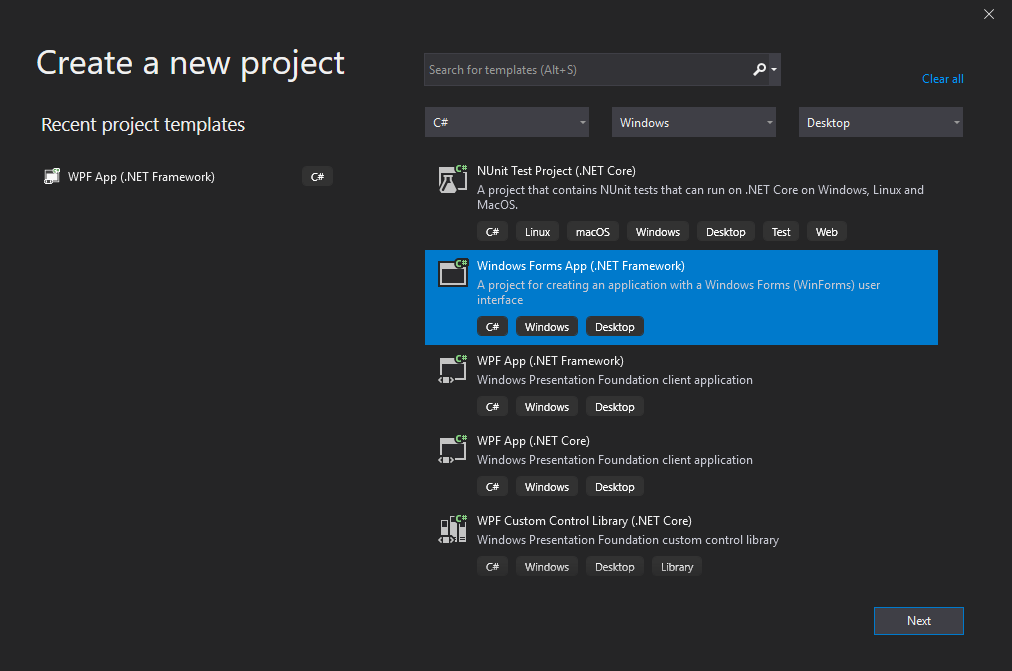
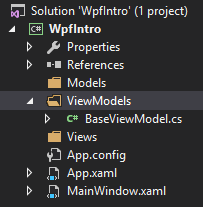
**MVVM How to:**

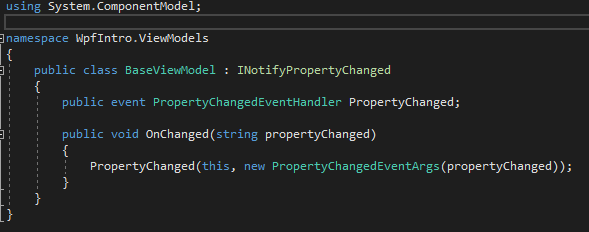
1. Create a new WPF project in Visual Studio 2019. Call it **WpfIntro**

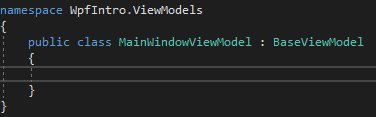
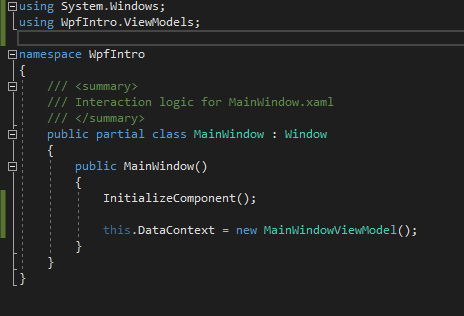
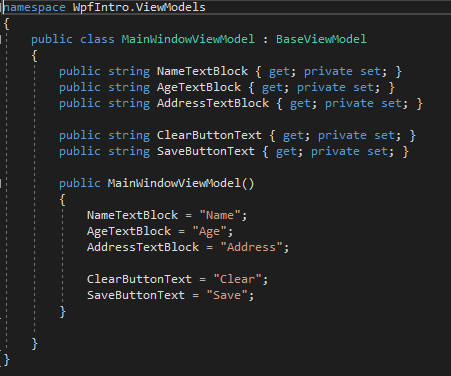
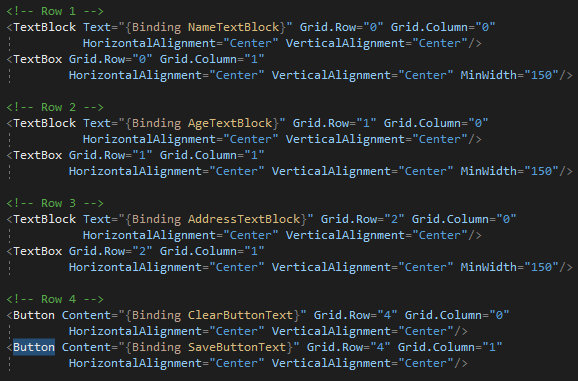
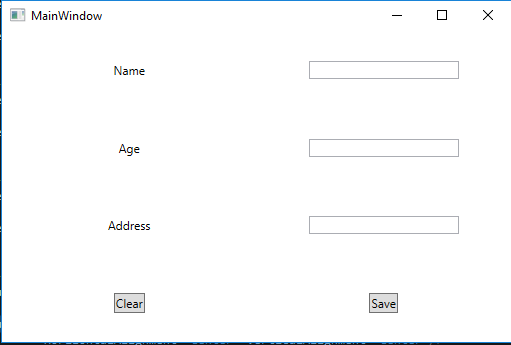
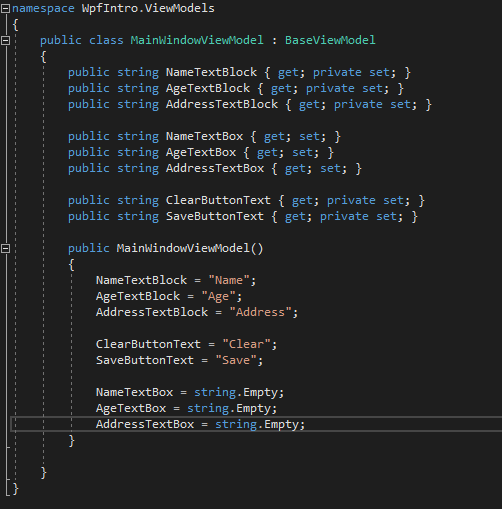
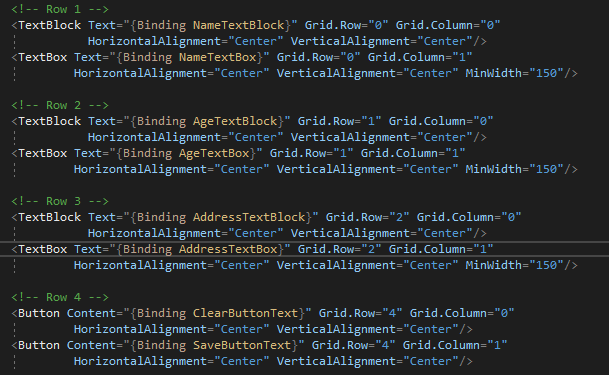
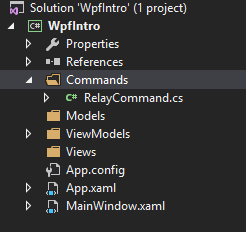
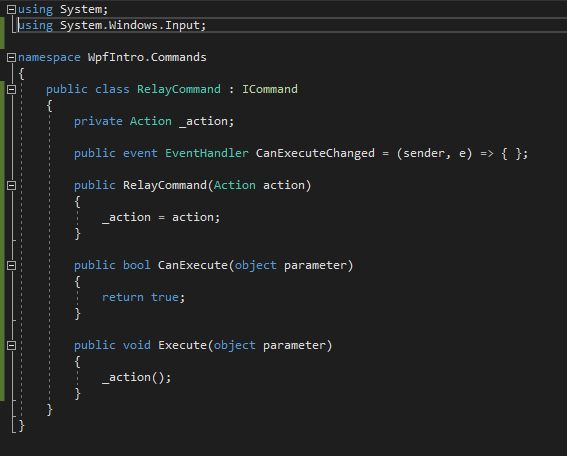
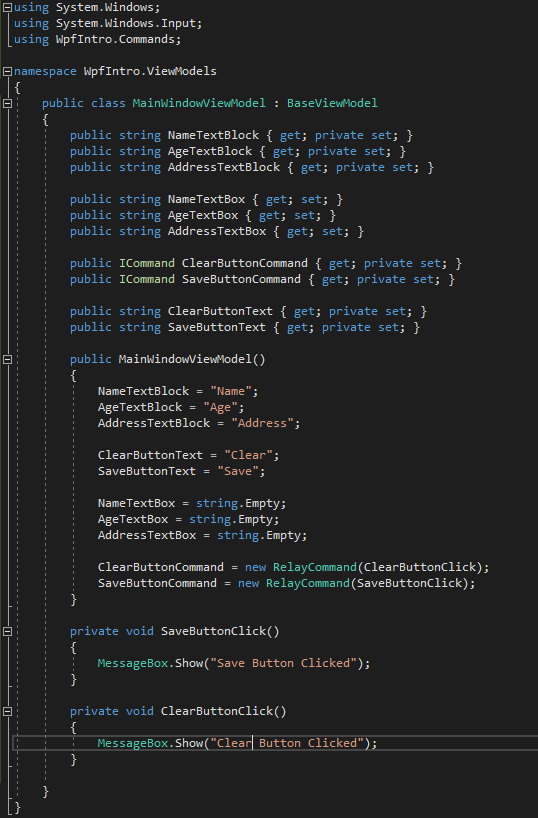
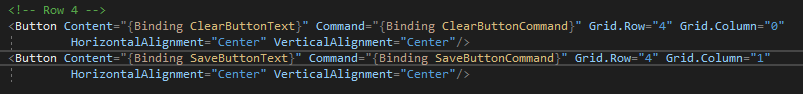
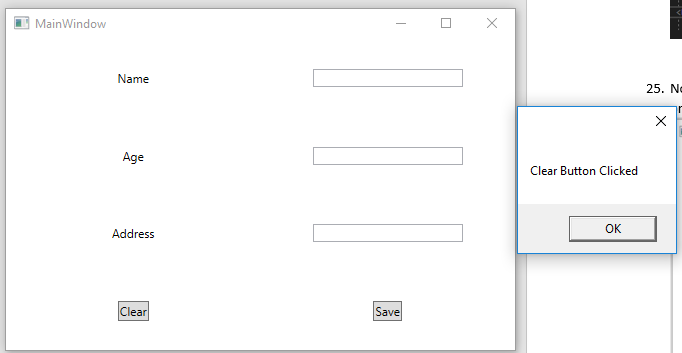
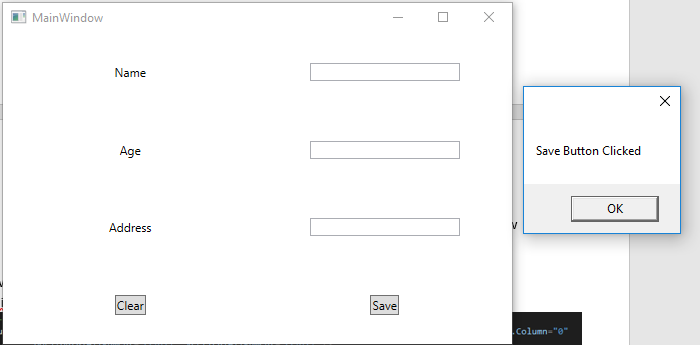
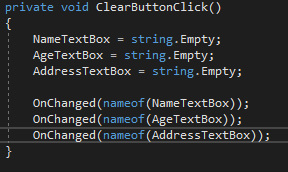
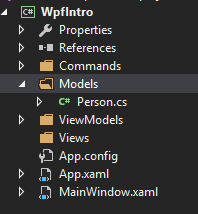
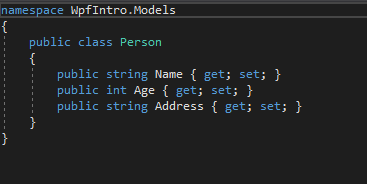
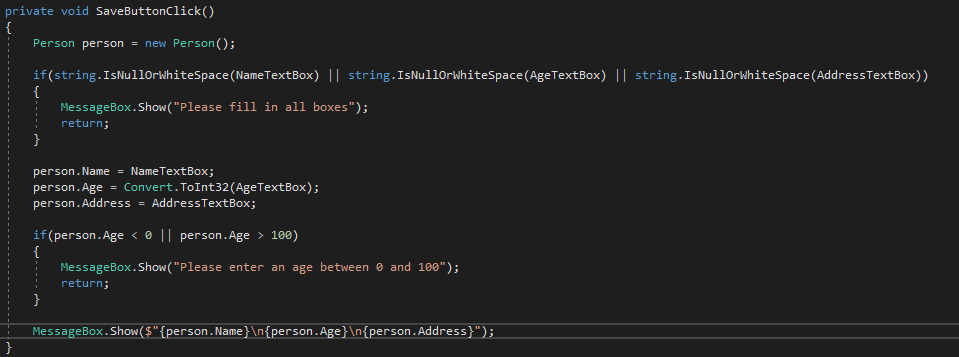


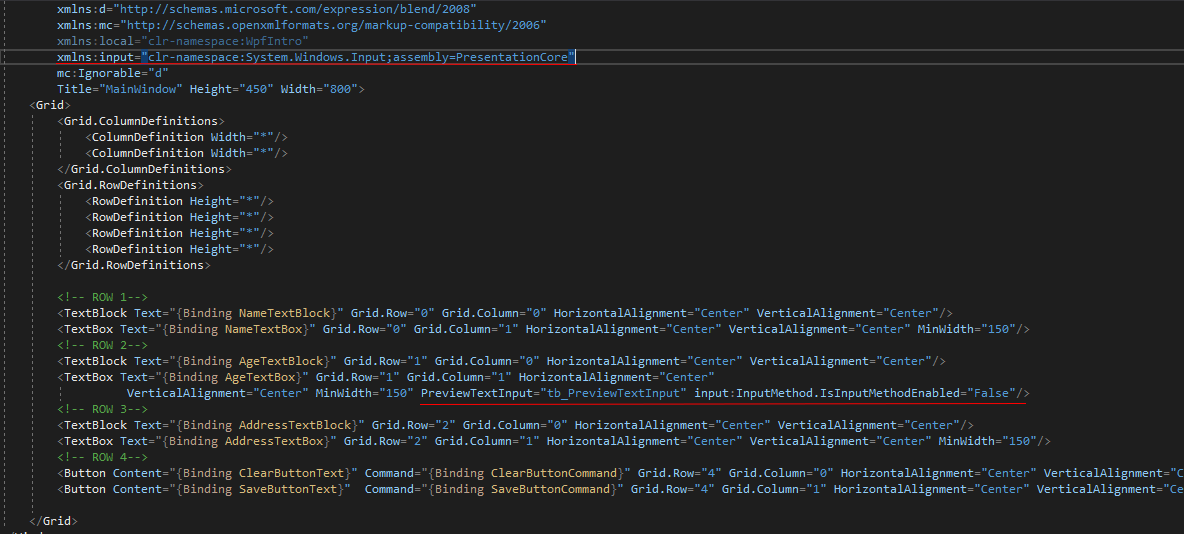
1. Add the following Folders to the project file by right clicking on the name and going to Add
   1. Models
   2. Views
   3. ViewModels



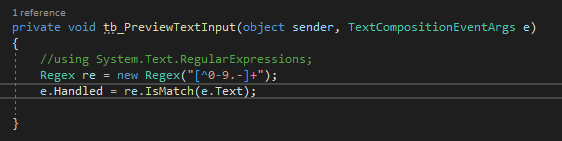
1. In *ViewModels* create a new class called *BaseViewModel*
2. Enter the following code:



1. This BaseViewModel class acts as a base class for all your future viewmodels which will all inherit from this. The INotifyPropertyChanged and the OnChanged method sends a signal to the UI whenever a property of the viewmodel has changed and updates the UI with the new value
2. Next Create a class called *MainWindowViewModel* in the *ViewModel* Folder
   1. This Class is what is going to hold all the C# code that the *MainWindow*.*xaml* will need to run
   2. Inherit the *BaseViewModel* class we just created
3. We are now ready to start adding controls to the UI on the *MainWindow.xaml*
4. Open up the *MainWindow.xaml* file and add the following to it
5. This sets up a basic Grid with 2 columns and 4 rows.
   1. The \* sets the size of the column/row to equal sizes which will scale up or down depending on window size
   2. Notice the Content and Text values are currently set to TEXT this is just a place holder for the time being
6. Right click the MainWindow.xaml file and select View Code. Usually all of the code that would run buttons, textboxes etc would be placed here. But all we are going to do is add one line of code. This line will tell Visual Studio that it is to get all of the values it needs for the XAML file from the ViewModel class we just created. 
7. Now open up the *MainWindowViewModel* class
8. We are going to add values to this class which will then be displayed by the UI
9. These values are going to replace the TEXT place holders in the XAML file
10. Back to the *MainWindow.xaml* file and replace the TextBlock Text and the Button Content values with the following code
    1. Inside the quotation marks for Text and Content has been changed to {Binding (TextBlockProperty)}
    2. This tells visual studio when running the application to go to the ViewModel and get the relevant property and display the values.
    3. These won’t be loaded until run time.
    4. Make sure they are the exact same name as the properties we just created in the *MainWindowViewModel* class
    5. Run your program and you should see the values being displayed
11. This seems like a lot of work to do when you could just add the text to the UI and leave it.
    1. Why things are done in this way is so that if the UI were to be changed from a WPF to a WinForm or to the new UWP framework all that needs to happen is the ViewModel code is copied over to the new project and the new project Binds to it again. This saves whole sections of UI having to be rewritten when the values already exist
    2. This also makes the UI side of the project to be very lightweight as all it will be doing is sending or receiving info from the ViewModel and not having to process complex tasks as well as display them all in the one place
12. On to the TextBoxes now!
13. Open up the *MainWindowViewModel* class and add the following code to it
    1. The NameTextBox, AgeTextBox, AddressTextBox values have all been created and set to an initial value of empty string.
    2. These are going to be Binding to the TextBoxes on the XAML file.
14. Back to the MainWindow.xaml file and in the TextBox Text value place the {Binding (TextBoxProperty)} 
15. Up to now all we have done is bind some properties to the UI. Now we need to be able to read and manipulate these properties in the ViewModel Class.
    1. This is done by Binding a Command to the relevant buttons
    2. Commands on the ViewModel class will activate when the Button that has been Bound to it on the UI side has been pressed
16. On the Project right click and Add a new folder called Commands and add a class called RelayCommand to this folder
    1. 
17. A RelayCommand lets the ViewModel class know that a button has been clicked or that an event has happened
18. Replace the code in the class to the following
    1. *RelayCommand* inherits from the *ICommand* interface
    2. The line: public event EventHandler CanExecuteChanged = (sender, e) => {};
       1. This is called an inline function which you can read about further but is not really covered in this tutorial.
    3. The CanExecute() function will always return true. This is to allow the Command to run. If set to false it will never run.
    4. The Execute() method tells our private field to actually run the code which we will inject into the RelayCommand via the Constructor
19. Back to the *MainWindowViewModel* class now and add the following lines of code
    1. We’ve added the new properties called *SaveButtonCommand* and *ClearButtonCommand* of type *ICommand*
    2. The Methods *SaveButtonClick* and *ClearButtonClick* are what will fire when the button on the UI is clicked.
    3. The *SaveButtonCommand* and the *ClearButtonCommand* have been set to an new instance of *RelayCommand* and the relevant Method is the code which we are *wanting* the *RelayCommand* to run for us
20. Now to Bind these Commands to the relevant Buttons on the UI. Back to the MainWindow.xaml file and add the following code to the Buttons
    1. The Command attribute has been added and again as with the TextBlocks and the TextBoxes this has been set to Command=”{Binding (CommandProperty)}”
21. Now if you compile and run the program and press one of the buttons a MessageBox should prompt you on screen to say that it has been pressed
22. This is all the work on the UI finished now.
    1. If you want to change colours or fonts or anything all you have to do is alter the attributes on the xaml as long as you leave the bindings the way they are.
23. Back to the *MainWindowViewModel* class and in the *ClearButtonClick* method enter the following code
    1. 
    2. Notice how we are again setting the *TextBox* properties to an empty string. Then we are calling the *OnChanged* method, which we created in the *BaseViewModel* class and letting it know that there has been a change in a property
    3. If you run the program now and type values into the *TextBoxes* and press Clear you will see that they now clear themselves
24. Now add a class to the *Models* folder and call it *Person* 
    1. 
25. Add the following code the the class
    1. 
26. This is now going to act like an Object that you would’ve came across in Object Oriented Development previously. Except no methods are going to be in this Object only Properties
27. Back to the *MainWindowViewModel* class
28. In the SaveButtonClick method replace the code with the following
    1. You will need to make sure the following using statements have been included:
       1. Using System, using WpfIntro.Model
    2. The first IF Statement checks to make sure that all the boxes have been filled in
       1. If they have not a message prompts the user to fill them all in and the code will no execute further
    3. Next a new Person is created and the values of the TextBox is added to the Persons values
       1. Of course, The AgeTextBox value is an int type. You will need to do this yourself or you can also follow the steps below to disable the input method in the AgeTextBox, and then use regular expressions to pre-detect whether the input in the input box is characters or numbers.

MainWindow.xaml

MainWindow.xaml



* 1. The next IF statement checks the age range is correct
     1. If it is wrong again it prompts the user to enter a correct age and will execute no further
  2. The final MessageBox just shows the user what has just been entered. The $”{value}\n{value}\n{value}” is a way to add values to a string without constantly appending it which takes up more code

This tutorial shows how to implement the Practical Intro to Wpf using the MVVM design pattern

It seems like more complex than just hard coding the values right into the UI but now your project is more modular. This is very useful for large projects and UI elements can change but the actual values of them will remain.

You can expand this project to save the Person to either a CSV or Json file.

We haven’t added anything to the Views folder as it was not needed in this case.