

Hands-On Lab

Ribbon with MFC - Native

* 1. Lab version: 1.1.0
  2. Last updated: 1/6/2011

Contents

[Overview 3](#_Toc282093898)

[Exercise 1: Creating a Default Ribbon in an Application 4](#_Toc282093899)

[Task 1 – Create a new MFC project with Ribbon in Visual Studio 2010 4](#_Toc282093900)

[Exercise 2: Adding Simple Controls to an Existing Ribbon 12](#_Toc282093901)

[Task 1 – Adding Simple Controls to an Existing Ribbon 12](#_Toc282093902)

[Task 2 – Add Event Handlers to the Controls 16](#_Toc282093903)

[Exercise 3: Adding and Using More Controls to an Existing Ribbon 17](#_Toc282093904)

[Task 1 – Build the Existing Ribbon Application to See the Ribbon Controls 17](#_Toc282093905)

[Task 2 – Disable and Enable a Button at Run Time 19](#_Toc282093906)

[Task 3 – Display the Currently Selected Combo Box Item 20](#_Toc282093907)

[Task 4 – Draw a Rectangle in the View when a Button is Clicked 22](#_Toc282093908)

[Task 5 – Change the Color of the Rectangle 23](#_Toc282093909)

[Task 6 – Move the Slider to Zoom the Rectangle 25](#_Toc282093910)

[Summary 27](#_Toc282093911)

Overview

* 1. This tutorial is intended for MFC developers who are developing desktop applications and want to take advantage of the new MFC Ribbon framework. The tutorial steps you through how to add a default Ribbon to a small application, edit various Ribbon controls in Ribbon Designer, and then use the controls at run time. You will learn how easily MFC Ribbon framework allows you to edit and use Ribbon controls. When you are finished, you will have performed all the steps necessary to add and customize a basic Ribbon in an application.
  2. The tutorial will involve real-time compiling of code and copying code from this document. In the event that a copying error (or any other problem) prevents the application from compiling, you can find fully completed samples in the tutorial package, along with the final source code for each exercise. These samples can be used to unblock compiling errors.

# Prerequisites

* 1. You must have the following items to complete this lab:
  + Microsoft Visual Studio 2010
  + Windows 7

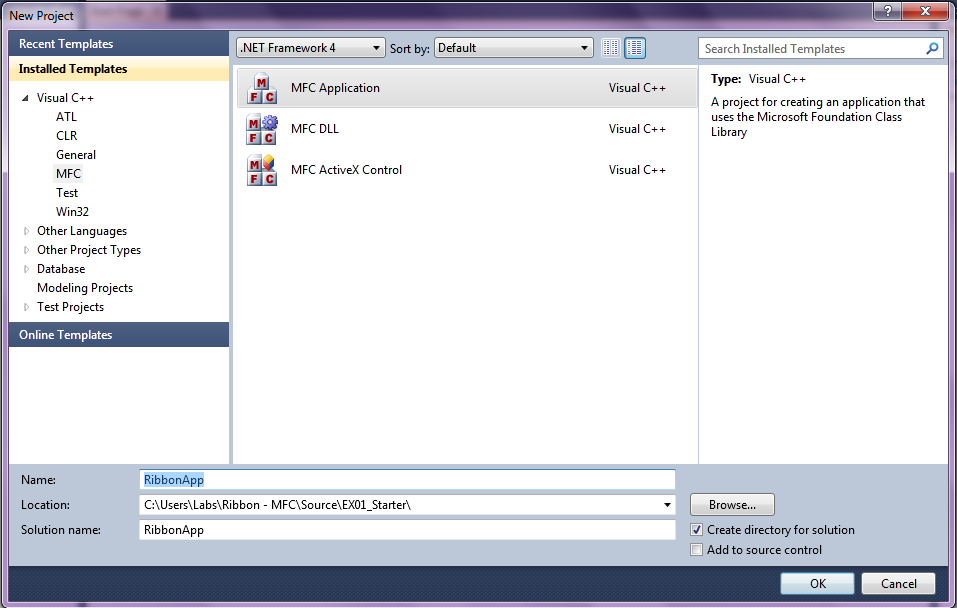
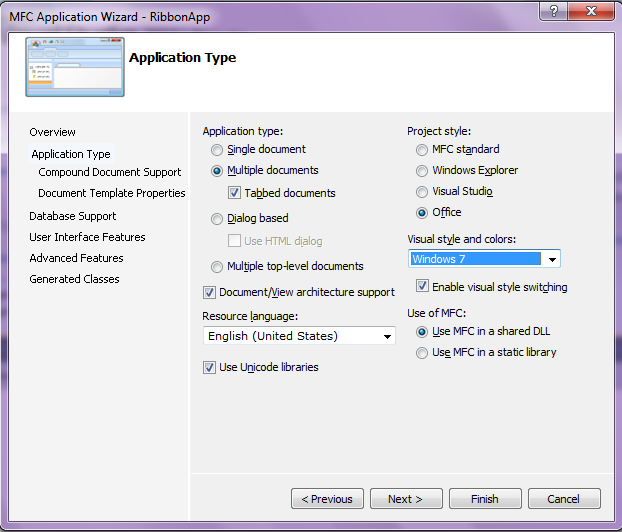
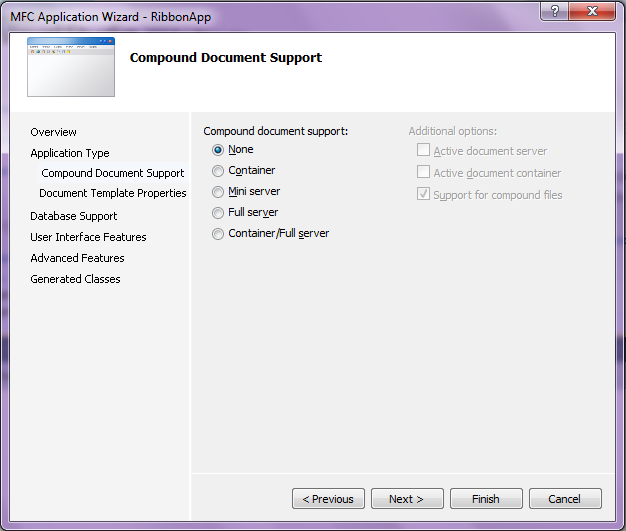
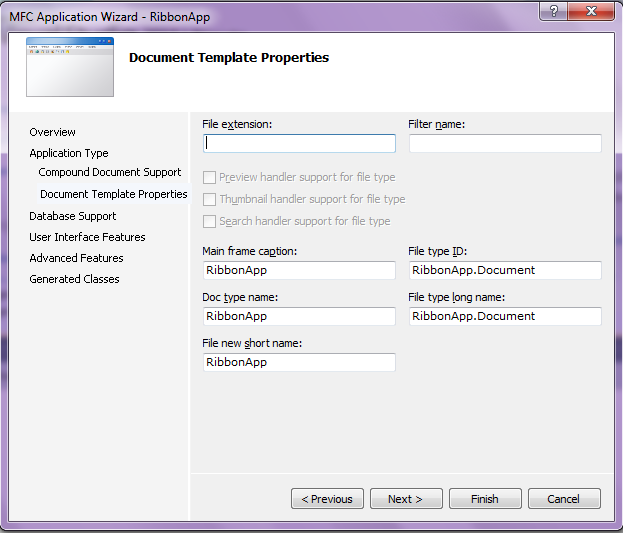
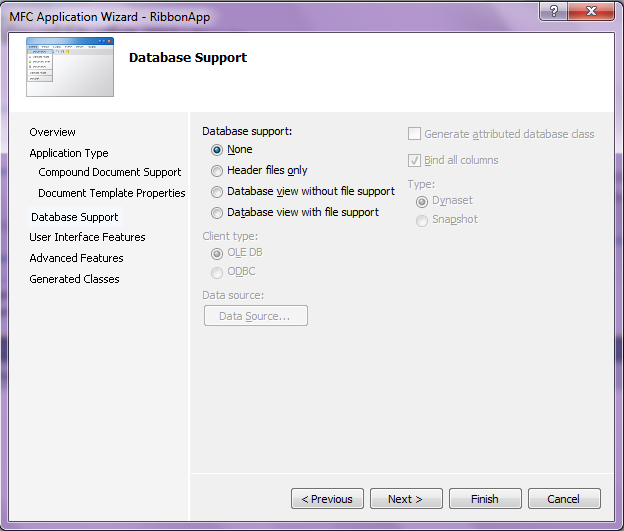
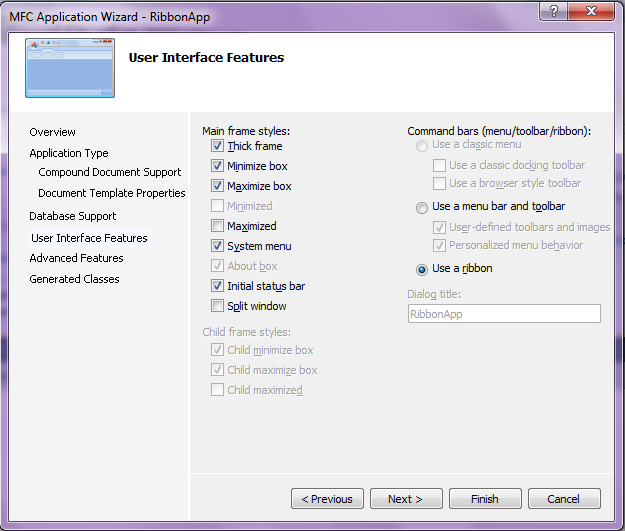
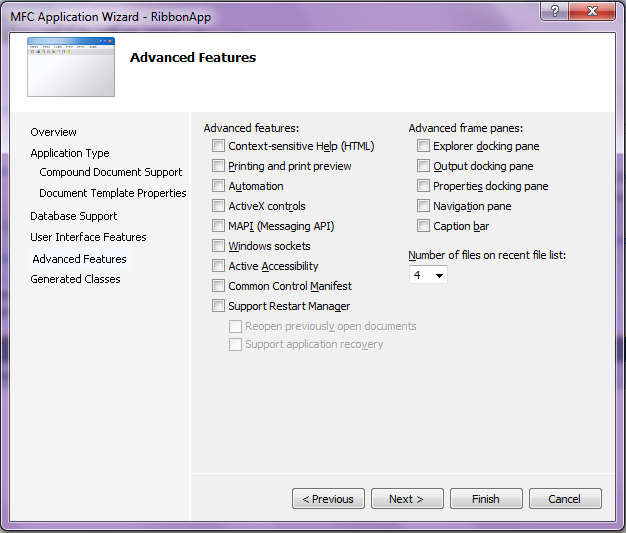
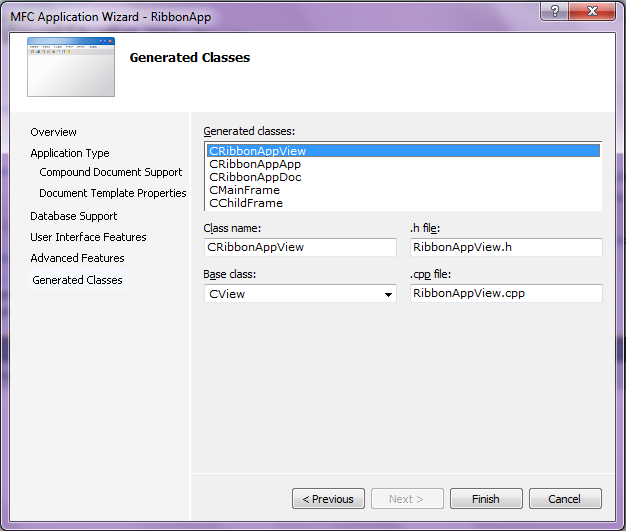
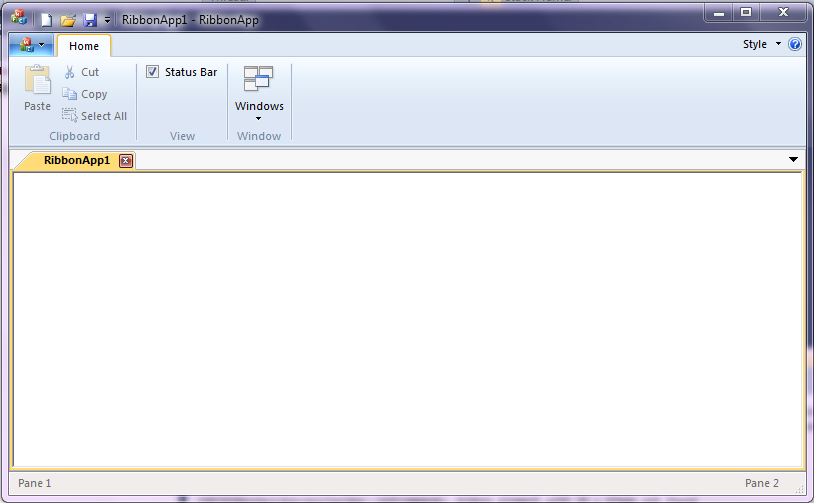
# Learning Objectives

* 1. While completing this tutorial, you will learn how to:
  + Create a default Ribbon for your application in Visual Studio® 2010
  + Add Ribbon controls such as buttons, checkbox and chunks in Ribbon Designer
  + Add Event handler functions for Ribbon control in Ribbon Designer
  + Use the Ribbon controls at run time

Exercise 1: Creating a Default Ribbon in an Application

* 1. In this exercise, you will start by creating a MFC application with Ribbon in Windows 7 style from scratch in Visual Studio 2010. You do not need to write any code to do that. Just click, click and click!

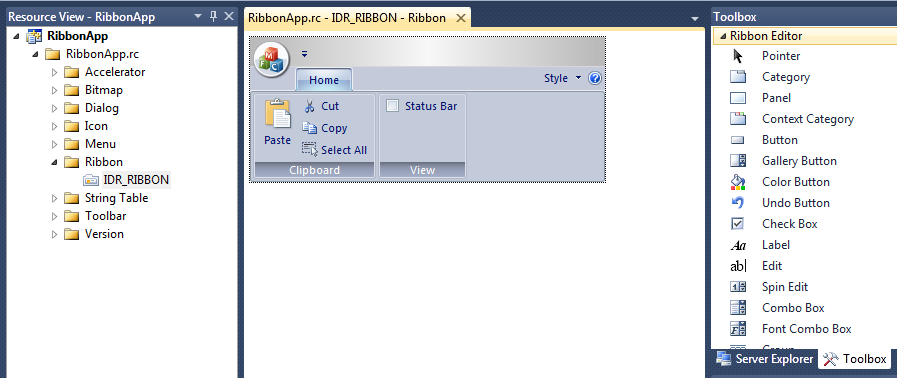
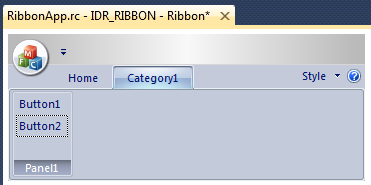
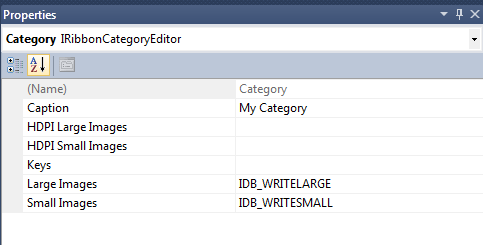
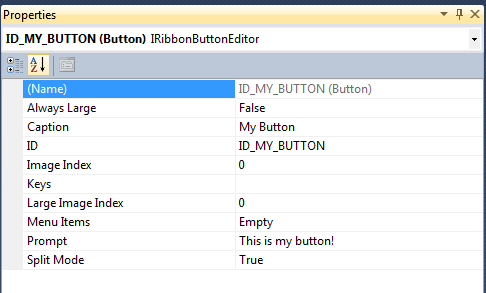
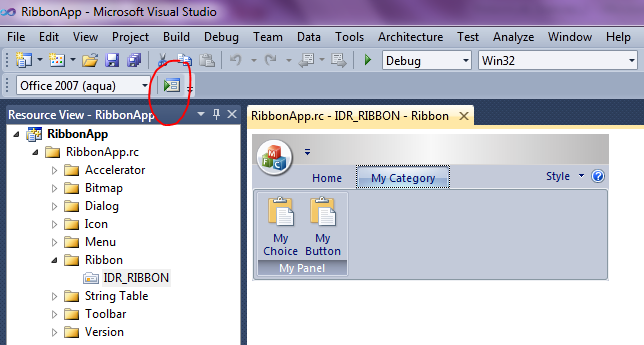
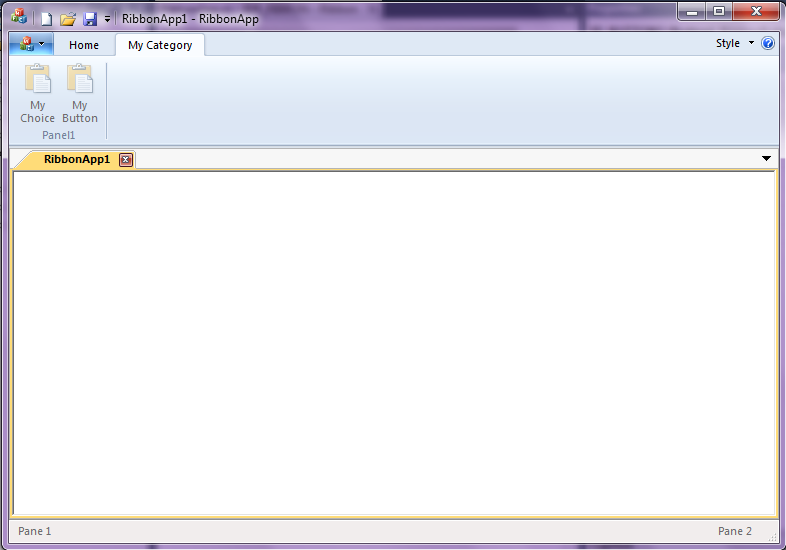
Task 1 – Create a new MFC project with Ribbon in Visual Studio 2010

* 1. In Visual Studio 2010, use the MFC Application Wizard to create an MFC application that has a default ribbon. To run the wizard, on the File menu, point to New, and then select Project. The New Project dialog box will be displayed.
  2. In the New Project dialog box, expand the Visual C++ node in the Project Types pane and select MFC. In the Templates pane, select MFC Application. Type a name for the project, such as RibbonApp, and click OK. The MF Application Wizard will be displayed. Please refer to Figure 1-1.
     1. 
     2. Figure 1-1
  3. Click **Next** in the Welcome screen of the MFC Application Wizard.
  4. In the **Application Type** screen, change **Project Style** to “*Office*”, and change **Visual style and colors** to “*Windows 7*”. Leave everything else as it is, and click **Next**. Please refer to Figure 1-2.
     1. 
     2. Figure 1-2
  5. In the **Compound Document Support** screen, leave the default setting (the “*None*” checkbox is checked). Please refer to Figure 1-3.
     1. 
     2. Figure 1-3
  6. In the **Document Template Properties** screen, leave everything at the default. Click **Next**. Please refer to Figure 1-4.
     1. 
     2. Figure 1-4
  7. In the **Database Support** screen, leave everything at the default settings (“*None*” should be selected). Click **Next**. Please refer to Figure 1-5.
     1. 
     2. Figure 1-5
  8. In the **User Interface Features** screen, make sure that the **Use a ribbon** option is selected. Click **Next**. Please refer to Figure 1-6.
     1. 
     2. Figure 1-6
  9. In the **Advanced Features** screen, clear all options. Click **Next**. Please refer to Figure 1-7.
     1. 
     2. Figure 1-7
  10. In the **Generated Classes** screen, leave everything at the default settings. Click **Finish**. Please refer to Figure 1-8.
      1. 
      2. Figure 1-8
  11. To build the application, on the **Build** menu, select **Build Solution**. If the application builds successfully, run the application by selecting **Start Debugging** from the **Debug** menu. The wizard will automatically create a Ribbon in Windows 7 Style with one Ribbon category that is named Home. This category contains three ribbon panels, which are named Clipboard, View and Window. Please refer to Figure 1-9.
      1. 
      2. Figure 1-9

Exercise 2: Adding Simple Controls to an Existing Ribbon

* 1. Visual Studio 2010 has a new feature called Ribbon Designer for MFC application, which makes it easy to add and edit any Ribbon controls to your application.
  2. In this exercise, you will learn how to drag a control in Ribbon Designer and how to add an event handler easily. For this exercise, you will use the RibbonApp solution from EX02\_Starter\Begin\RibbonApp folder.

Task 1 – Adding Simple Controls to an Existing Ribbon

* 1. Launch RibbonApp.sln in Visual Studio 2010.
  2. In the **Solution Explorer**, double-click the **RibbonApp.rc** file to open the **Resource View** pane.
  3. In the Resource View, expand RibbonApp.rc -> Ribbon -> IDR\_RIBBON node. Double-click the file to open the Ribbon Designer. The Resource View panel, Ribbon Bar Designer panel, and Toolbox panel are shown in Figure 2-1.
     1. 
     2. Figure 2-1
  4. Drag a Category control from the Toolbox to the Ribbon Bar. In the Designer, you will see a Category named “*Category1*” and a Panel in Category1 named “*Panel1*”. Drag two Button controls from the Toolbox to “*Panel1*”. The buttons are named “*Button1*” and “*Button2*” by default. The Ribbon Bar has a new look, which is shown in Figure 2-2.
     1. 
     2. Figure 2-2
  5. Right-click “*Category1*”, and select Properties to open the Properties pane. As shown in the Property window in Figure 2-3, you can easily rename the Caption to “*My Category*”, and edit the other properties. In the Property pane, select “*IDB\_WRITELARGE*” from the dropdown list of Large Images, and select “*IDB\_WRITESMALL*” from the dropdown list of Small Images. This defines the image collection for the elements in this category.
     1. 
     2. Figure 2-3
  6. Open the Properties pane of the buttons you just created, and change the captions to “My Button” and “*My Choice*”.
  7. To add an image to your buttons, in the Properties pane for each of the buttons click **Large Image Index** (you will see an ellipsis button () when you select the value field, which will allow you to browse the image collection.) Click the ellipsis button (), and select “*0*” in the Image Collection dialog. Do the same for the **Image Index** property. Finally, edit the other properties as shown in Figure 2-4.
     1. 
     2. Figure 2-4
  8. Test your Ribbon by clicking the “Test Ribbon” button on the toolbar, as shown in Figure 2-5.
     1. 
     2. Figure 2-5
  9. Build and Run your application. Now, it has a new look as shown in Figure 2-6. The two buttons that you created (“*My Choice*” and “*My Button*”) are disabled by default. The buttons will be enabled once you add a click event handler for each of them.
     1. 
     2. Figure 2-6

Task 2 – Add Event Handlers to the Controls

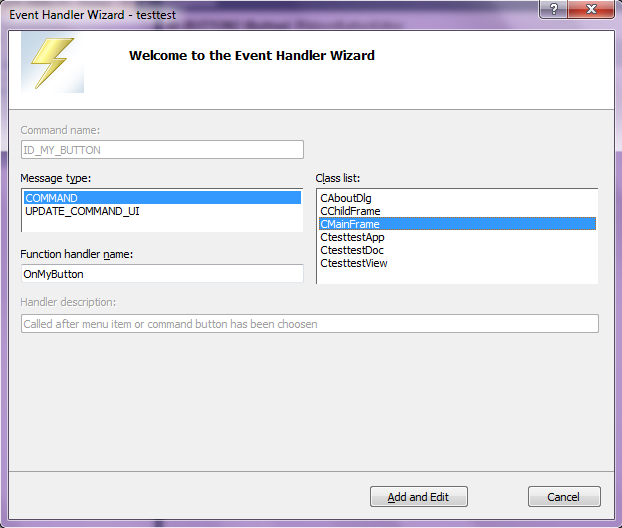
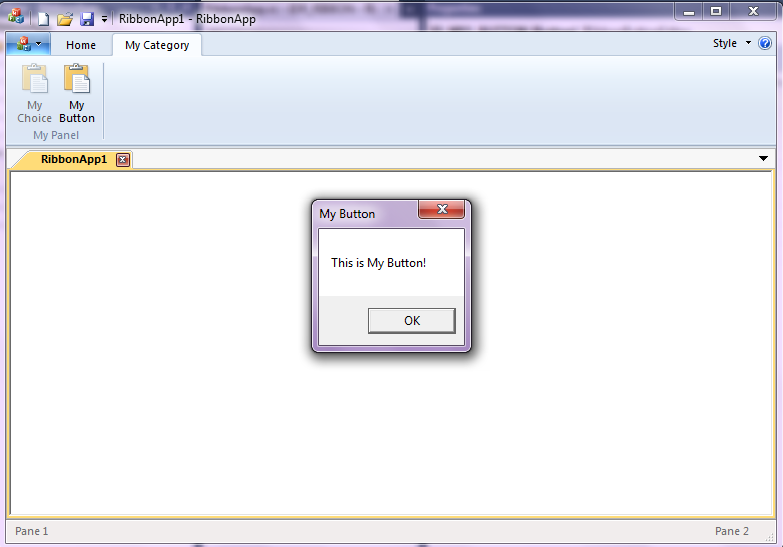
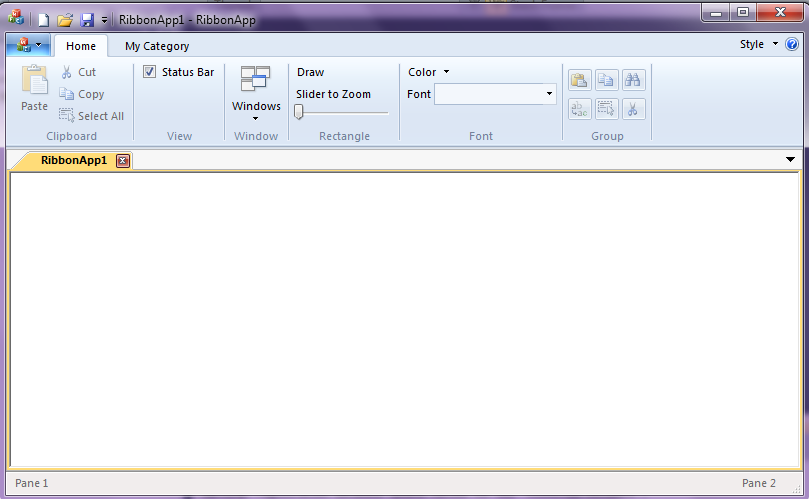
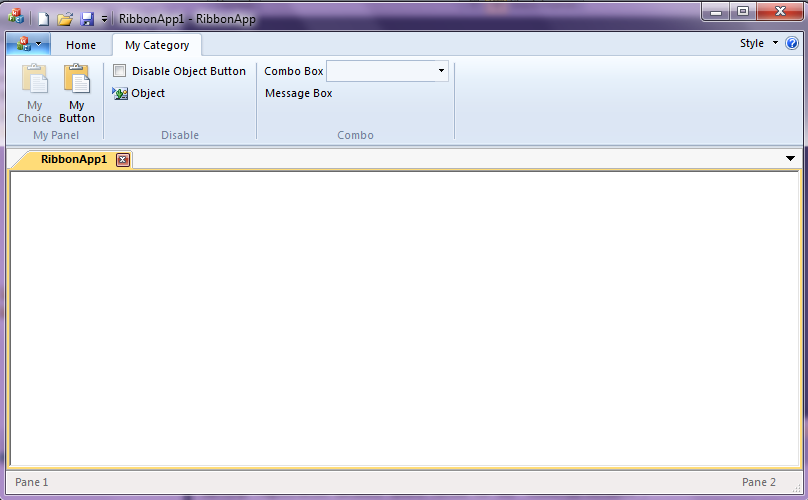
* 1. It is very easy to add event handlers in the Ribbon Designer. Right-click the “*My Button*” button control, and select “Add Event Handler”. The Event Handler Wizard will open. Type “*OnMyButton*” in the **Function handler name** textbox, select “*Command*” from the **Message type** list and “*CMainFrame*” from the **Class list**, as shown in Figure 2-7.
     1. 
     2. Figure 2-7
  2. Click “Add and Edit”. The wizard will generate some functions automatically for you, and then open the source code editor. Here, you will see the empty function **CMainFrame::OnMyButton()**in MainFrm.cpp.
  3. Add the following line of code to the OnMyButton method in Mainfrm.cpp.
     1. C++
     2. **MessageBox(TEXT("This is My Button!"), TEXT("My Button"), MB\_OK);**
  4. Build and Run your solution. Your application will have a Ribbon Bar with one Category and two buttons. Click the “*My Button*” button, and test your application as shown in Figure 2-8.
     1. 

Figure 2-8

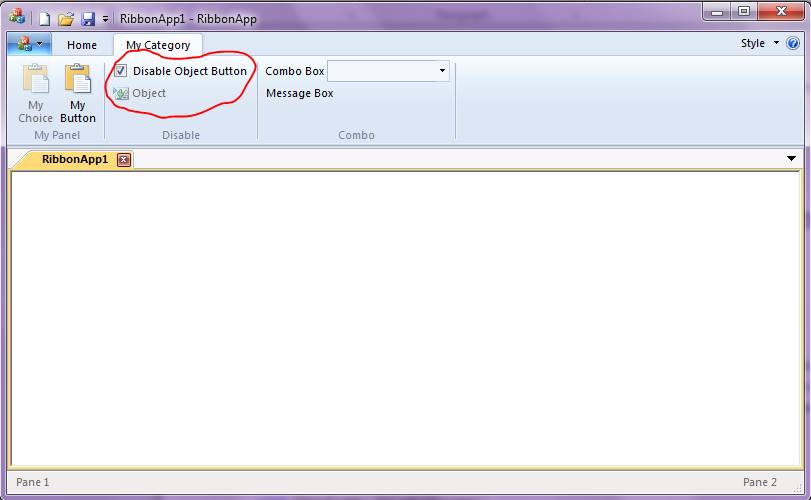
Exercise 3: Adding and Using More Controls to an Existing Ribbon

Exercise 1 demonstrated how to use the MFC Wizard to create an application with a default ribbon. Exercise 2 demonstrated how to use Ribbon Designer to add or modify Ribbon controls easily. This exercise will demonstrate more about Ribbon controls, and focus on how to use Button, CheckBox, Slider, and ComboBox MFC Ribbon controls at run time.

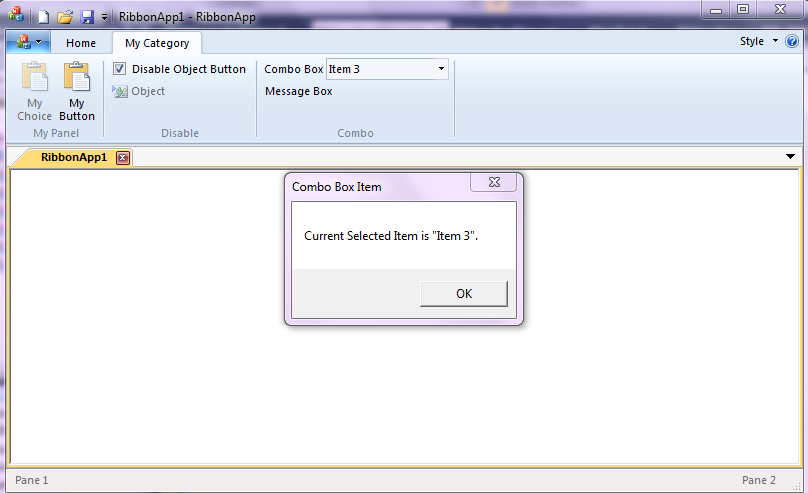
Task 1 – Build the Existing Ribbon Application to See the Ribbon Controls

* 1. Open the RibbonApp solution from the EX03\_Starter\Begin\RibbonApp folder.
  2. On the **Build** menu, click **Rebuild Solution**.
  3. On the **Debug** menu, click **Start Debugging**.
  4. The application as shown in Figure 3-1 will open. All of the ribbon controls shown here were created using the Ribbon Designer. (Exercise 2 shows how to use the Ribbon Designer.)
     1. Figure 3-1
  5. Click the “*My Category*” tab, and you will see the ribbon change, as shown in Figure 3-2.
     1. 
     2. Figure 3-2
     3. **Note:** For your convenience, some variables and functions (such as event handler functions and helper functions) are already declared and defined for you in this project, so you only need to focus on the implementation in the following tasks.
  6. Close the application.

Task 2 – Disable and Enable a Button at Run Time

* 1. You can add code manually to disable or enable a button. As shown in Figure 3-2, the *Disable* panel of *My Category* includes one checkbox and one button. You can trigger an event by selecting or clearing the checkbox. Selecting the checkbox will disable the Object button. The variables have been declared for you, along with some necessary functions.
  2. To update the event handler functions for the checkbox control in MainFrm.cpp, enter the following code inside the **CMainFrame::OnDisableCheckbox** and **CMainFrame::OnUpdateDisableCheckbox** methods:
     1. C++
     2. void CMainFrame::OnDisableCheckbox()
     3. {
     4. **m\_bChecked = !m\_bChecked;**
     5. }
     6. void CMainFrame::OnUpdateDisableCheckbox(CCmdUI \*pCmdUI)
     7. {
     8. **pCmdUI->SetCheck(!m\_bChecked);**
     9. }
  3. You must also update an event handler function for the button object in MainFrm.cpp as follows:
     1. C++
     2. void CMainFrame::OnUpdateDisableObject(CCmdUI \*pCmdUI)
     3. {
     4. **pCmdUI->Enable(m\_bChecked);**
     5. }
  4. Build and run your application. If you select the “*Disable Object Button*” checkbox, the “Object” button will be disabled, as shown in Figure 3-3.
     1. 
     2. Figure 3-3

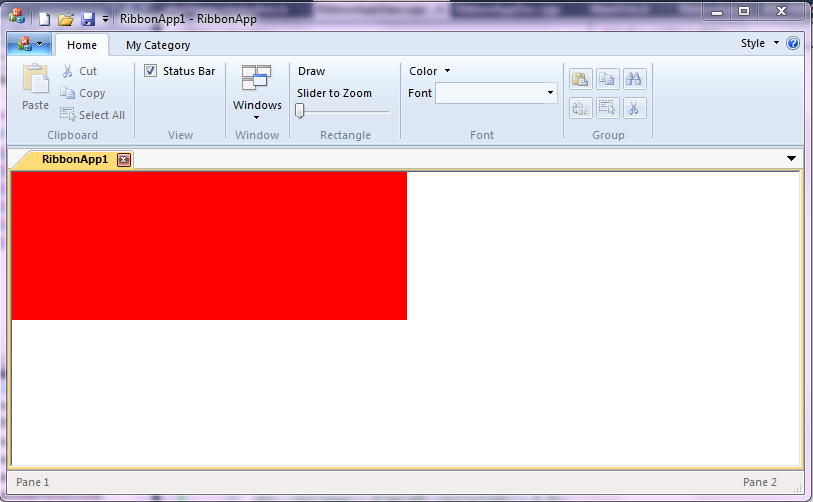
Task 3 – Display the Currently Selected Combo Box Item

* 1. Update the CMainFrame::OnComboMessage method in Mainfrm.cpp, so that a message box will be displayed when an item is selected from the dropdown list in the combo box:
     1. C++
     2. void CMainFrame::OnComboMessage()
     3. {
     4. **CMFCRibbonComboBox\* pFontComboBox = DYNAMIC\_DOWNCAST(CMFCRibbonComboBox, m\_wndRibbonBar.FindByID(ID\_COMBO\_BOX));**
     5. **// Get the selected index**
     6. **int nCurSel =pFontComboBox->GetCurSel();**
     7. **if (nCurSel >= 0)**
     8. **{**
     9. **CString item=pFontComboBox->GetItem(nCurSel);**
     10. **CString sMessage = \_T("");**
     11. **sMessage.Format(\_T("Current Selected Item is \"%s\"."),item);**
     12. **MessageBox(sMessage, \_T("Combo Box Item"), MB\_OK);**
     13. **}**
     14. **else**
     15. **{**
     16. **MessageBox(\_T("Please select one item from droplist of Combo Box."), \_T("Combo Box Item"), MB\_OK);**
     17. **}**
     18. }
  2. Build and run the application.
  3. Select an item from the combobox, and click the “*Message Box*” button. A message box will be displayed, as shown below in Figure 3-4.
     1. 
     2. Figure 3-4

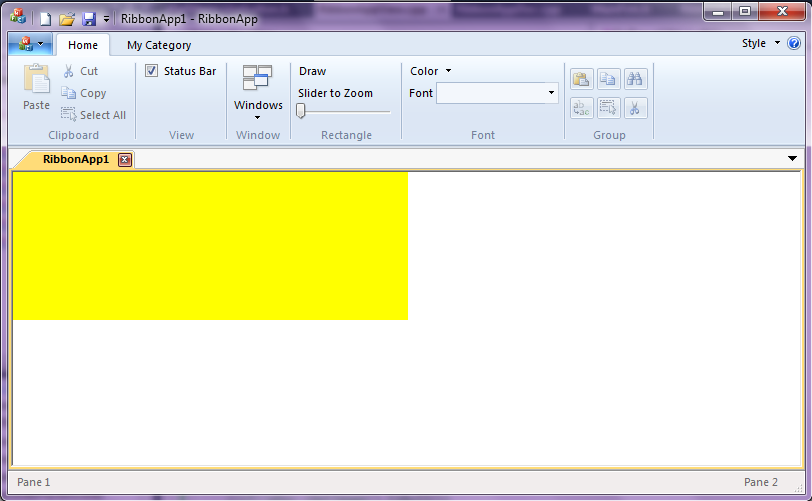
Task 4 – Draw a Rectangle in the View when a Button is Clicked

* 1. The Rectangle panel on the Home category includes a Draw button (see Figure 3-1). Update the following constructor and methods in the **RibbonAppDoc.cpp** file as shown below, so that a rectangle will be drawn when the Draw button is clicked.
     1. C++
     2. CRibbonAppDoc::CRibbonAppDoc()
     3. {
     4. **m\_bDraw = FALSE;**
     5. }
     6. bool CRibbonAppDoc::EnableDraw(void)
     7. {
     8. **return m\_bDraw;**
     9. }
     10. void CRibbonAppDoc::OnRectDraw()
     11. {
     12. **m\_bDraw = TRUE;**
     13. **UpdateAllViews(NULL);**
     14. }
  2. In the **RibbonAppView.cpp** file, add the following lines to the **CRibbonAppView::OnDraw** function after the comment “//TODO: add draw code for native data here”, as follows:
     1. C++
     2. **// Draw a rectangle**
     3. **CRect client;**
     4. **CBrush brush;**
     5. **GetWindowRect(&client);**
     6. **if (pDoc->EnableDraw() && brush.CreateSolidBrush(RGB(255,0,222)))**
     7. **{**
     8. **int width=client.Width()/2; // to make it smaller**
     9. **int height= client.Height()/2;**
     10. **CRect rect=CRect(0,0, width, height);**
     11. **pDC->FillRect(rect, &brush);**

**}**

* 1. Build and run the solution. Click the Draw button to see the application as shown in Figure 3-5.
     1. 
     2. Figure 3-5

Task 5 – Change the Color of the Rectangle

* 1. The Color combobox in the Font panel controls the color of the rectangle. To enable the Color control, replace the implementation of the **CRibbonAppDoc::GetColor** method in the **RibbonAppDoc.cpp** file with the following code:
     1. C++
     2. COLORREF CRibbonAppDoc::GetColor(void)
     3. {
     4. **CMFCRibbonBar\* pRibbon = ((CMDIFrameWndEx\*) AfxGetMainWnd())->GetRibbonBar();**
     5. **ASSERT\_VALID(pRibbon);**
     6. **CMFCRibbonColorButton\* pColor = DYNAMIC\_DOWNCAST(**
     7. **CMFCRibbonColorButton, pRibbon->FindByID(ID\_FONT\_COLOR));**
     8. **// Get the selected color**
     9. **return pColor->GetColor();**
     10. }
  2. Update the event handler function of **CRibbonAppDoc::OnFontColor** in **RibbonAppDoc.cpp** file.
     1. C++
     2. void CRibbonAppDoc::OnFontColor()
     3. {
     4. **UpdateAllViews(NULL);**
     5. }
  3. In the **RibbonAppView.cpp** file, update the **CRibbonAppView::OnDraw** method by replacing the line “*brush.CreateSolidBrush(RGB(255,0,222))*” to the following one inside the **if** clause:
     1. C++
     2. **brush.CreateSolidBrush(pDoc->GetColor())**
  4. Build and run the application.
  5. Click the Draw button, and select the color Yellow from the Color combo box to see the new UI shown in Figure 3-6.
     1. 
     2. Figure 3-6

Task 6 – Move the Slider to Zoom the Rectangle

* 1. Update the helper method **CRibbonAppDoc::GetSliderFactor** in the **RibbonAppDoc.cpp** file as follows:
     1. C++
     2. // Return the factor of zooming the rectangle
     3. double CRibbonAppDoc::GetSliderFactor(void)
     4. {
     5. **// Get a pointer to the ribbon bar**
     6. **CMFCRibbonBar\* pRibbon = ((CMDIFrameWndEx\*) AfxGetMainWnd())->GetRibbonBar();**
     7. **ASSERT\_VALID(pRibbon);**

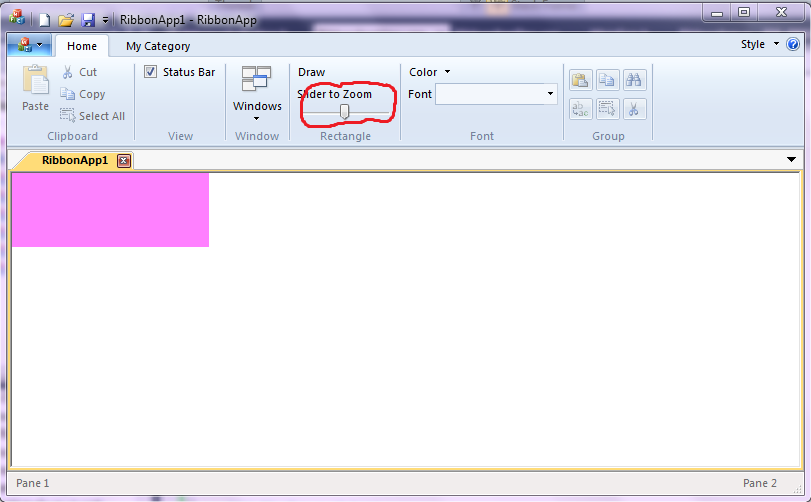
**CMFCRibbonSlider\* pSlider = DYNAMIC\_DOWNCAST(CMFCRibbonSlider, pRibbon->FindByID(ID\_RECT\_SLIDER));**

* + 1. **// Get current position**
    2. **int position =pSlider->GetPos();**
    3. **return (double)position/(double)pSlider->GetRangeMax();**

}

* 1. Update the event handler function of the Slider as follows:
     1. C++
     2. void CRibbonAppDoc::OnRectSlider()
     3. {
     4. **if(GetAsyncKeyState(VK\_LBUTTON)==0)**
     5. **{**
     6. **UpdateAllViews(NULL);**
     7. **}**
     8. }
  2. In the **RibbonAppView.cpp** file, update the first **if** clause in the **CRibbonAppView::OnDraw** method to enable the Slider function.
     1. C++
     2. **if (pDoc->EnableDraw() && brush.CreateSolidBrush(pDoc->GetColor()))**
     3. **{**
     4. **int width=client.Width()/2; // to make it smaller**
     5. **int height= client.Height()/2;**
     6. **double factor=pDoc->GetSliderFactor();**
     7. **if (factor)**
     8. **{**
     9. **width=width\*factor;**

**height=height\*factor;**

* + 1. **}**
    2. **CRect rect=CRect(0,0, width, height);**
    3. **pDC->FillRect(rect, &brush);**
    4. **}**
  1. Build and run the application.
  2. Click the Draw button, and then move the slider. When the Slider is moved, the rectangle will be resized using a zoom factor, as shown in Figure 3-7.
     1. 
     2. Figure 3-7
     3. **Note:** By the end of this exercise, the application you build with EX03\_Starter\Begin\RibbonApp\RibbonApp.sln should be the same as the application built using EX03\_Starter\End\RibbonApp\RibbonApp.sln.

Summary

* 1. You have successfully completed the MFC Ribbon introductory tutorial. You have learned how the new MFC Ribbon Framework can help you to quickly and easily create a Ribbon application using the MFC wizard and Ribbon Designer.