MFC GUIs 1: Example code and tutorial

Building Graphical User Interfaces (GUIs)

Using the Microsoft Foundation Classes (MFC)

All of these examples assume that Microsoft Visual C++ 2017 is the compiler being used. Based on a tutorial written by Dr. Rick Coleman.

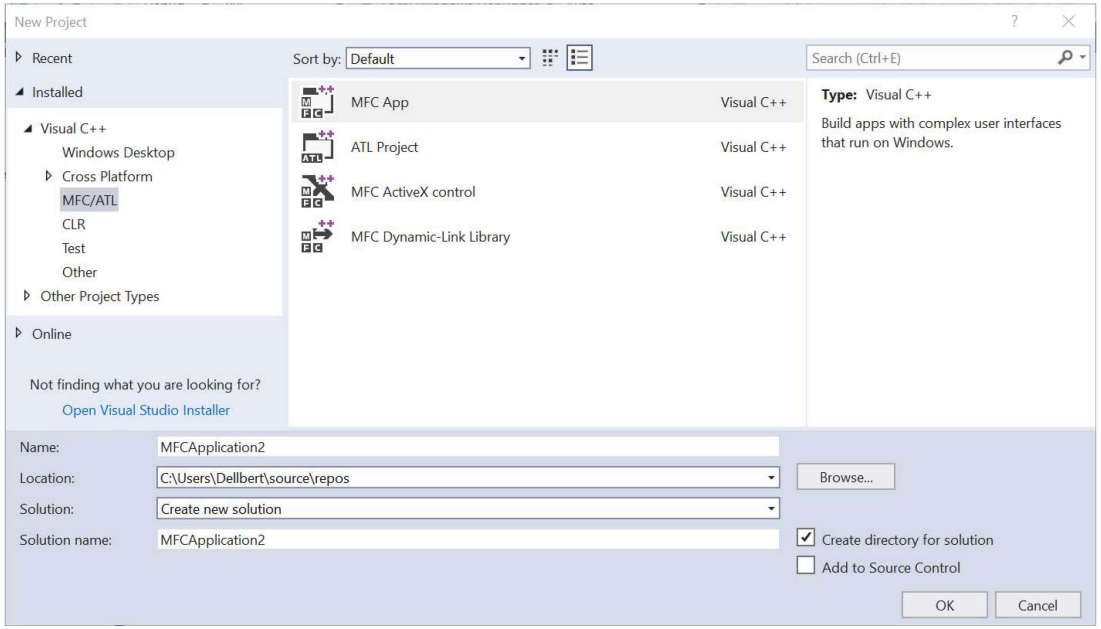
You also must have the "Visual C++ MFC for x86 and x64" feature enabled in the VS Installer.

Exercise 3: A dialog based application

Dialog windows may be of two types. The first is **modal** meaning it must be closed before you can do anything else in the application. The best known examples of modal dialog boxes are the Open and Save file dialog boxes. The second type of dialog is **Non-modal or Modeless** which can stay open even if you shift focus to another window of the application. In this exercise you will build a dialog based application in which the main window is a non-modal dialog box. The main window is an instance of a CDialog which is derived from CWnd.

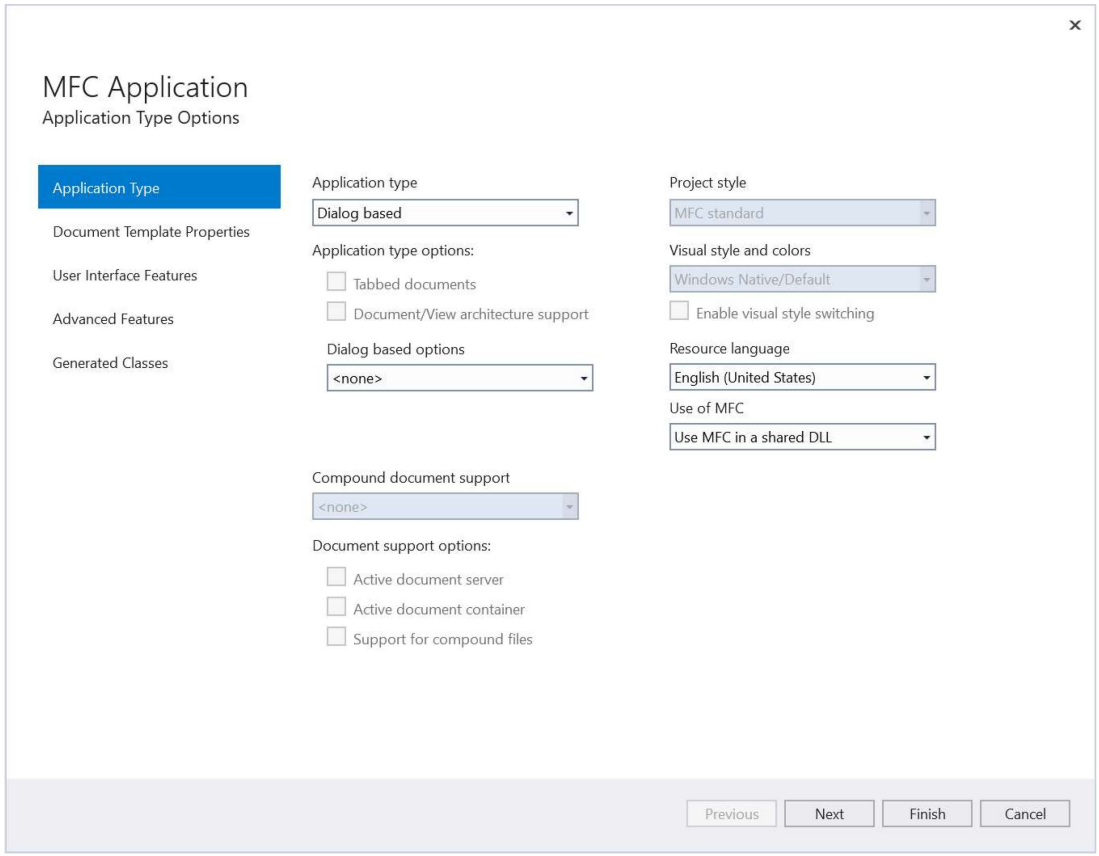
Creating the MFC Project

To create an MFC project, bring up the standard "New Project" dialog in VS 2017 and configure it as follows.

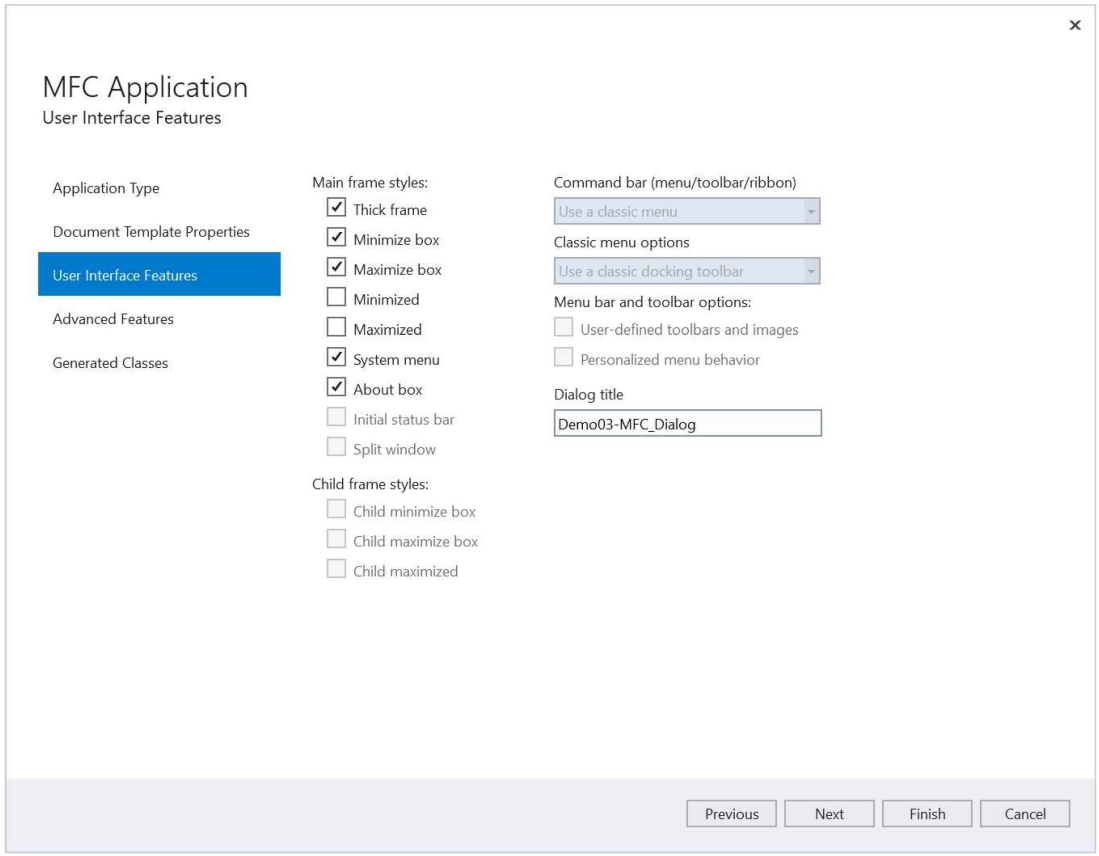
1. 

2. Click OK. That will bring up some more dialog boxes to configure the MFC project. Configure it according to the next screenshots.

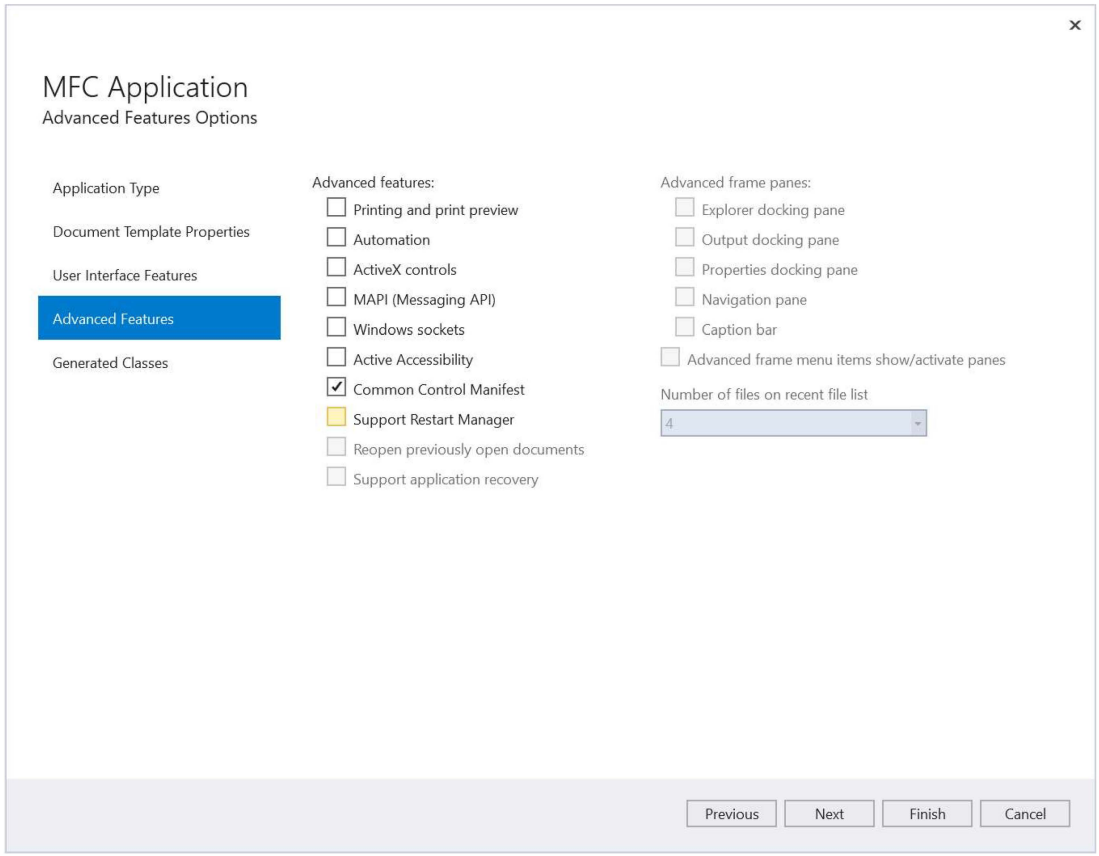
3.


MFC Application
Application Type Options
Application Type
Document Template Properties
User Interface Features
Advanced Features
Generated Classes
Application type
Dialog based
Application type options:
Tabbed documents
Document/View architecture support
Dialog based options
<none>
Compound document support
<none>
Document support options:
Active document server
Active document container
Support for compound files
Project style
MFC standard
Visual style and colors
Windows Native/Default
Enable visual style switching
Resource language
English (United States)
Use of MFC
Use MFC in a shared DLL
Previous
Next
Finish
Cancel

4.


MFC Application
User Interface Features
Application Type
Document Template Properties
User Interface Features
Advanced Features
Generated Classes
Main frame styles:
Thick frame
Minimize box
Maximize box
Minimized
Maximized
System menu
About box
Initial status bar
Split window
Child frame styles:
Child minimize box
Child maximize box
Child maximized
Command bar (menu/toolbar/ribbon)
Use a classic menu
Classic menu options
Use a classic docking toolbar
Menu bar and toolbar options:
User-defined toolbars and images
Personalized menu behavior
Dialog title
Demo03-MFC_Dialog
Previous
Next
Finish
Cancel

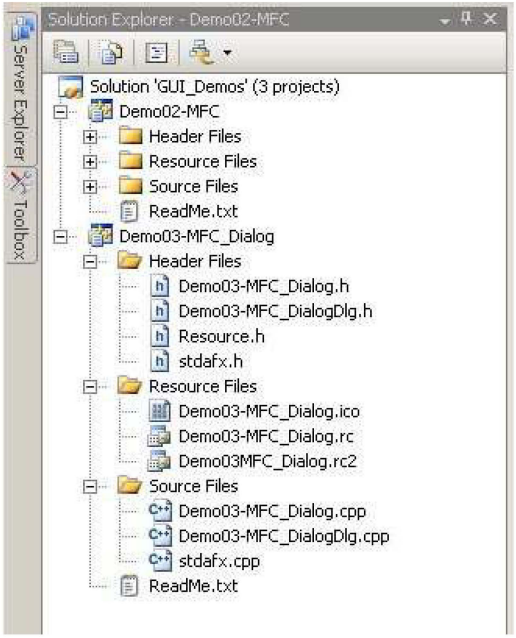
5.



6. Next, click "Finish" and Visual Studio will automatically generate a bunch of code.

The Files Visual Studio Created

Take a look at the files created automatically for you when you created the MFC project. The image below shows the listing of those files from the demonstration program shown in class.



The Application File - *projectname*.cpp

Open this file and take a look at these items:

- The Message Map macros **BEGIN_MESSAGE_MAP** and **END_MESSAGE_MAP** These enclose all of the message handlers for the application. Visual Studio will add to these as you add handlers. The arguments passed to **BEGIN_MESSAGE_MAP** are the name of the application class and its super class. Note that the application class name has been created from: 'C' + ProjectName + "App".

```
BEGIN_MESSAGE_MAP(CDemo03MFC_DialogApp, CWinApp)
     ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()
```

- The Constructor. Normally you would initialize class variables, etc. here, however, just before the dialog window is displayed there is a call to the InitInstance() function where you should do most of your major initializations.

```
CDemo03MFC_DialogApp::CDemo03MFC_DialogApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

- The app instance. This is a static instance of your dialog application.

```
// The one and only CDemo03MFC_DialogApp object
CDemo03MFC_DialogApp theApp;
```

- **InitInstance()** - In this function you perform any major initializations needed.

The Dialog File - *projectname*Dlg.cpp

Open this file and take a look at these items:

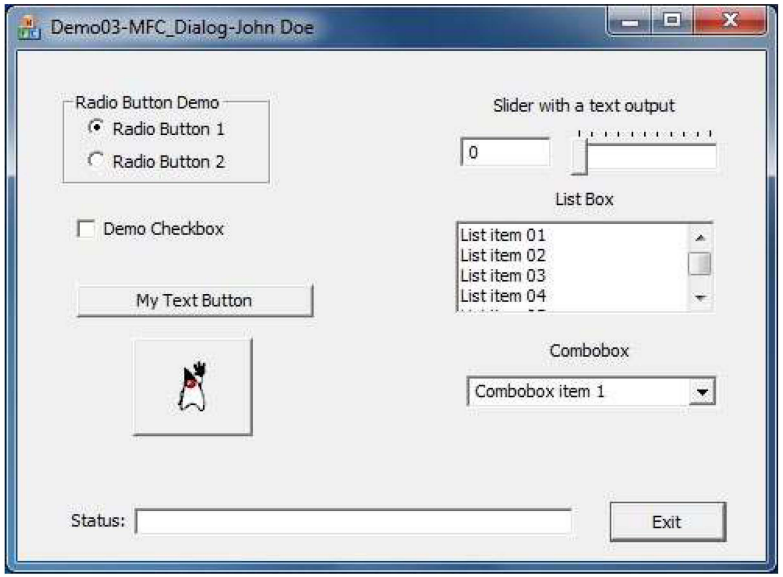
- The about dialog - The first section defines the class for the About dialog box if you select the "About" option from a help menu.
- The main application dialog class is defined next.
- The Message Map (see detailed description above) uses macros to create the same functionality as the WndProc function in the API examples. After you build the GUI come back and look at this.
- **OnInitDialog()** - This is where you can do any initializations of the dialog widgets, such as adding items to combo boxes or list boxes.
- **OnSysCommand()** - Handles the WM_COMMAND message.
- **OnPaint()** - Handles the WM_PAINT command to repaint the screen.
- As you add event handlers for each of the widgets in the GUI the callback functions will be automatically created here. You will need to add the appropriate code to handle each of these events.

Building the GUI

If the resource editor is not already open showing the dialog window, open it by expanding the ***applicationName*** item in the Resource View pane. Expand the **Dialog** folder and double click the **IDD_*APPLICATION_NAME*_DIALOG** resource. If the Resource View is not visible make it so by selecting View->Other Windows->Resource View.

Look at the left or right edge of the Visual Studio window and find where the **Toolbox** tab is located. When you click this it expands to display all the widgets you can add to the dialog box. There are a number of catagories of widgets. You want to expand the **Dialog Editor** widgets.

You can now build your GUI by clicking a widget in the list then clicking in the dialog box where you would like to creat the widget. Below you will find notes on how to create the demonstration dialog box shown in this image. You may copy it or feel free to experiment with your own layout and design.



OK and Cancel Buttons

By default the resource editor adds an OK and Cancel button to the dialog. You can edit or delete either of these. Both automatically close the dialog box when you create an event handler for them.

1. Delete the cancel button. Click it in the resource editor and then press the Delete key.
2. Change the text on the OK button to Exit.
    1. Right click the button and select Properties from the drop down menu. (The Properties can also be found as a tab in the margin of the main Visual Studio window.)
    2. Locate the **Caption** property and change it to **Exit**.
3. Double click the button to automatically add the code for an event handler function. Look at the bottom of the ***ApplicationName*Dlg.cpp** file and you should see a function: **void *ApplicationName*Dlg::OnBnClickedOk()**. If this function does not already have the code in it to terminate the application (some versions of Visual Studio do not automatically add it) then add the single line of code **OnOK();**.

Radio Button Groups

1. Create a Group Box widget first.
2. Create the radio buttons inside the group. (If you create the buttons first you can select all of them, press ctrl-X to cut them, then select the group box, finally press ctrl-V to paste the radio buttons inside the group box. Note: this does not move the location of the radio buttons it just links them to each other via the group box. This also does not group the widgets so they can be moved as a unit in the visual editor.)
3. To set the automatic selection feature, select the first button in the group and set its Group property to True. Set the Group property of all others buttons to False.
4. Right click the first radio button and select **Add Variable**.
5. In the dialog box give the variable a name, like m_RadioButton1 and click OK. **If you see a check box labeled "Control Variable" make sure it is checked so you can create a variable of type CButton.**
6. Look at the code in the ***ApplicationName*Dlg.cpp** file in the function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_RADIO1, m_RadioButton1);
```

7. In the OnInitDialog() function initialize the radio button to selected with

```
m_RadioButton1.SetCheck(1);
```

8. Double click each radiobutton widget to create an event handler for it.

Checkbox widget

1. Create a check box widget.
2. Right click the check box widget and select **Add Variable**.
3. In the dialog box give the variable a name, like m_CheckBox and click OK
4. Look at the code in the ***ApplicationName*Dlg.cpp** file in the function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_CHECK1, m_CheckBox);
```

5. You can now use the variable to get the state with:

```
int state = m_CheckBox.GetCheck();
```

   If this returns 1 the check box is checked, if it returns 0 it is unchecked.
6. Double click the check box widget to create an event handler.

Text Button

1. Create a button. Set its text string in the properties pane to something like "My Text Button".
2. Double click the button widget to create an event handler.

Icon Button

1. Create an icon resource by right clicking on "Resources" in the Solution Explorer, expanding the "Add" menu, selecting the "Resource" option, and selecting "Icon" in the dialog that pops up.
2. This will bring up a new icon with a number of different sizes and color depths of icons. Delete all of these by, on each one, right clicking and selecting "Delete."
3. Once you've deleted all the existing icons, create a new one by righting clicking in the (now empty) image type list and selecting the "New Image Type" option. In the image type list, select the "32x32 24 bit" option.
4. Copy the image shown here to the clipboard. (In most browsers, you can do this by right clicking on the image and selecting "Copy Image.") 
5. In the visual editor for the icon resource, select the 32x32 24 bit image type we created.
6. Paint over all of the teal background to make it a uniform background color (you can use the rectangle fill to do this).
7. Paste the copied image into the icon. It will be much too large to fit but you can use the handles (the tiny squares at the corners and center of each side) and drag to resize the image to fit. Be careful that you do not unselect the icon before resizing it as you cannot re-select the entire image then.
8. Save the icon.
9. Back in the dialog box visual editor create a square button. Set its **Caption** property in the properties pane to an empty string.
10. Set its **Icon** property in the properties pane to **true**.

11. Right click the button and select **Add Variable**.
12. In the dialog box give the variable a name, like m_IconButton and click OK
13. In the OnInitDialog () function initialize the icon for the button with:

```
if (m_IconButton.GetIcon() == NULL)
	m_IconButton.SetIcon(LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_ICON1)));
```

    You will substitute your icon ID for "IDI_ICON1".
14. Double click the button widget to create an event handler.

Slider widget with text display

1. Add a slider widget.
2. Right click the slider and select **Properties**.
3. If your version of Visual Studio lets you set the minimum and maximum values set these to 0 and 100 respectively.
4. If your version of Visual Studio lets you set the tick interval set this to 10.
5. If your version of Visual Studio has ***Tick Marks*** as a property for the slider, set this to **true**.
6. Add an Edit Control text box next to the slider.
7. Right click the text box widget and select **Add Variable**.
8. In the dialog box give the variable a name, like m_SliderState and click OK.
9. Look at the code in the ***ApplicationName*Dlg.cpp** file at function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_EDIT1, m_SliderState);
```

10. Right click the slider widget and select **Add Variable**.
11. In the dialog box give the variable a name, like m_Slider and click OK.
12. Look at the code in the ***ApplicationName*Dlg.cpp** file at function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_SLIDER1, m_Slider);
```

13. Double click the slider widget to create an event handler function in ***ApplicationName*Dlg.cpp**.
14. Look for the function **OnNMCustomdrawSlider1()**. Each time the slider changes you can display its reading with:

```
int x = m_Slider.GetPos(); // Get current position
char str[32];
sprintf(str, "%d", x);  // Create the string
this->m_SliderState.SetWindowText(str); // Set the text box
```

    Don't forget to add **#include <string>** at the top of this file so you can use sprintf.
15. To initialize the slider object add the following code to the OnInitDialog() function in ***ApplicationName*Dlg.cpp**:

```
// Init the range for the slider
m_Slider.SetRange(0, 100);
// Set tic marks at an interval of 10
for(int i=0; i<=90; i+=10)
	m_Slider.SetTic(i);
```

Listbox widget

1. Add a list box widget.
2. Right click list box and select **Add Variable**
3. In the dialog box give the variable a name, like m_ListBox and click OK.
4. Look at the code in the ***ApplicationName*Dlg.cpp** file at function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_LIST1, m_ListBox);
```

5. Using the variable add several items to the list box in OnInitDialog(). For example:

```
m_ListBox.AddString("List item 01");
m_ListBox.AddString("List item 02");
m_ListBox.AddString("List item 03");
m_ListBox.AddString("List item 04");
m_ListBox.AddString("List item 05");
m_ListBox.AddString("List item 06");
m_ListBox.AddString("List item 07");
m_ListBox.AddString("List item 08");
m_ListBox.AddString("List item 08");
m_ListBox.AddString("List item 10");
```

6. Double click the list box widget to create an event handler.
7. In the event handler get the current selected string with:

```
char str[32];
m_ListBox.GetText(m_ListBox.GetCurSel(), str);
```

Combobox widget

1. Add a combo box widget.
2. Left click the drop down button of the combobox. Use the gray handle that appears in the center bottom of the combobox to set the size of the drop down display box. This is important or you wont see anything.
3. Right click the combo box and select **Add Variable**.
4. In the dialog box give the variable a name, like m_ComboBox and click OK.
5. Look at the code in the ***ApplicationName*Dlg.cpp** file at function doDataExchange(). You will see that it has added a line like:

   ```
   DDX_Control(pDX, IDC_COMBO1, m_ComboBox);
   ```

   Using the variable add several items to the combo box in OnInitDialog(). For example:

```
m_ComboBox.AddString("ComboBox item 1");
m_ComboBox.AddString("ComboBox item 2");
m_ComboBox.AddString("ComboBox item 3");
m_ComboBox.AddString("ComboBox item 4");
m_ComboBox.AddString("ComboBox item 5");
```

1. Set the initial selection in the combobox with:

```
m_ComboBox.SetCurSel(0);
```

2. Double click the combo box widget to create an event handler.
3. In the event handler get the current selected string with:

```
char str[32];
m_ComboBox.GetLBText(m_ComboBox.GetCurSel(), str);
```

4. In the visual editor click the down arrow on the right side of the combobox. The selection handles will change. The handle in the middle bottom will be solid black. Use this handle to stretch the limits of the combobox. This will set the size of the combobox when it is clicked to open it when the application is running. If you do not set this it will be difficult to select anything. If the limit you set is not enough to display all the options in the combobox then a scroll bar will automatically appear on the right side of the combobox when it is dropped down.

Status text widget

1. Add an Edit Control text box at the bottom of the dialog box.
2. Right click the text box widget and select **Add Variable**.
3. In the dialog box give the variable a name, like m_StatusTextBox and click OK.
4. Look at the code in the ***ApplicationName*Dlg.cpp** file at function doDataExchange(). You will see that it has added a line like:

```
DDX_Control(pDX, IDC_EDIT2, m_StatusTextBox);
```

5. You can now use the variable to access the text box and set and get the text in any of the handler functions with:

```
m_StatusTextBox.SetWindowText("some text");
// OR
char str[64];
m_StatusTextBox.GetWindowText(str, 63);
m_StatusTextBox.SetWindowText(str);
```

6. Go through each of the other handler functions and have them print in the m_StatusTextBox the action that has just occurred. For example, have the icon button handler print "Icon button clicked."

Set Window Title

1. In the ***ApplicationName*Dlg.cpp** locate the function **OnInitDialog()**
2. After all your other initializations and before the **return true;** line add the following, changing **Demo03-MFC_dialog** to the name of your application:

```
        // Set window title
        this->SetWindowTextA("Demo03-MFC_Dialog");
```

Compile and run your application. Play with the widgets to see how they work. Try adding and experimenting with the other widgets

For more information on any of the widgets try doing a Google search on the widget name **+MFC**.