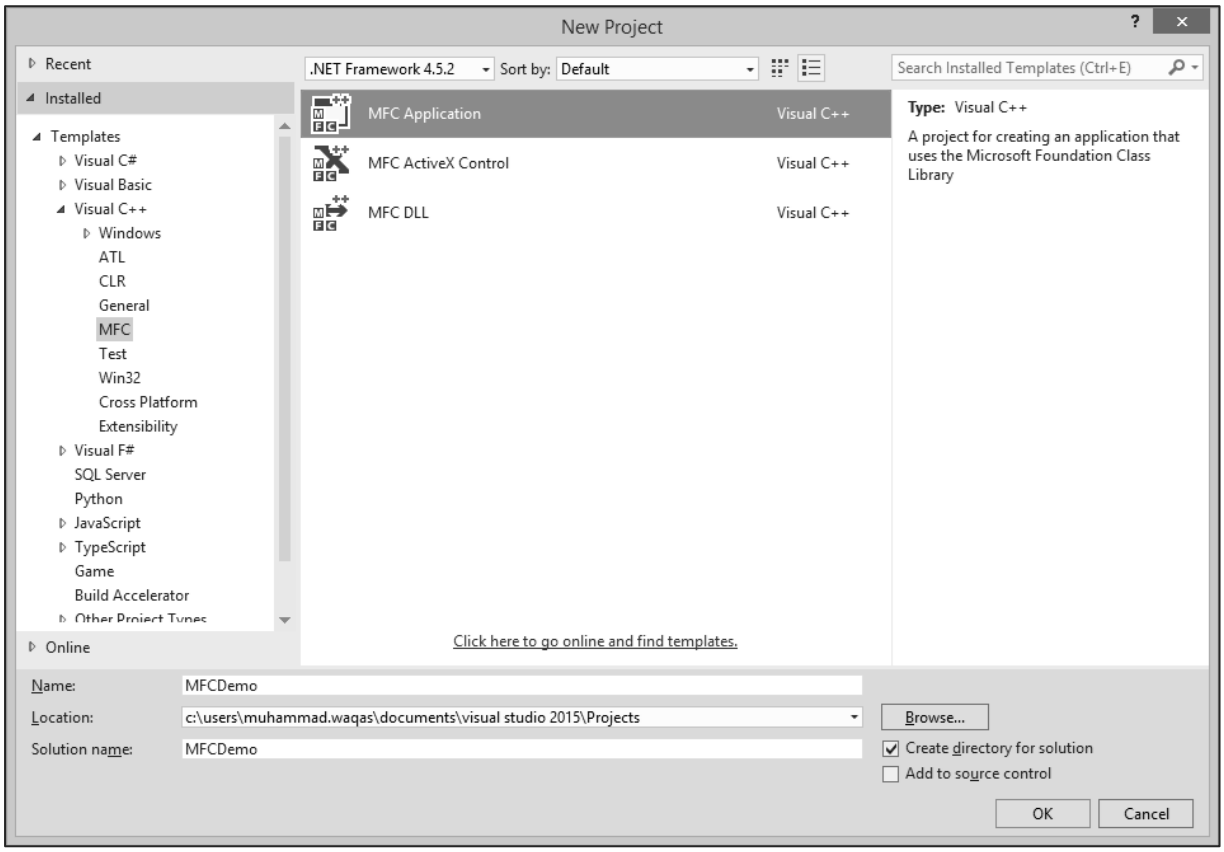
In this chapter, we will look at a working MFC example. To create an MFC application, you can use wizards to customize your projects. You can also create an application from scratch.

## 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

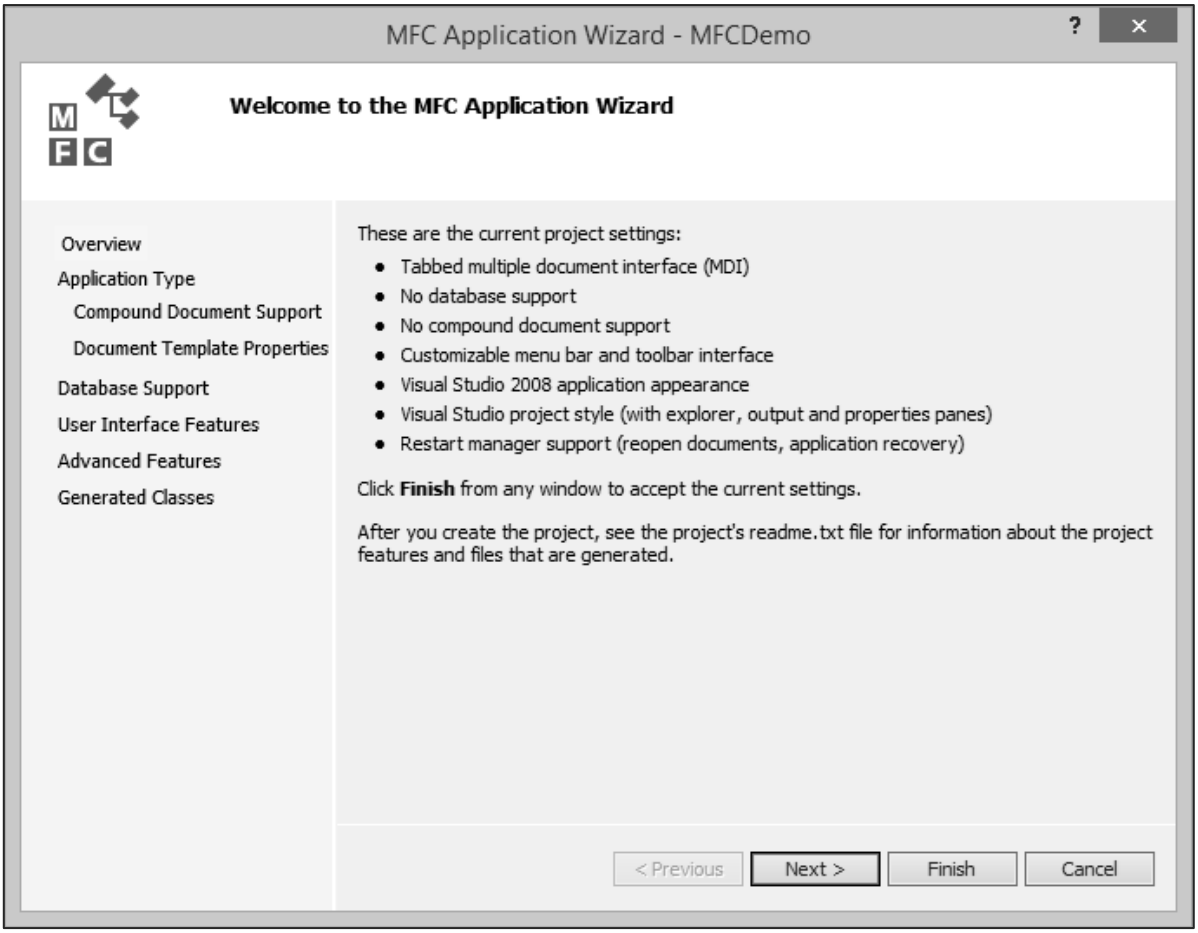
**Step 4:** In the middle pane, select MFC Application.

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

You will see the following dialog.



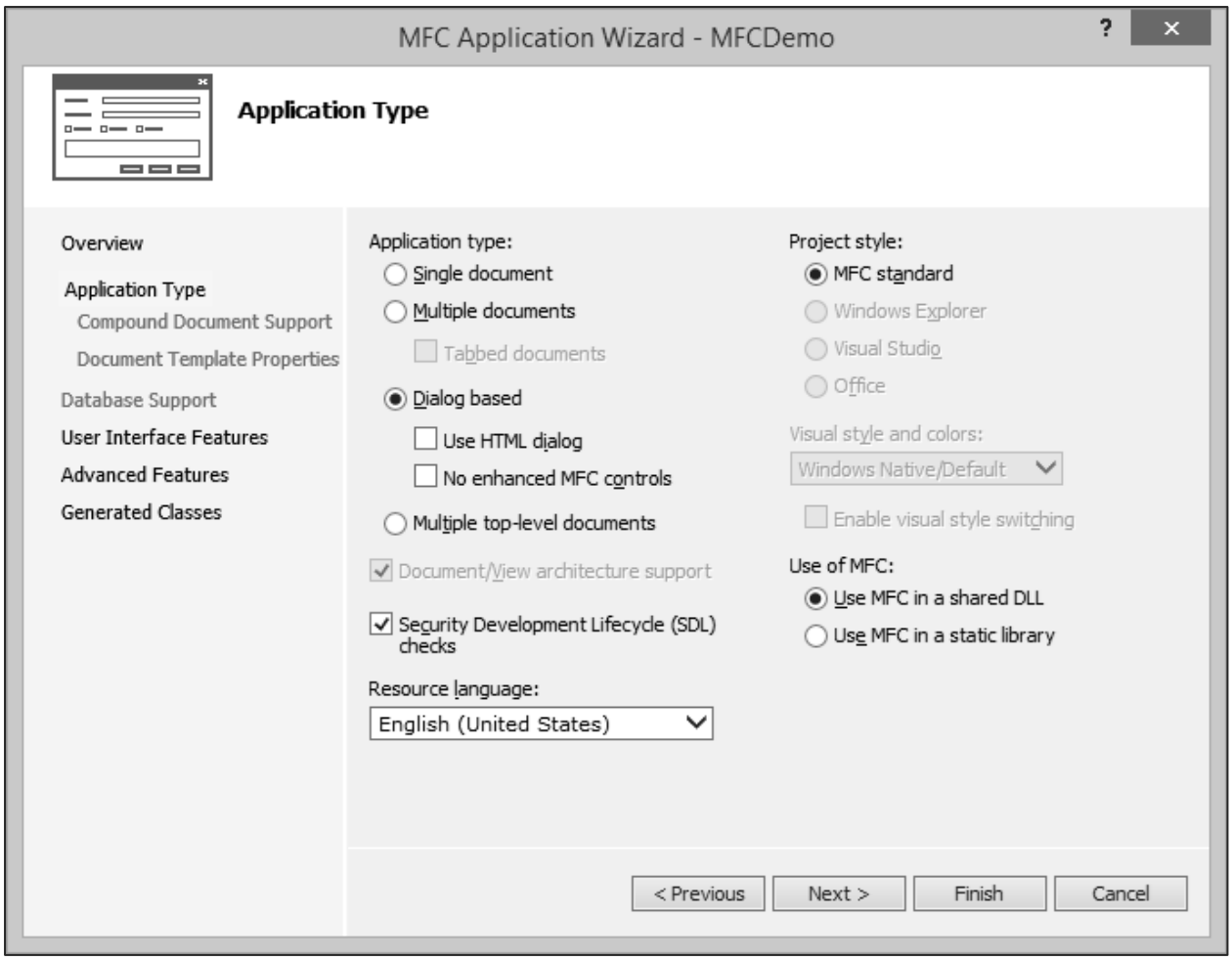
tutorialspoint
SIMPLYEASYLEARNING




**Step 6:** Click Next.



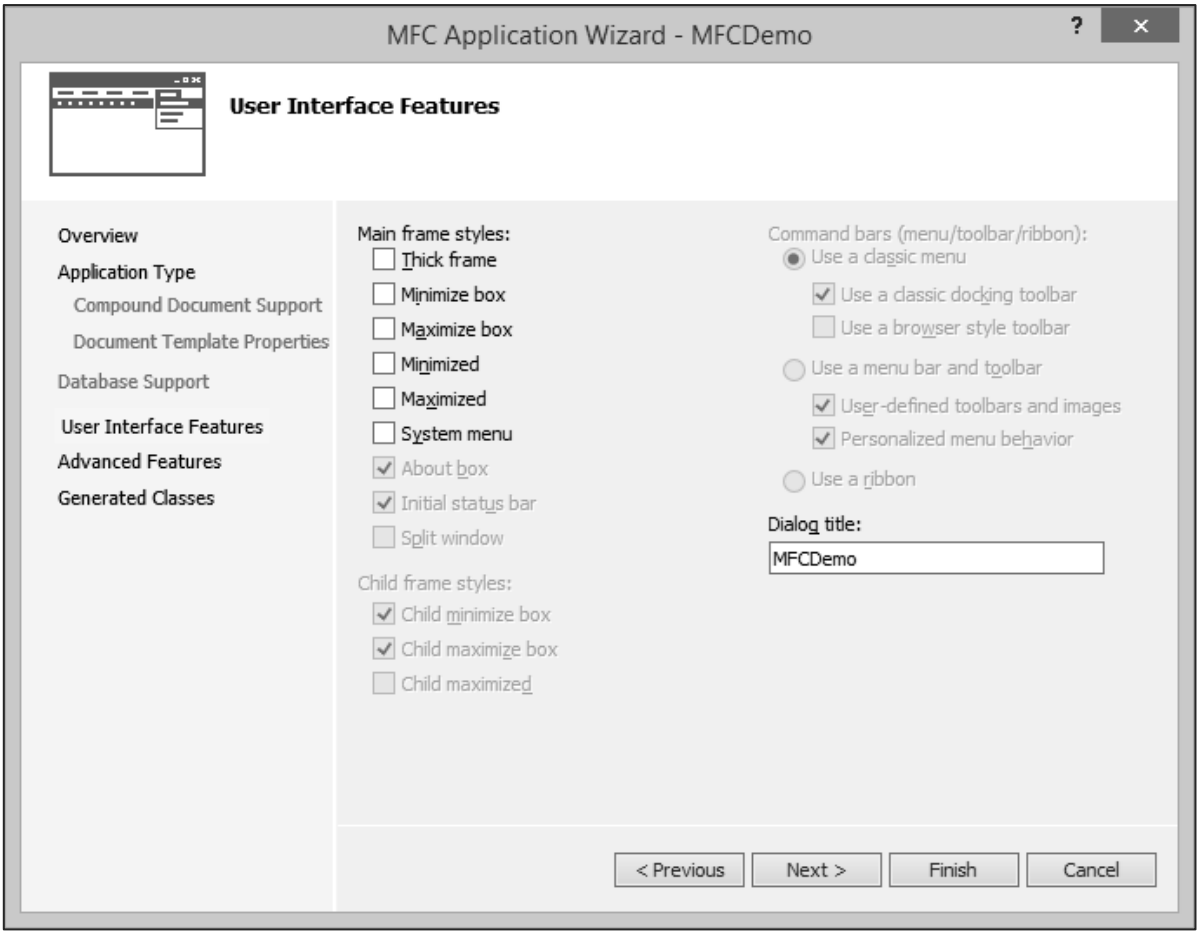
tutorialspoint
SIMPLYEASYLEARNING




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



tutorialspoint
SIMPLYEASYLEARNING

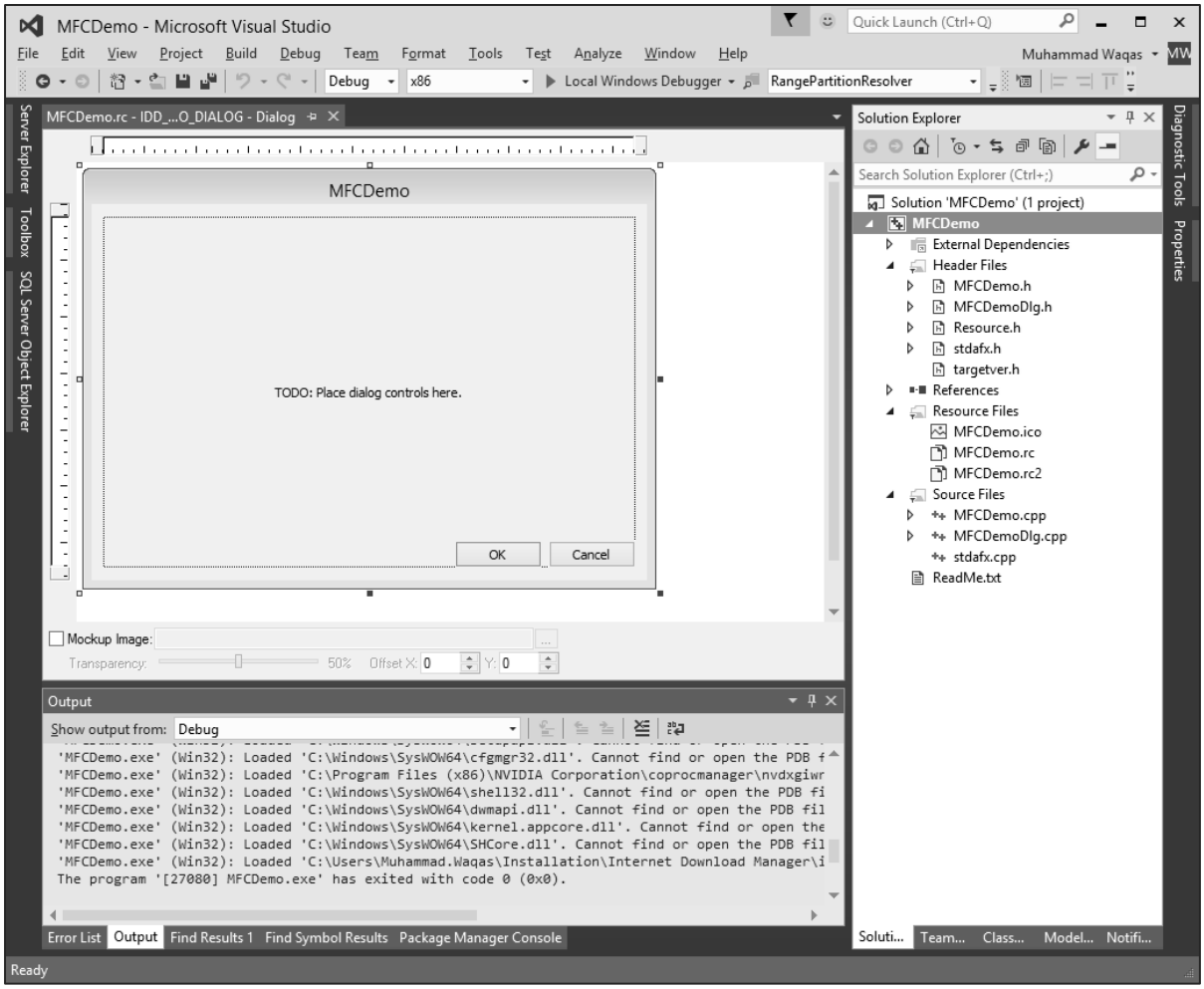



**Step 8:** Uncheck all options and click Finish button.



tutorialspoint
SIMPLYEASYLEARNING

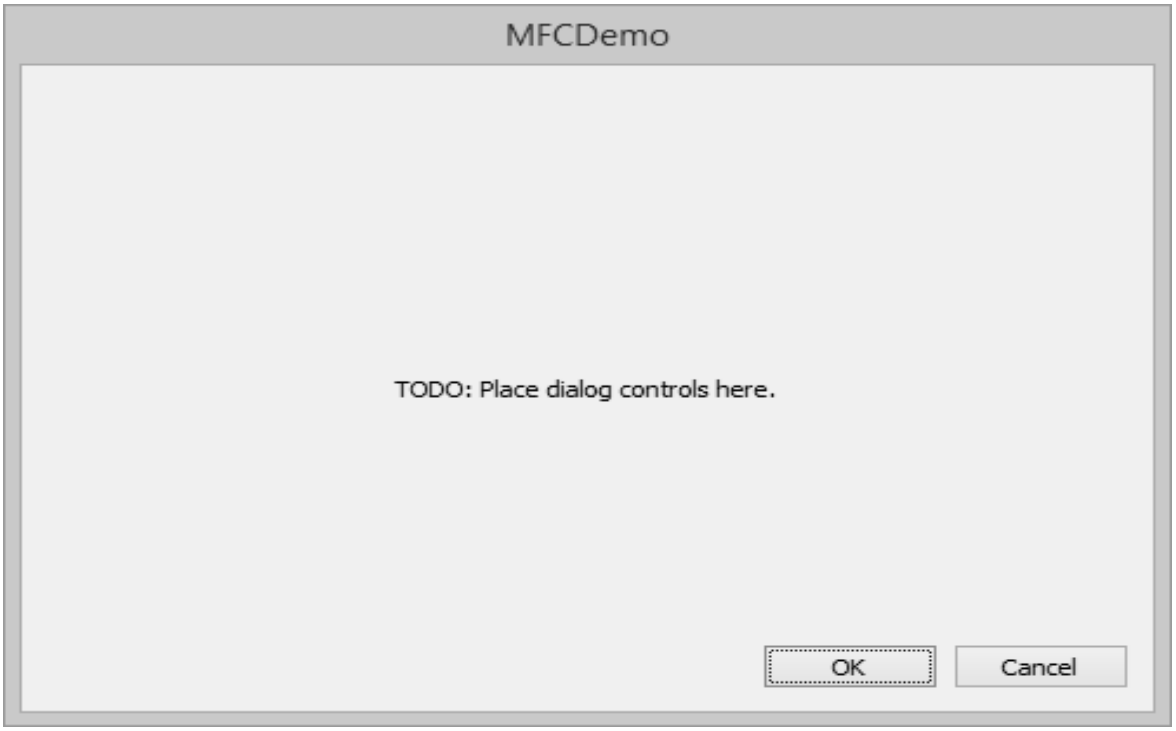
You can now see that the MFC wizard creates this Dialog Box and the project files by default.




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



tutorialspoint
SIMPLYEASYLEARNING




## Create Project from Scratch

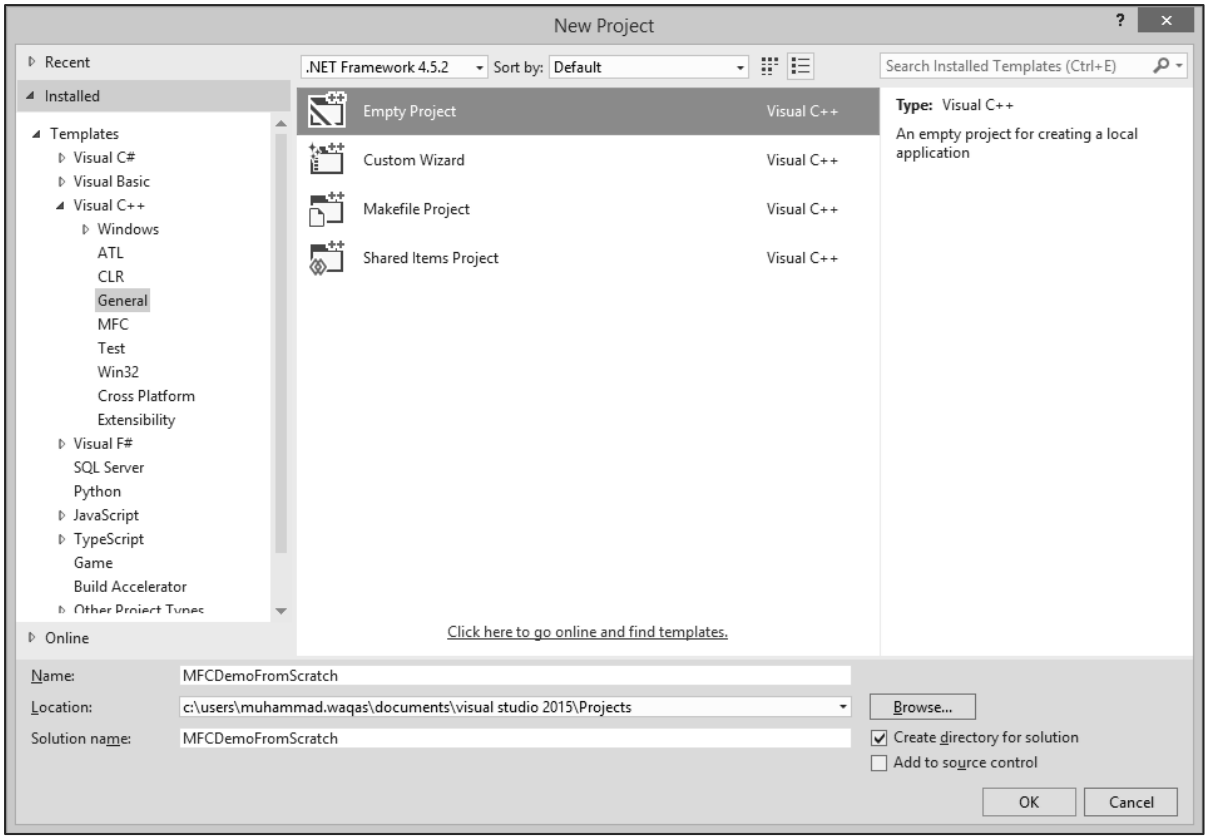
You can also create an MFC application from scratch. To create an MFC application, you need to follow the following Steps.

**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.



tutorialspoint
SIMPLYEASYLEARNING



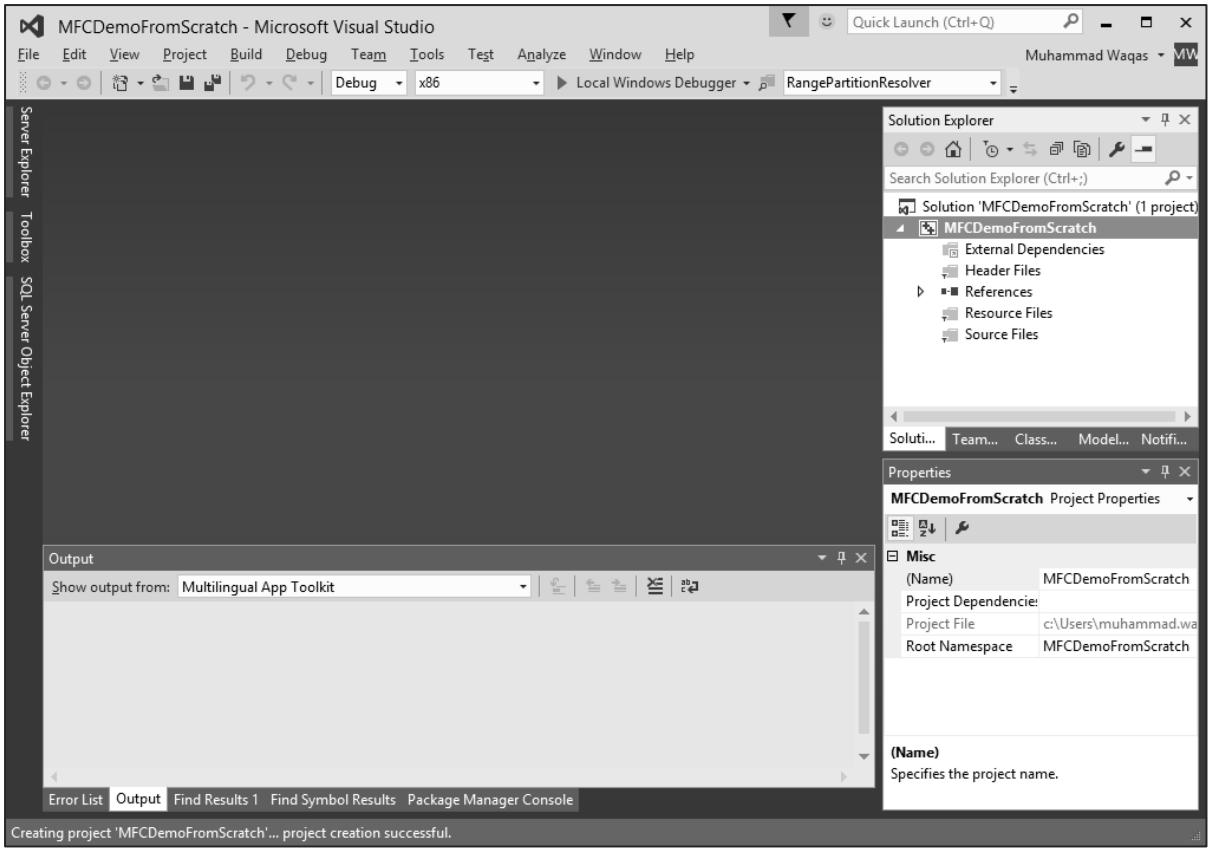

**Step 3**: From the left pane, select Templates -> Visual C++ -> General.

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

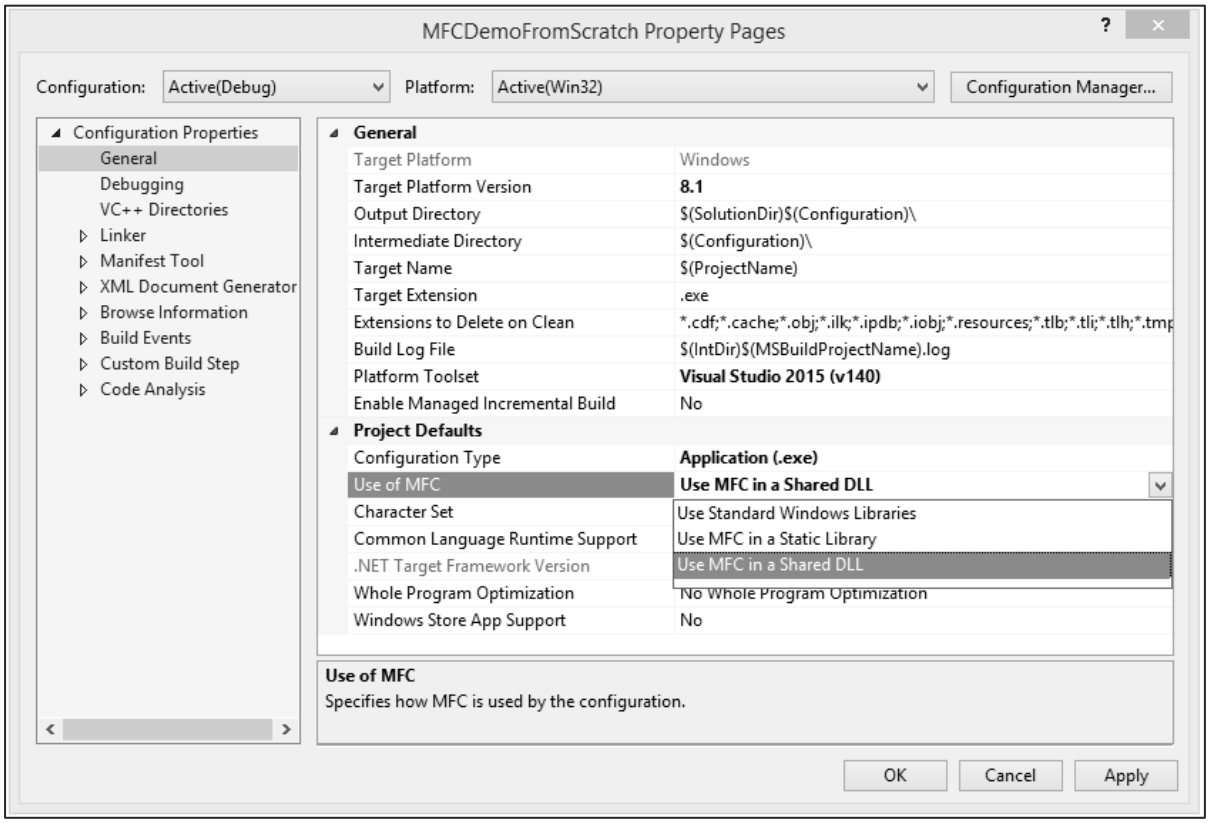
**Step 5**: Enter project name 'MFCDemoFromScratch' in the Name field and click OK to continue. You will see that an empty project is created.



tutorialspoint
SIMPLYEASYLEARNING




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



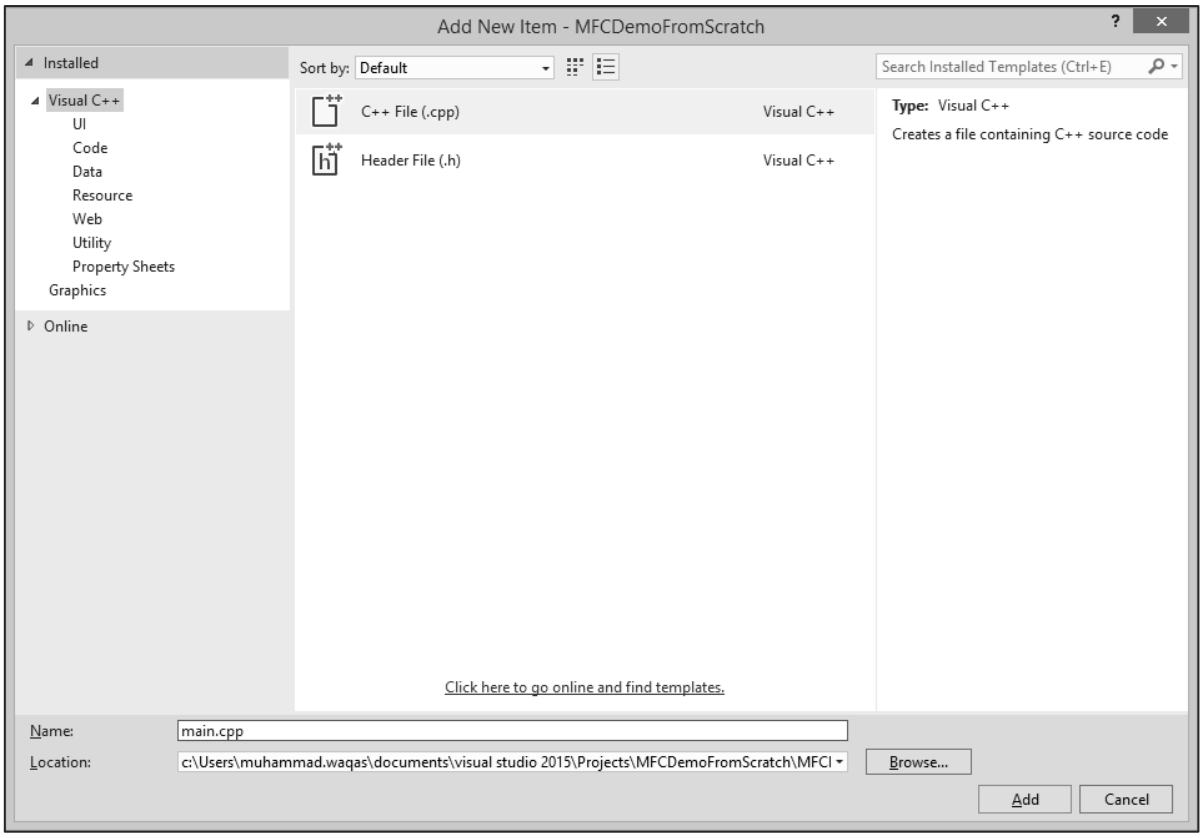



tutorialspoint
SIMPLYEASYLEARNING

**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.

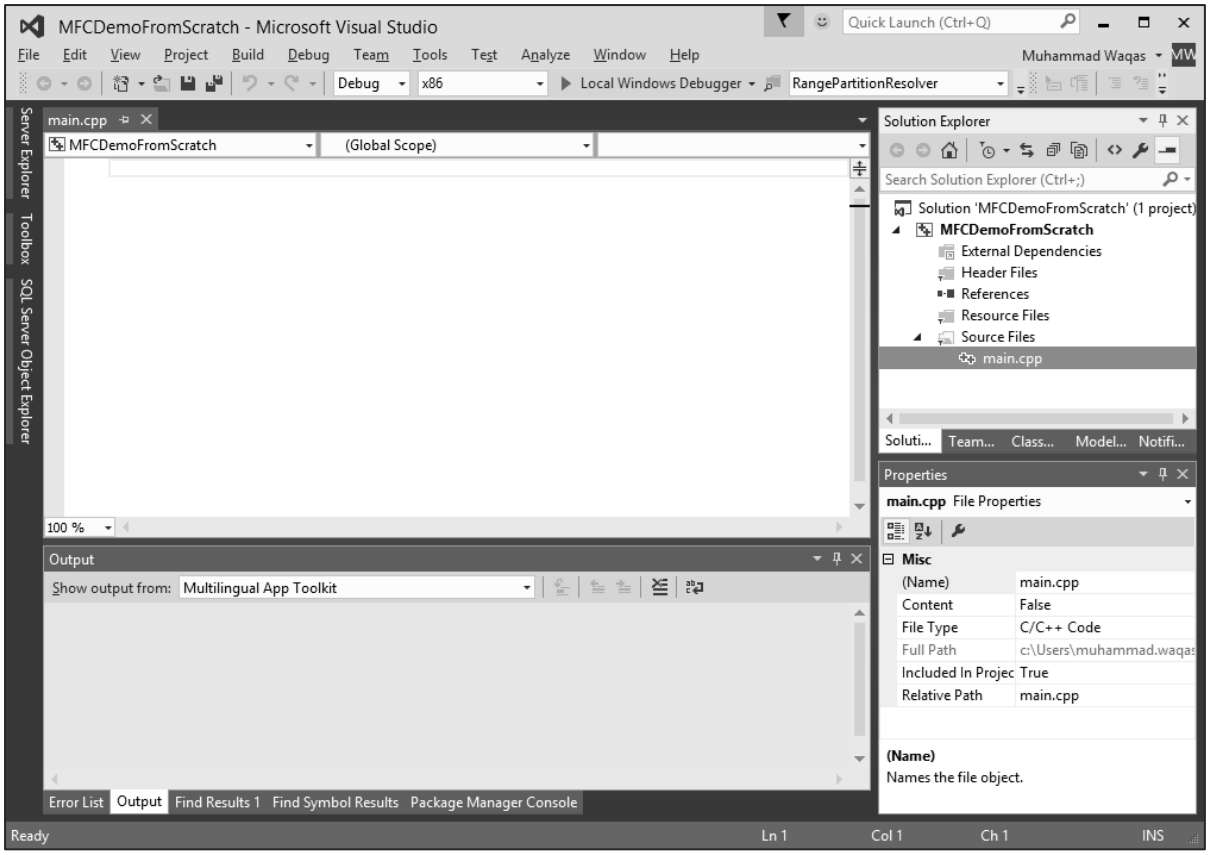
**Step 9**: As it is an empty project now; we need to add a C++ file. So, right-click on the project and select Add -> New Item…




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



tutorialspoint
SIMPLYEASYLEARNING




**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
************************************
```



tutorialspoint
SIMPLYEASYLEARNING

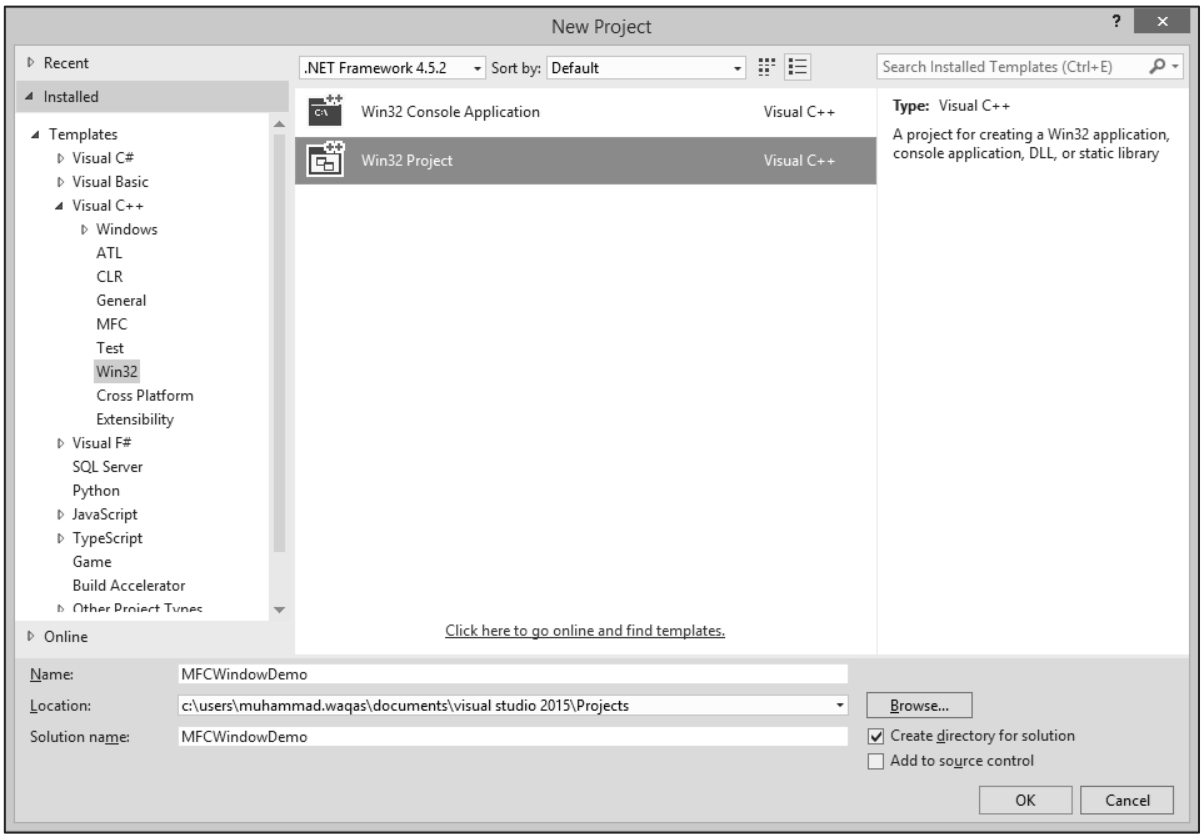
# 5. MFC — Windows Fundamentals

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.



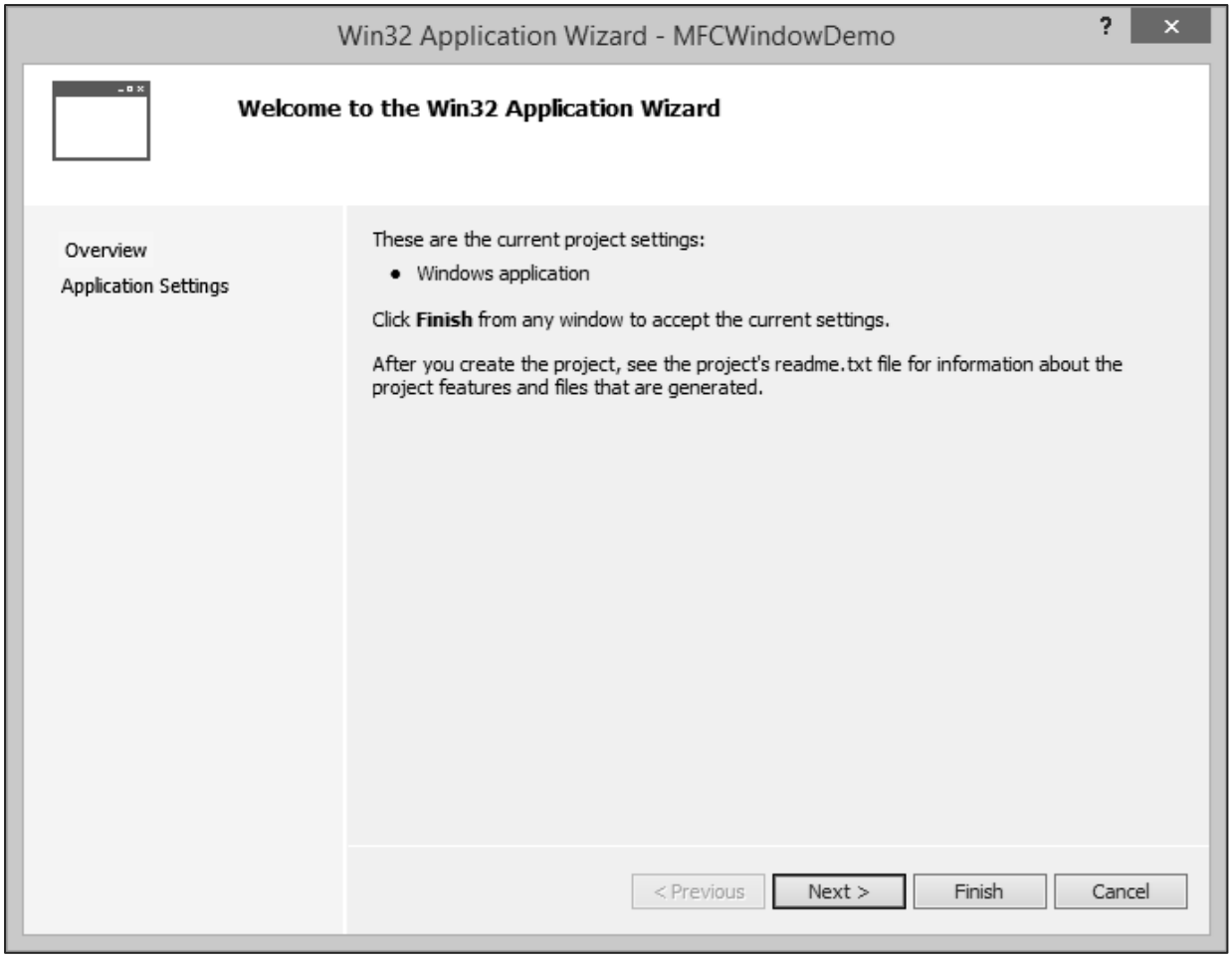

**Step 3**: From the left pane, select Templates -> Visual C++ -> Win32.

**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.



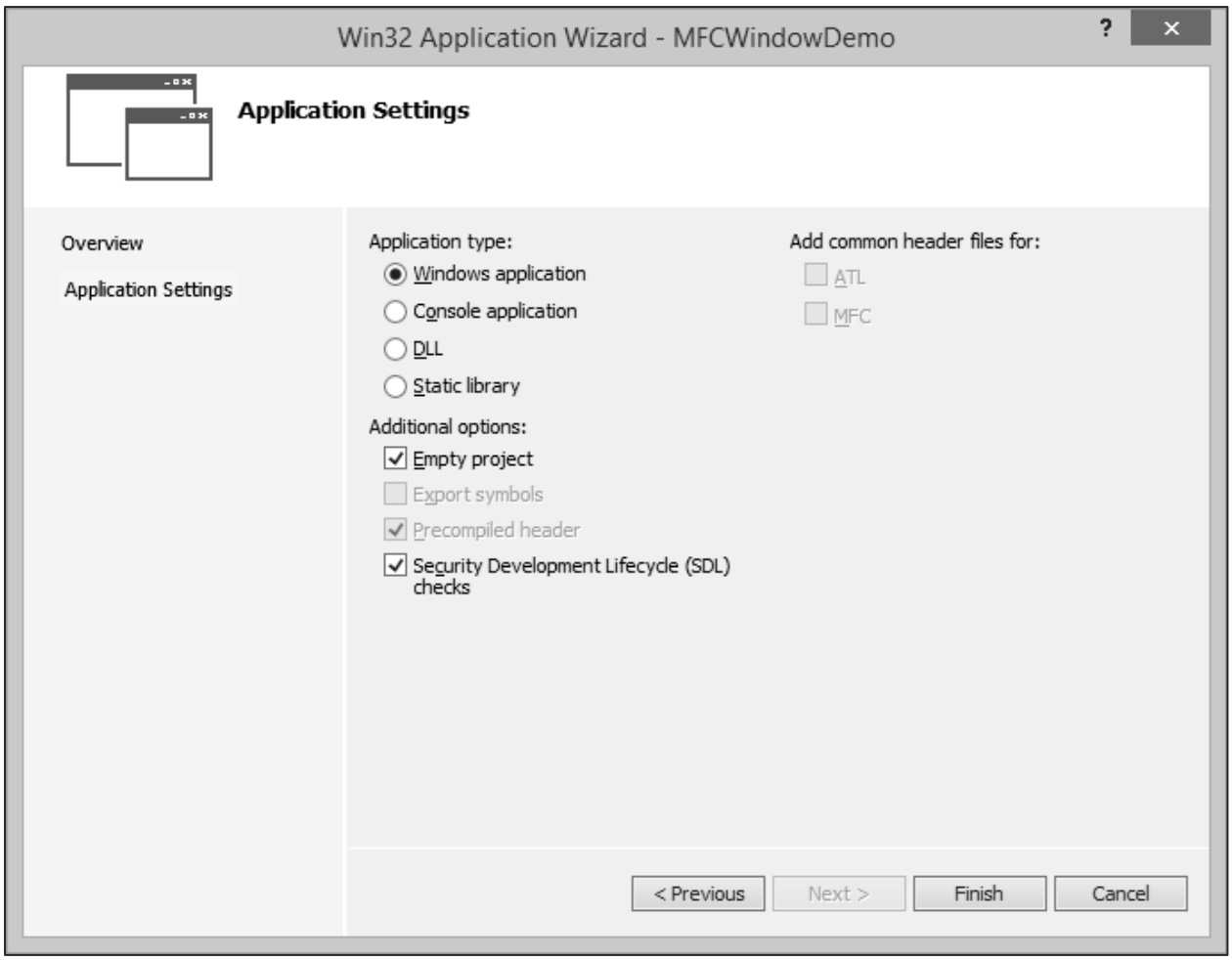
tutorialspoint
SIMPLYEASYLEARNING




**Step 6:** Click Next.



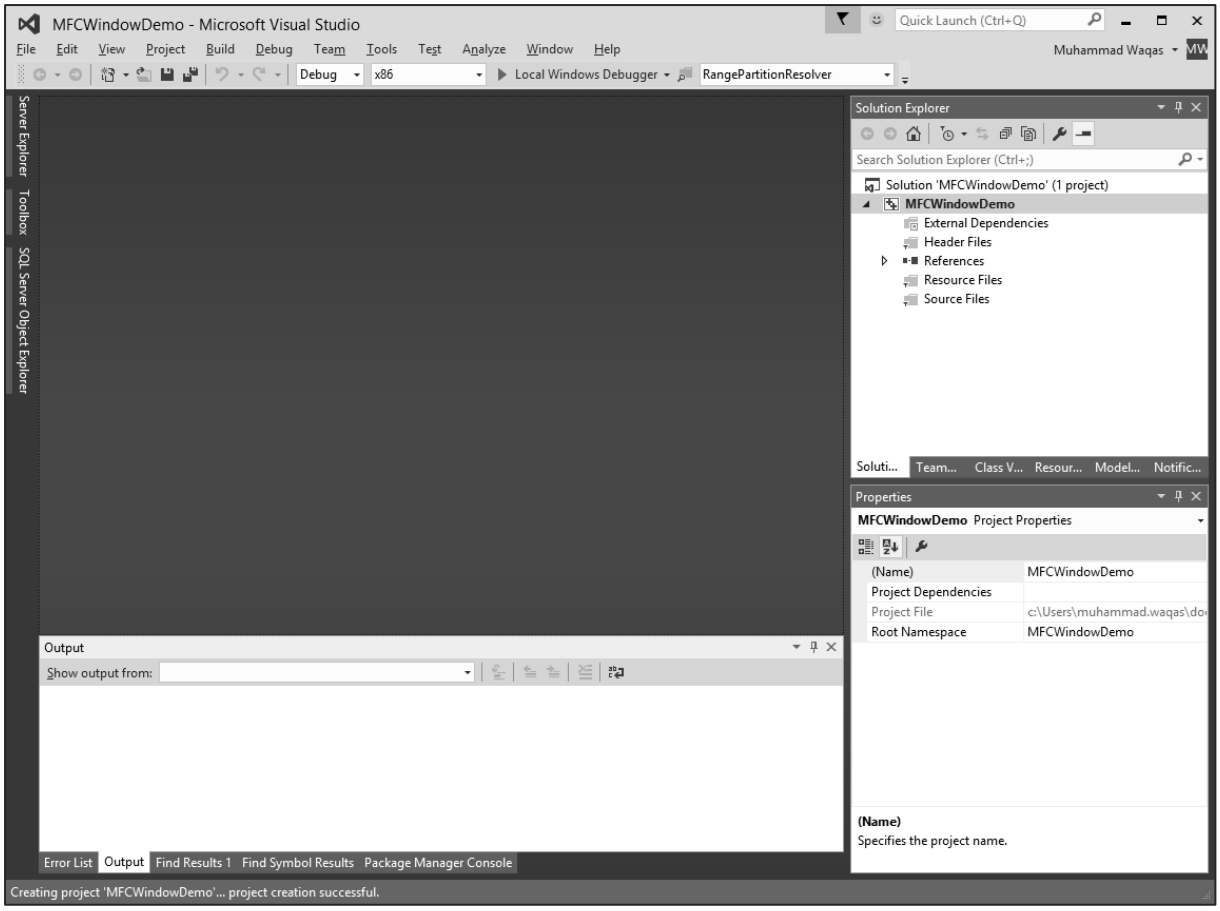
tutorialspoint
SIMPLYEASYLEARNING




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



tutorialspoint
SIMPLYEASYLEARNING

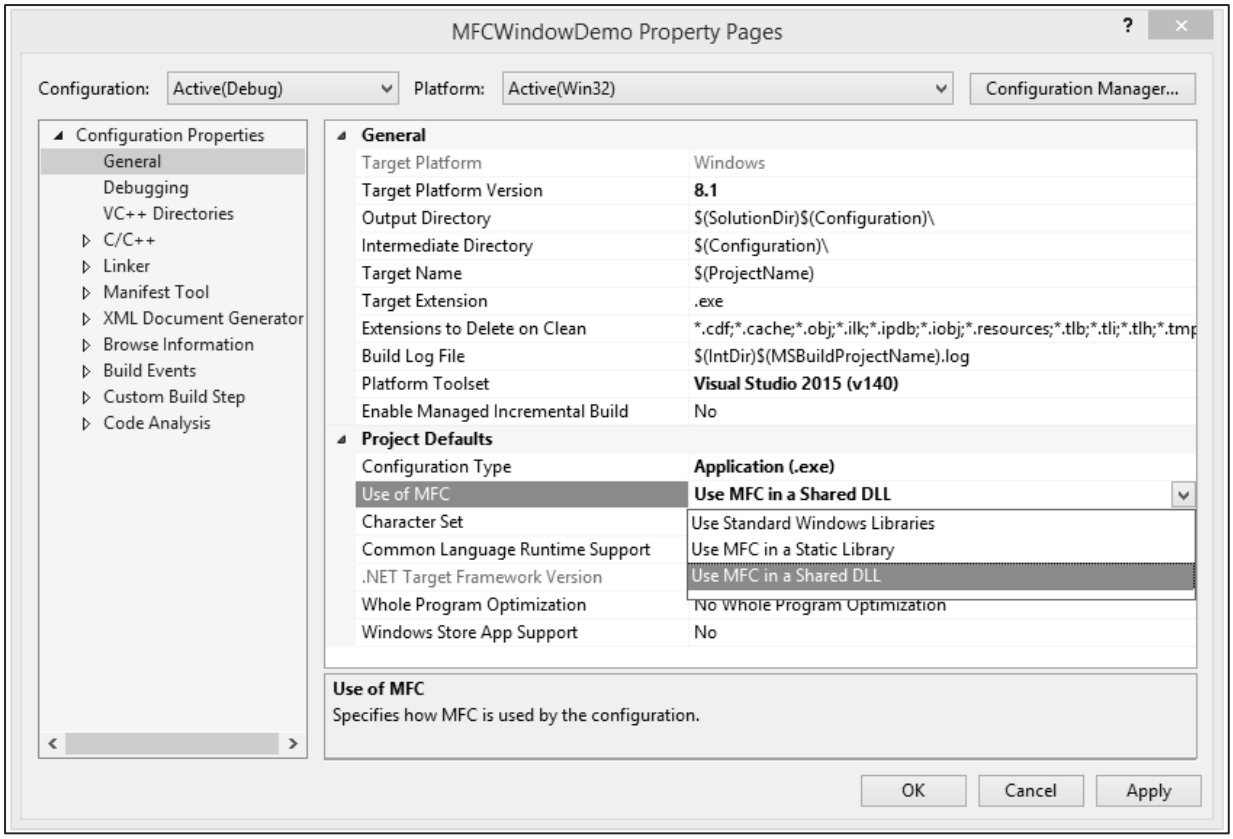



**Step 8:** An empty project is created.

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



tutorialspoint
SIMPLYEASYLEARNING




**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.

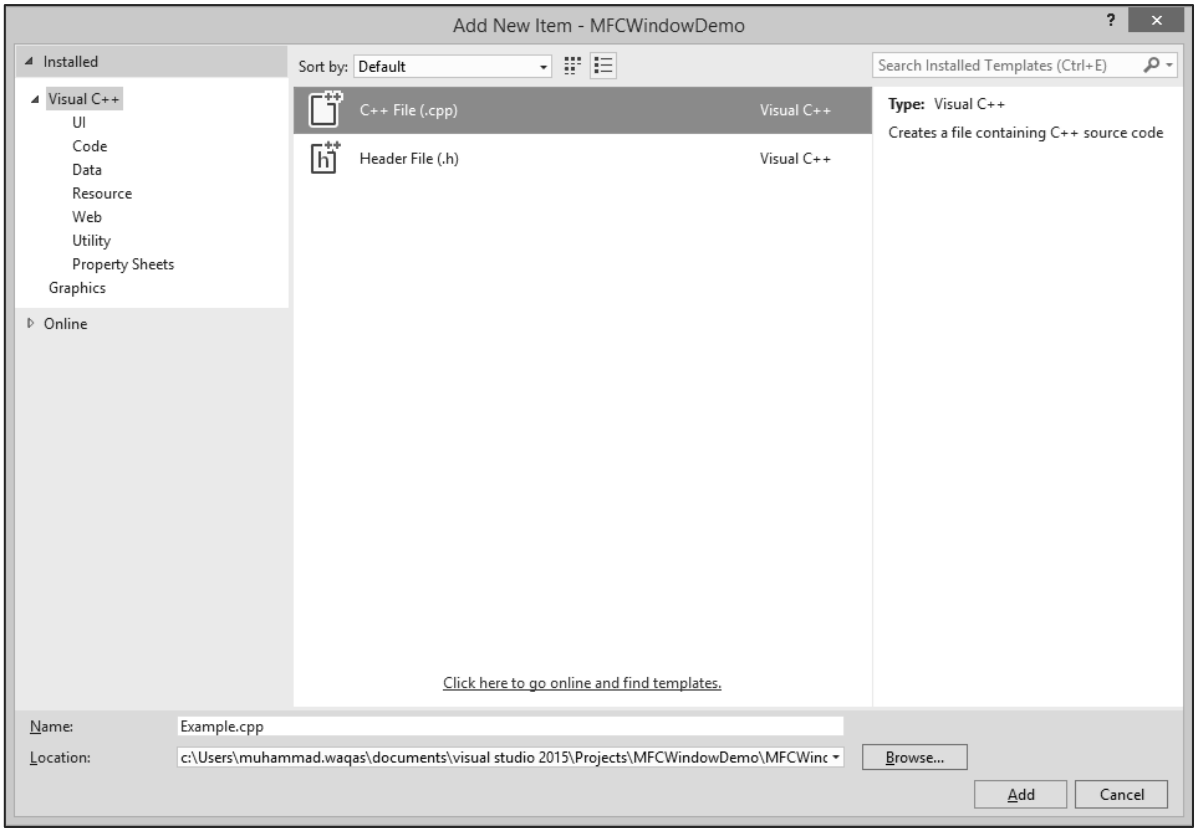
**Step 12:** Add a new source file.

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

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



tutorialspoint
SIMPLYEASYLEARNING




**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;
      }
};
```



tutorialspoint
SIMPLYEASYLEARNING

**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;
      }
};
```



tutorialspoint
SIMPLYEASYLEARNING

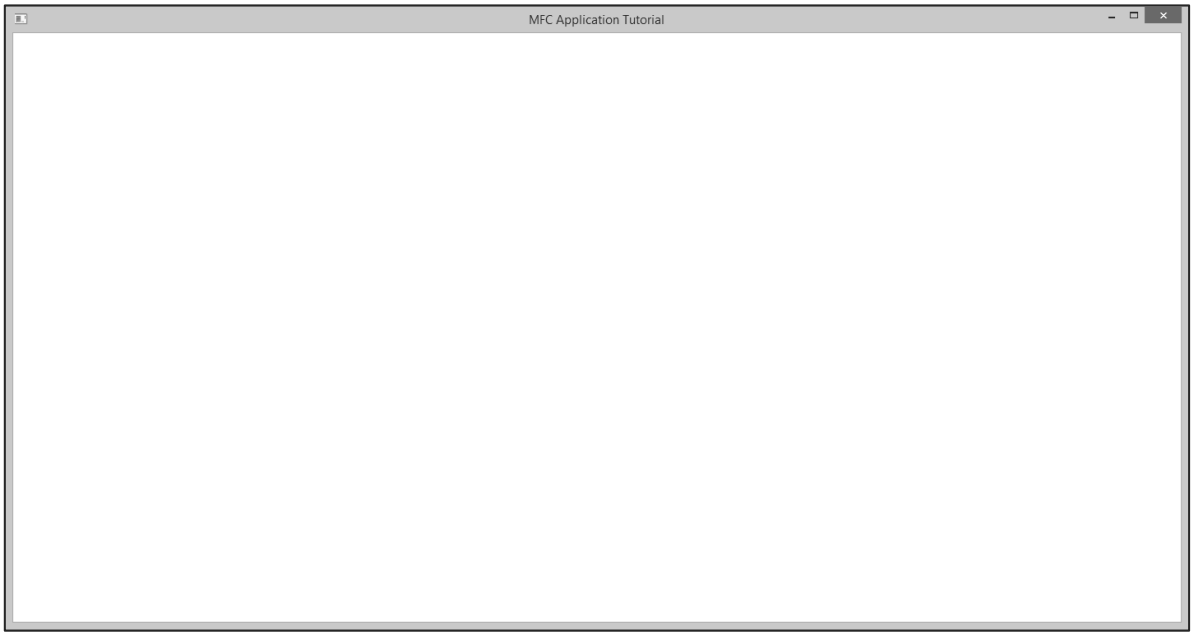
**Step 2:** Following is the complete implementation of Example.cpp file.

```
#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;
```



tutorialspoint
SIMPLYEASYLEARNING

**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 |
|---|---|
| **WS_BORDER** | Creates a window that has a border. |
| **WS_CAPTION** | Creates a window that has a title bar (implies the WS_BORDER style). Cannot be used with the WS_DLGFRAME style. |
| **WS_CHILD** | Creates a child window. Cannot be used with the WS_POPUP style. |
| **WS_CHILDWINDOW** | Same as the WS_CHILD style. |
| **WS_CLIPCHILDREN** | Excludes the area occupied by child windows when you draw within the parent window. Used when you create the parent window. |
| **WS_CLIPSIBLINGS** | 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. |
| **WS_DISABLED** | Creates a window that is initially disabled. |
| **WS_DLGFRAME** | Creates a window with a double border but no title. |
| **WS_GROUP** | 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 |



tutorialspoint
SIMPLYEASYLEARNING

| | |
|---|---|
| | 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). |
| **WS_HSCROLL** | Creates a window that has a horizontal scroll bar. |
| **WS_ICONIC** | Creates a window that is initially minimized. Same as the WS_MINIMIZE style. |
| **WS_MAXIMIZE** | Creates a window of maximum size. |
| **WS_MAXIMIZEBOX** | Creates a window that has a Maximize button. |
| **WS_MINIMIZE** | Creates a window that is initially minimized. For use with the WS_OVERLAPPED style only. |
| **WS_MINIMIZEBOX** | Creates a window that has a Minimize button. |
| **WS_OVERLAPPED** | Creates an overlapped window. An overlapped window usually has a caption and a border. |
| **WS_OVERLAPPED WINDOW** | Creates an overlapped window with the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles. |
| **WS_POPUP** | Creates a pop-up window. Cannot be used with the WS_CHILD style. |
| **WS_POPUPWINDOW** | 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. |
| **WS_SIZEBOX** | Creates a window that has a sizing border. Same as the WS_THICKFRAME style. |
| **WS_SYSMENU** | Creates a window that has a Control-menu box in its title bar. Used only for windows with title bars. |
| **WS_TABSTOP** | 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. |
| **WS_THICKFRAME** | Creates a window with a thick frame that can be used to size the window. |
| **WS_TILED** | Creates an overlapped window. An overlapped window has a title bar and a border. Same as the WS_OVERLAPPED style. |
| **WS_TILEDWINDOW** | 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. |
| **WS_VISIBLE** | Creates a window that is initially visible. |
| **WS_VSCROLL** | 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:

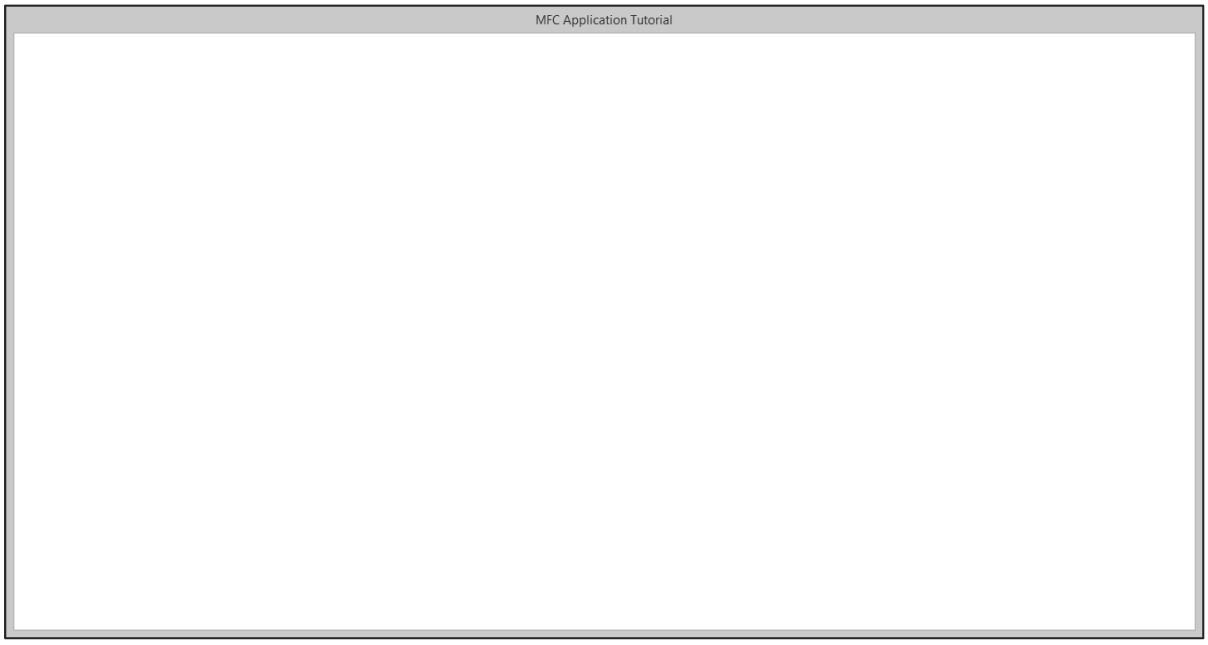
```
class CMyFrame : public CFrameWnd
{
public:
      CMyFrame()
      {
```



tutorialspoint
SIMPLYEASYLEARNING

```
            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.



tutorialspoint
SIMPLYEASYLEARNING

- 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);
```



tutorialspoint
SIMPLYEASYLEARNING

## 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);
```

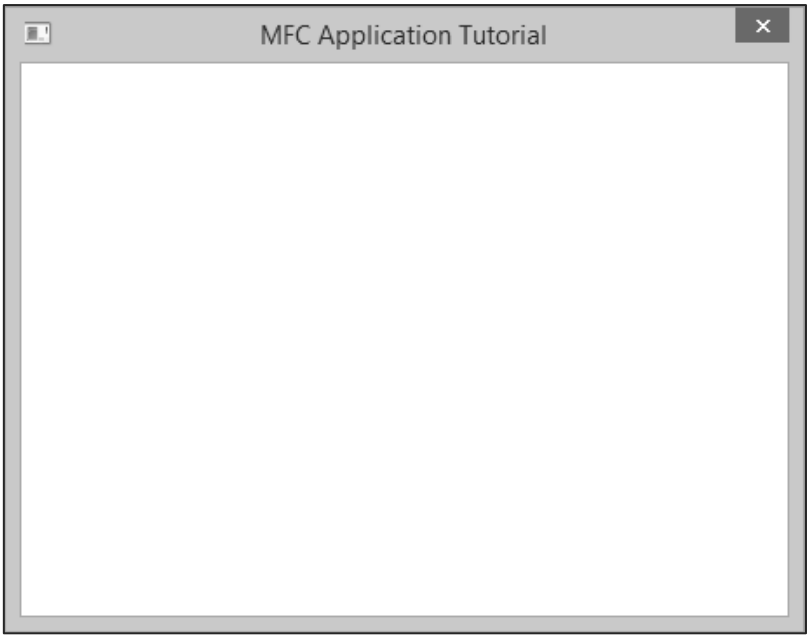
Let us look into a simple example in which we will specify the location and the size of the window

```
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.



tutorialspoint
SIMPLYEASYLEARNING




## 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.

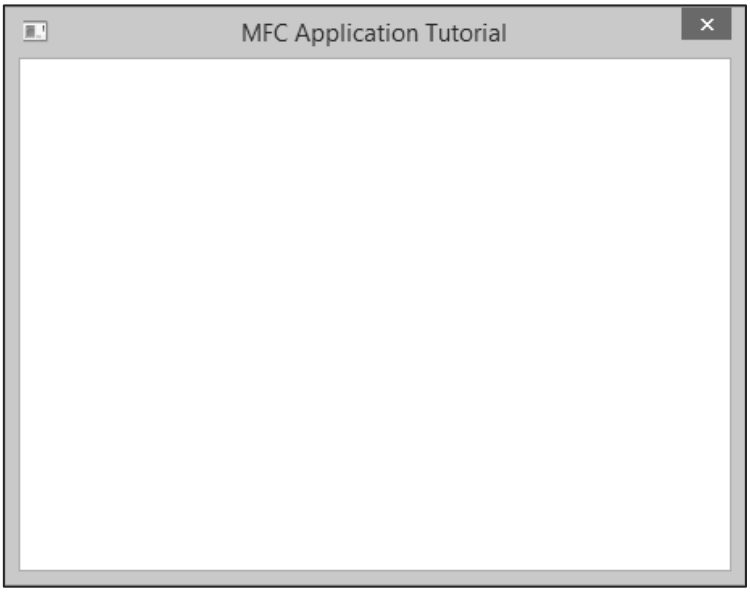
Let us look into an example which has only one Window, and there is no parent Window available, so we will pass the argument with NULL value as shown in the following code:

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



tutorialspoint
SIMPLYEASYLEARNING

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






tutorialspoint
SIMPLYEASYLEARNING

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**



tutorialspoint
SIMPLYEASYLEARNING