# FundamentalsOfBuilding\_NET\_DesktopApplications

This document contains notes pertaining to the Pluralsight course “Fundamentals Of Building .NET Desktop Applications”. Some of the clips of the Pluralsight course are documented. The names of those clips are copied to the section headers in this document and to the table below. Using Microsoft Word, you can use the names in the table as hyperlinks to navigate to any particular clip. But using Apache Open Office, these hyperlinks do not work; instead, they merely serve as a table of contents. You can navigate to the start of any clip via bookmarks; type F5 to bring up the Navigator; then double-click Bookmark1 for 1st clip header, Bookmark 2, for 2nd clip header, etc.

|  |  |
| --- | --- |
| [Create a Windows Forms App](#_Create_a_Windows_1) | [Build the Main Layout](#_Build_the_Main) |
| [Add a User Control for the Header](#_Add_a_User) | [Understand Data Binding and BindingSource](#_Understand_Data_Binding) |
| [Set up the BindingSource and the ViewModel](#_Set_up_the) | [Connect the ListBox to the BindingSource](#_Connect_the_ListBox) |
| [Bind the Detail Controls](#_Bind_the_Detail) | [Bind the ComboBox](#_Bind_the_ComboBox) |
| [Implement the Refresh Logic](#_Implement_the_Refresh) | [Save the Data](#_Save_the_Data) |

I have bypassed the sections pertaining to the WinUI demo and the WPF demo, and I am concentrating on the Win-Forms demo. But I will need the result of some of the work done for WinUI, which the Pluralsight course uses (in common) for WinUI, WPF, and WinForms. Therefore,

* Download *fundamentals-building-dot-net-desktop-applications.zip* from the **Exercise files** of this Pluralsight course.
* Copy *C:\Users\user\_name\Downloads\fundamentals-building-dot-net-desktop-applications.zip\*

*04\demos\after* to a place on the local computer where you will maintain the shadowed version

of the demo.

* Double-click (the copied) EmployeManager.sln to start Visual Studio. In response there will be a few minutes of delay while Visual Studio downloads some files.

It is instructive to take a brief tour of the code in the “Libraries” folder in **Solution Explorer**.

* EmployeeManager.Common/DataProvider/IEmployeeDataProvider specifies an interface, IEmployeeDataProvider, for the methods that interact with the database. In this solution IEmployeeDataProvider is implemented by the class EmployeeDataProvider (in EmployeeDataProvider.cs).
* EmployeeManager.Common/Model contains C# files that define data structures; in Entity Framework these would be the “domain classes”. (“domain” classes are C# classes that model data in a database.)
* EmployeeManager.DataAccess/EmployeeDataProvider.cs contains an in-memory impersonation of a tiny database. In a fully implemented application, this file might contain Entity-Framework code, instead.
* EmployeeManager.ViewModel contains C# classes and other code that I do not understand at present. Refer to the clips [Understand Data Binding and BindingSource](#_Understand_Data_Binding) and [Set up the BindingSource and the ViewModel](#_Set_up_the).

/\*

I have bypassed the earlier sections pertaining to the WinUI demo and the WPF demo. Somewhere in these sections the Pluralsight instructor has constructed the “EmployeeManager” solution. To fill in the gap

* Open Visual Studio. VisualStudio displays a welcome screen titled **Visual Studio 2019**.
* Click **Continue without code** - in the right-hand panel. Visual Studio displays an empty user interface; if the **Solution Explorer** is shown, it will be empty, and the central pane will also be empty.
* From the pull-down menu – at the top line – click **File -> New -> Project...** Visual Studio displays a dialog titled **Create a new project**.
* Type “blank” in the **search** box. Visual Studio displays the **Blank Solution** option in the right-hand pane.
* Select the **Blank Solution** option, and click the **Next** command button. Visual Studio displays a dialog titled **Configure your new project**.
* Type “EmployeeManager” in the **Solution name** text box.
* Navigate to the folder where you want this solution to reside. On my Windows 10 computer, I chose

C:\Users\sncole\source\GitRepositories\VSAndOtherPlatformProjects\

FundamentalsOfBuilding\_NET\_DesktopApplications\

* Click the **Create** command button. Visual Studio displays “EmployeeManager” as the solution name in the right-hand panel.

\*/

## Create a Windows Forms App

Open Visual Studio to the EmployeeManager solution. If you encounter any errors from Visual Studio while performing the following instructions, you might have an out-of-date version of Visual Studio. Before installing Visual studio updates, the following steps are recommended.

* Make certain that the operating system (Windows 10) is up-to-date.
* Reboot.
* Do Disk Cleanup.
* Reboot.
* Download and install Visual Studio update.

Now proceed as follows.

* Right-click the solution name in the **Solution Explorer**. Visual Studio displays a pop-up menu.
* Click **Add -> New Solution Folder**. Visual Studio displays a folder named “NewFolder1” in the **Solution Explorer**.
* Rename the new folder to “WinForms”.
* Right-click “WinForms” in the **Solution Explorer**. Visual Studio displays a pop-up menu.
* Click **Add -> New Project...** Visual Studio displays a dialog titled **Add a new project**.
* Make certain that the language choice (immediately under the Search for templates) box contains **C#**.
* Type “WinForms” in the search box. Visual Studio displays a list of templates in the right-hand panel.
* Scroll down the list of templates until you find **Windows Forms App**. (The name “Windows Forms App” should not be followed by any parenthetic text; blank indicates “Core”.) Click it; then click the **Next** command button. Visual Studio displays a dialog titled **Configure your new project**.
* Give the new project the name “EmployeeManager.WinForms”, and click the **Next** command button. Visual Studio displays a dialog titled **Additional information**.
* Select .NET 5.0 as the **Target Framework**. Click the **Create** command button. Visual Studio adds the template components of the new (EmployeeManager.WinForms) project. Visual Studio may, also, select one of the elements of this project to display (by default). On my computer it displays Form1.cs (Design).

Next we will take a tour of what the template provides, and gain familiarity with the designer.

* In the **Solution Explorer** rename Form1.cs to MainForm.cs. Visual Studio responds with a pop-up window containing . . .

You are renaming a file. Would you also like to perform a rename in this project of all references to the code element ‘Form1’?

* Click the **Yes** button. Visual Studio responds by renaming all items in the EmployeeManager.WinForms project to “MainForm”.
* If “MainForm.cs” is not showing (a blank windows form), double-click it in the **Solution Explorer** to make it visible.
* Double-click (to open) Program.cs. The instructions in the (static) method **main()** execute when the application starts. The last instruction starts MainForm().
* Right-click MainForm.cs in **Solution Explorer**. Then select **ViewCode** in the pop-up menu. This allows you to interact with the code pertaining to MainForm (e.g. events).
* **Solution Explorer** also shows MainForm.Designer.cs. This contains code that is automatically generated in response to design changes.
* To illustrate migrate to the designer image of MainForm. While MainForm is selected, open the **Properties** window. Note, for example, that the **Text** property is “Form1”.
* The icons on the 2nd line of **Properties** allow you to list the properties alphabetically or categorized. The lightning-bolt icon allows you to interact with events. For example, double-click **Load** in this list, and Visual Studio creates a skeleton for the event handler in MainForm.cs (code).
* Navigate to MainForm.cs (designer). Open the **Toolbox** (View -> Toolbox). Visual Studio displays a dialog titled **Toolbox.**
* If it is not already expanded, click **All Windows Forms** in **Toolbox**. Visual Studio responds by displaying a large number of controls that are available to be added to the form.
* Drag the **Button** control onto the MainForm designer.
* Navigate to **Properties**, and Visual Studio displays the properties of this **Button** control.
* Using this **Button**’s **Text** property, type the name “Save”, and Visual Studio displays “Save” on the button’s face.
* Change the control’s name using **(Name)** - immediately above the list of properties. Change the name to “btnSave”. Visual Studio confirms the name change by changing the name of the control in the **Properties** window – above the icons.
* Select the lightning-bolt icon to go to btnSave’s events. Double-click the **Click** event. Visual Studio responds by creating a skeleton for the event handler in MainForm.cs (code).
* Navigate to MainForm.Designer.cs. Toward the bottom of this file, you will note a line of code containing “Windows Form Designer Generated Code” followed by the declaration for btnSave -the button that we have just added to the form.
* Notice – at the top of the page – that the content of this page is the **partial class** MainForm; and the content of MainForm.cs (code) also contains **partial class** MainForm. These will be compiled into a single class. Therefore, the code that we might write in btnSave\_click() will have access to btnSave (and all of its properties).
* Navigate back to MainForm.Designer.cs. Click “+” to the left of “Windows Form Designer Generated Code”. Visual Studio displays the content of this block of code.
* Note - near the top of block of code – the code to create the button in the form. Also, after “// btnSave”, you will note the properties that were added via the designer, and the statement to install the button-click event handler.
* The comment at the top of the block of code warns the user not to use the code editor to modify the contents of the block via the code editor. (Modifications are almost always done by the designer.)
* But the Pluralsight instructor tells us that he does use the code editor to remove event handlers. The example that he used to illustrate this was deleting the btnSave\_click() event handler, which he did by removing the handler’s code from MainForm.cs (code). This caused VisualStudio to detect an error, because the deletion was incomplete; the statement in the hidden block in MainForm.Designer.cs has (as we observed) a statement to install the event handler. He corrected the error by removing that statement, as well. ***But I tried deleting the event handler in the button’s properties by erasing the name “btnSave\_Click”. This action caused both the event handler and its install statement to be erased.***
* The block of code is sandwiched between .SuspendLayout() and .ResumeLayout(). Without these statements, the run-time would be responding to the added controls and their properties before all of the statements in the block have been executed. This could cause the page to flicker, and it could result in additional time required to load the form and its controls.
* Delete the button from MainForm. Save the MainForm.cs (designer).
* In **SolutionExplorer** right-click the project, EmployeeManager.WinForms. Visual Studio responds with a pop-up menu.
* Click **Set as Startup Project** in the pop-up menu.
* Click from the pulldown menu **Debug -> Start Without Debugging**. Visual Studio compiles the project and runs the project – displaying the MainForm.

## Build the Main Layout

We now proceed to populate the form with controls that we pick from the T**oolbox**.

* In Visual Studio select MainForm.cs (designer). Make **Toolbox** and **Properties** visible.
* Set the **Font Size** property of MainForm to 13. This means that all controls in MainForm will inherit this size.
* The **Panel** control is used to contain a subset of the controls on a form. Select **Panel** from **Toolbox**, and drop it on the form.
* This **Panel** will be used for the header of the form. Use **Properties** to rename the panel to pnlHeader.
* Change pnlHeader’s **Dock** property to **Top**. (One can do this visually – by clicking the down arrow at the right edge, and by clicking the top area in the pop-up diagram.) Visual Studio responds by placing the panel across the top of the form.
* Make a copy of pnlHeader on the form. Select the panel; right-click, and select **Copy**; move the mouse to the center of the form; right-click, and select **Paste**.
* This copied panel will be used for navigation. Use **Properties** to rename the copied panel to pnlNavigation.
* Change pnlNavigation’s **Dock** property to **Left**. Visual Studio responds to placing the panel next to the left edge of the form.
* Resize pnlNavigation to make it narrower.
* Copy pnlNavigation (using methods described above).
* Rename this 3rd panel to pnlMainArea.
* Change pnlMainArea’s **Dock** property to **Fill**. As a result, pnlMainArea fills the remaining space in the form.
* Increase the width of pnlNavigation, so that it’s width about 20% of the form’s width.
* Drop another panel into pnlNavigation – for the refresh button. Rename this panel to pnlRefresh.
* Set pnlRefresh’s **Dock** property to **Top**.
* Drop a command button into pnlRefresh. Enlarge it.
* Notice that when pnlNavigation is resized, the **Button** does not change size. Look at the **Button**’s **Anchor** property - “Top, Left”. This means that the button’s distance to the top edge of pnlRefresh is a fixed value, and the button’s distance to the left edge of pnlRefresh is a fixed value. If we change the **Anchor** property to “Top, Left, Right”, the button will enlarge when pnlNavigation is enlarged.
* Change the height of pnlRefresh, and the position of the button, so that the button is (roughly) centered in pnlRefresh.
* Drop a **ListBox** into pnlNavigation. Set the **Dock** property of this **ListBox** to **Fill**.
* Rename the **ListBox** to lsbEmployees. Rename the **Button** to btnRefresh. Rename btnRefresh’s **Text** property to “Refresh”.
* Drop a **Label** onto pnlMainArea. Copy the label into the clipboard, and paste it twice. The result is label1, label2, and label3 on the panel. Arrange them so that they are in a vertical column, with label1 at the top, and label3 at the bottom.
* Rename label1 to lblFirstName, and set its **Text** property to Firstname.
* Rename label2 to lblEntryDate, and set its **Text** property to Entry date.
* Rename label3 to lblJobRole, and set its **Text** property to Job role.
* Drop a **TextBox** onto pnlMainArea. Place it immediately below lblFirstName. Widen it so that it looks good on the form.
* Notice that the **TextBox’s Anchor** property is Top, Left. Change **Anchor** property to Top, Left, Right. This lets the **TextBox** shrink or expand when one shrinks or expands the form horizontally.
* Rename the **TextBox** to txtFirstName.
* Just to review our renaming, click the down arrow on the **Properties** window, 2nd line. Visual Studio shows the list of the various controls.
* You can also review the parent/child relationships among the controls by selecting **View -> Other Windows -> Document Outline**. Click the name of a control to navigate to that control.
* Drop a **DateTimePicker** onto pnlMainArea. Place it immediately below lblEntryDate. Widen it so that it looks good on the form.
* Drop a **ComboBox** onto pnlMainArea. Place it immediately below lblJobRole. Widen it to the same width as txtFirstName. Change its **Anchor** property to Top, Left, Right.
* Drop a **CheckBox** onto pnlMainArea. Place it below the **ComboBox**.
* Rename the **CheckBox** to chkIsCoffeeDrinker. Set chkIsCoffeeDrinker’s **Text** property to “Is coffee drinker”.
* Drop a **Button** onto pnlMainArea. Place it below chkIsCoffeeDrinker. Resize it to make it look good.
* Rename the **Button** to btnSave. Set btnSave’s **Text** property to ”Save”.
* Change btnSave’s **Anchor** property to Left, Bottom.

Rebuild and run the application. Experiment with resizing the form.

## Add a User Control for the Header

* Right-click EmployeeManager.WinForms in the **Solution Explorer**. Visual Studio displays a pop-up menu.
* Click **Add -> New Folder.** Visual Studio displays a new folder under **Dependencies**. Rename this new folder “Controls”.
* Right-click "Controls” in the **Solution Explorer**. Visual Studio displays a pop-up menu.
* Click **Add -> User Control (Windows Forms).** Visual Studio displays a pop-up dialog titled “Add New Item – Employee Manager.WinForms” - with “A reusable Windows Forms (Win Forms) control” displayed in the right-hand panel.
* Rename this new control HeaderControl in the **Name:** text box at the bottom of the dialog.
* Click the **Add** command button in the dialog. Visual Studio adds the user control to the project; it displays HeaderControl.cs [Design] as an empty window representing HeaderControl’s visual aspect.
* Widen HeaderControl to about 3 times its default size.
* Set HeaderControl’s **BackColor** property to #F05A28 (hexadecimal RGB color code).
* Drop a **PictureBox** onto HeaderControl.
* Note a small arrow in the upper-left corner of the **PictureBox**. This is called a “smart tag”. One clicks the smart tag to gain access to the most important options of the control – instead of using **Properties**.
* Click the “smart tag”. Visual Studio pops up a dialog titled **PictureBox Tasks**.
* Click **Choose Image...** Visual Studio pops up a dialog titled **Select Resource**.

***For the next step you will need a copy of the Wired Brain Coffee logo. You can find it in the Exercise files for this course – at***

fundamentals-building-dot-net-desktop-applications.zip\03\demos\before\logo.png

***Download it to a folder on the local computer****.*

* Click the **Local resource:** radio button. Then click the **Import...** command button. Finally navigate to the local copy of logo.png, and select it. Visual Studio displays a copy of the image in the dialog’s right-hand pane.
* Click **OK** in **Select Resource**.
* Select **Zoom** from the **Size Mode:** pulldown list in **PictureBox Tasks**. Visual Studio resizes the logo, and displays it in the **Picture Box**.
* Resize the **Picture Box** to make it a little bit larger and approximately square.
* Move the **Picture Box** to the upper-left corner of HeaderControl.
* Drop a **Label** onto HeaderControl.
* Change the **Font Size** of the label to 28.
* Change the **Text** of the label to "Employee Manager”.
* Move the label to the right of the **Picture Box** - approximately aligned with the coffee cup.
* Reduce the width of HeaderControl to get rid of space on the right and bottom.
* Click the **Picture Box**; hold down Ctrl on the keyboard, and click the label. Now the Properties window allows you to change the properties for the 2-control combination (as if they were a single control).
* Change the **Anchor** property of this pair of controls to **Top** (only). Now these two controls are not anchored **Left** or **Right**. Therefore, when HeaderControl is resized wider or narrower, this pair of controls remains centered.

This completes the design of HeaderControl, and it can be used in MainForm. To use it, it is necessary to build the project. Right click EmployeeManager.WinForms, and then click **Build** in the context menu. When the **Build** succeeds, Visual Studio displays HeaderControl at the top of **Toolbox**.

* Drop HeaderControl onto pnlHeader.
* Set the **Dock** property of this (instance of) HeaderControl to **Fill**. Visual Studio displays this HeaderControl instance in the panel. But it looks a little different from what we saw when designing HeaderControl. The reason is that our chosen font size (28) is in conflict with the form’s font size (13) (partially inherited by the user control). The compromise is to “scale in the ratio of the new font size” (a quote from Pluralsight that I don’t completely understand). But there is a remedy.
* Select HeaderControl’s designer, and open **Properties**.
* Examine the **AutoScaleMode** property. By default it is **Font**; change it to **None**.
* Rebuild the project, and then examine MainForm again. Significantly improved.

Run the application. It works well cosmetically.

## Understand Data Binding and BindingSource

**BindingSource** is a class (or a tool) built into Windows Forms used to connect forms to data. **BindingSource** has two pertinent properties – **DataSource** and **Current**. **MainViewModel** is one of the classes that was added when we loaded the solution at the beginning; (see Libraries/EmployeeManager.ViewModel/MainViewModel.cs). lsbEmployees has a **DataSource**, and we will assign its value to be **BindingSource.DataSource**. Changing the selection in the List Box will automatically change the value of **BindingSource.Current**. Additional automatic value assignments will occur as a consequence of these bindings – to properties of txtFirstName and btnSave.

## Set up the BindingSource and the ViewModel

* Navigate to MainForm.cs (code).
* Right-click (in **Solution Explorer)** EmployeeManager.WinForms -> **Dependencies**. Visual Studio opens a context menu.
* Click **Add Project Reference...** Visual Studio responds by opening a dialog titled **Reference Manager**.
* Add some references: click check boxes adjacent to EmployeeManager.Common, EmployeeManager.DataAccess, and EmployeeManager.ViewModel. Click **OK.**
* In MainForm() (the constructor), append the following statement after InitializeComponent();

\_viewModel = new MainViewModel ( new EmployeeDataProvider() );

* Hover the mouse over “MainViewModel”, and type <ctrl>.; accept the invitation to add the **using** statement with regard to EmployeeManager.ViewModel.
* Hover the mouse over “EmployeeDataProvider”, and type <ctrl>.; accept the invitation to add the **using** statement with regard to EmployeeManager.DataAccess.
* Hover the mouse over “\_viewModel”, and type <ctrl>.; accept the invitation to add the declarationstatement

private MainViewModel \_viewModel;

* Hover the mouse over “\_viewModel” in the declaration, and type <ctrl>.; accept the invitation to make the \_viewModel property **readonly**.
* In MainForm\_Load() (the form-load event handler), we can use \_viewModel. The statement

\_viewModel.Load();

loads the data.

* Next we set up a **BindingSource**. It can be done in the code file or in the designer.
* Navigate to MainForm.cs (designer).
* Drop a **BindingSource** (from the toolbox) onto MainForm. Visual Studio confirms that the tool was addded by displaying a window – immediately below the MainFrom desinger window – containing bindingSource1.
* Open the **Properties** window for bindingSource1.
* Rename bindingSource1 to employeeBindingSource.
* Navigate to MainForm.cs (code).
* Add a 2nd statement to MainForm\_Load() . . .

employeeBindingSource.DataSource = \_viewModel.Employees;

## Connect the ListBox to the BindingSource

In this clip the ListBox in our application is connected to the BindingSource, which was added to MainForm in the previous clip.

* Navigate to MainForm.cs (designer).
* Select the ListBox lsbEmployees. Observe – in the **Properties** window - that the value of lsbEmployees.DataSource is (none).
* The property might be changed from (none) to employeeBindingSource in the Designer, but the Pluralsight instructor recommends that the assignment be done in MainForm.cs (code).
* Navigate to MainForm.cs (code).
* Add 3rd and 4th statements to MainForm\_Load() . . .

employeeBindingSource.DataSource = \_viewModel.Employees;

lsbEmployees.DisplayMember = "FirstName";

Rebuild and run the application. Observe – in the left-hand panel – the first names of the 5 employees in our mock database.

## Bind the Detail Controls

In this clip we connect – to the data - the controls that specify an employee’s details.

* Navigate to MainForm.cs (code) - MainFormLoad().
* Add the following statement (below existing statements).

txtFirstName.DataBindings.Add

( "Text", employeeBindingSource, "FirstName", false,

DataSourceUpdateMode.OnPropertyChanged );

This induces a new value to be applied to the txtFirstName.Text whenever this value changes as indicated by the data in employeeBindingSource.

* To bind the DateTimePicker, first we have to remedy a forgotten step – to give this property a meaningful name. Rename it to dtpEntryDate.
* Add another statement below the txtFirstName.DataBindings.Add() statement.

dtpEntryDate.DataBindings.Add ( "Value", employeeBindingSource, "EntryDateTime" );

(You can find the property name from the Solution Explorer by expanding EmployeeManager.ViewModel/EmployeeViewModel.cs; the only property of type DateTime is named “EntryDateTime”.)

* Add another statement below the dtpEntryDate.DataBindings.Add() statement.

chkIsCoffeeDrinker.DataBindings.Add ( "Checked", employeeBindingSource,"IsCoffeeDrinker" );

* At the end of this clip we want to bind a property of the “Save” command button – whether the button is enabled (the **Enabled** property).
* Add another statement below the chkIsCoffeeDrinker.DataBindings.Add() statement.

btnSave.DataBindings.Add ( "Enabled", employeeBindingSource, "CanSave" );

Rebuild and run the application. Try changing the name in the Firstname box. When you change it, the same changes are mirrored in the list box. If you erase all of the letters in the name, the “Save” button becomes disabled. As you select different names in the List Box, the employee details change appropriately.

## Bind the ComboBox

In this clip we bind the Job Role ComboBox to the data.

* Navigate to MainForm.cs (designer), and Click the Job Role ComboBox. We observe – in the **Properties** window – that its name is still ComboBox1.
* Rename ComboBox1 to cboJobRole.
* Navigate to MainForm.cs (code) - MainFormLoad().
* Insert – after the lsbEmployees.DisplayMember statement – the following statements.

cboJobRole.DataSource = \_viewModel.JobRoles;

cboJobRole.DisplayMember = "RoleName";

cboJobRole.ValueMember = "Id";

cboJobRole.DataBindings.Add ( "SelectedValue", employeeBindingSource, "JobRoleId" );

(The 2nd statement indicates that the "RoleName" property is visible in the ComboBox. The 3rd statement indicates that the “Id” property is available as a value. The 3rd and 4th statements define the binding between the combo box and the employeeBindingSource . You can find these property names from the Solution Explorer by expanding EmployeeManager.Common/Model/JobRole.cs and EmployeeManager.ViewModel/EmployeeViewModel.cs.)

Rebuild and run the application. As you select different names in the List Box, the employee’s Job Role changes appropriately. Unfortunately, there is a flaw; the application does not prevent the user from typing into the combo box. As a remedy do the following.

* Navigate to MainForm.cs (designer), and Click the Job Role ComboBox. We observe – in the **Properties** window – that the **DropDownStyle** property is **DropDown**.
* Change the **DropDownStyle** property to **DropDownList**. This prevents the user from typing into the combo box.

Rebuild and run the application. The user is now prevented from typing text into the combo box.

## Implement the Refresh Logic

The purpose of the “Refresh” button is to instruct the application to reload the data.

* Navigate to MainForm.cs (designer).
* As we have seen earlier you can induce an event handler for a button’s **Click** event via the **Properties** window. But a faster way to do this (only for the **Click** event) is to double-click the button in design mode. Double-click btnRefresh. Observe that Visual Studio responds by providing a skeletal btnRefresh\_Click() event handler in MainForm.cs.
* The work to be done in btnRefresh\_Click() is the same as what was done in MainForm\_Load().
* Select all of the code in MainForm\_Load(); then type <ctrl>., accept the **Extract method** invitation, and type “LoadData” to rename the new method.
* Use the clipboard to move LoadData() to the bottom of MainForm class.
* Copy “LoadData();” from MainFormLoad() to btnRefresh\_Click().

Build and run the application. Make a change for one of the employees. Then click the “Refresh” button. Whoops! We get an Argument-Exception error. (It pertains to the cboJobRole.DataBindings statement.) The problem is that cboJobRole.DataBindings.SelectedValue was set originally when MainForm\_Load() was executed; now the application is attempting to set cboJobRole.DataBindings.SelectedValue a second time, because btnRefresh\_Click() was invoked. This same problem would have occurred to the other statements in LoadData() if the DataBindings are already initialized. We should bypass these statements if the DataBindings are already initialized.

* Insert the following code into LoadData() - immediately before the cboJobRole.DataBindings statement.

var areDataBindingsInitialized = cboJobRole.DataBindings.Count > 0;

if (!areDataBindingsInitialized)

* Enclose the subsequent statements in LoadData() with “{“ and “}”.

Build and run the application again. When we changed the text in the Firstname text box, we avoided the Exception, but the name did not revert back to the original value. We can change the code as follows to refresh the data in the employeeBindingSource.

var areDataBindingsInitialized = cboJobRole.DataBindings.Count > 0;

if (areDataBindingsInitialized)

{

employeeBindingSource.ResetBindings ( false );

}

else

{

<subsequent statements in LoadData()>

}

Build and run the application again. The problems have been resolved.

## Save the Data

In this clip we deal with “Save” command button.

* Navigate to MainForm.cs (designer). Double-click btnSave to induce a skeletal handler for the “Save” button’s **Click** event.
* Type the following code into the body of btnSave\_Click().

if ( employeeBindingSource.Current is EmployeeViewModel employeeViewModel && employeeViewModel.CanSave )

{

employeeViewModel.Save();

}

***I, Stephen Cole, simply copied the body of btnSave\_Click from the Pluralsight intructor’s comments. There is almost nothing about the "if" clause that I understand. 07/16/21***

Ultimately employeeViewModel.Save() invokes the SaveEmployee() method in EmployeeManager.DataAccess/EmployeeDataProvider.cs. That method writes text to the console, which shows visible text in the **Output** window.