This is a work in progress and is likely to contain bugs. The helpers don’t act very friendly and will throw errors when not addressed properly. If you choose to use it for something, feel free to change any or all of it. I won’t be offended. You saw something in it you liked to get you started, and that was the point.

1. **Compatibility**
   1. The base project targets .Net 4.0 Client Profile making it backward compatible to Windows 7 with .Net 4.0 Client Profile or higher installed. It has not been build tested on Windows Vista or earlier.
2. **Summary**
   1. I began exploring WPF as a replacement for Windows form applications (winforms) due to poor performance in data grids. While WPF has its drawback, I believe the overall potential exceeds that of winforms. Below I discuss some of the major pros and cons of the framework and some workarounds I’ve provided.
3. **Video Acceleration**
   1. After all of these years, Windows still treats winforms like every application will be written for systems that were designed in the 80’s. Little things like limits to the number of controls on a form and unpreventable intermittent flickering when attaching a data source to a grid which gets even worse if you try to do something like, say, format a row. Something it was built to do. Most of the problems stem from the fact that Microsoft just won’t add functionality that allows winforms to take advantage of graphics acceleration. Well, WPF does.
4. **Data binding and validation**
   1. Every developer is looking for ways to get things done faster, but without losing control over their favorite properties. Winforms offers some data binding functionality, and when it’s used, it works well. But WPF takes this a little further by inherently allowing binding. But not just to the value, you can actually bind to any property on the control. This is handy for people who like to manage all of their control settings from user settings. Another bonus is the ability to generate a durable namespace for your validation. As you encounter a new need for validation, you can add a class for it (usually to a single document). Then you move these classes forward to your next project and capitalize on past work. Then, thanks to a little help I’ve written, you can set up your data binding and validation rule in the same call. A form that used to take 60 lines of code might now take 10.
5. **Layout markup**
   1. Wpf was designed to be both a web and a desktop platform. The xaml schema of xml is declarative and accepted in most browsers. In theory, you can do just about anything you would do in the cs file in the markup. This allows for a lot of potential, but also a lot of headache as now you have an entirely new place to debug. Also, when you use it for the web, the cs files are no longer used for processing at all, so
   2. you are forced to do everything in the xaml markup. In light of these facts, I am of the opinion that unless specifically tasked to do so, and against my objections, I was to do a project in wpf for the web; I would become accustom to using the declarative markup. For now, I treat it as layout. That is its job. It does not also compute.
   3. Now there is a side note to that. Wpf is designed to foster animation. These functions can be considered part of the layout. Therefore, I believe that is where they belong and to this end, I have used them some. Keep in mind, if you want your animations to be reusable, your still going to want them to be in the cs files.
   4. One might argue also that binding could be considered part of the layout, and that is fine also, though I do think the helper would be more effective if not only because you can group all of you binding calls in one section of a document.
6. **Helpers**
   1. How I began this project was actually by writing a functioning project. While writing it, I was almost immediately struck by how similar it was to using winforms, while being frustratingly different at the same time. It seemed to me that for just about everything you wanted to do it took 4-6 lines of code. And in many cases, I was doing these things over and over. Helper classes are born for this and as soon as you have helpers, you have the notions of how they will help in other projects. And here we are.
7. **Validation**
   1. The validation helper namespace contains the validations rules namespace, which contains all of the validation rules I have established so far. These will allow you to assign a validation rule in your binding call by enumeration. It also contains a simple page for displaying validation errors.
8. **View Helpers**
   1. This contains the ListElementHelper class which has a work around in it for the annoying fact that the current version does not allow for the handling of items in a list box by their text. By text it allows you to set the selected item, get the selected item, move to the previous or next item. It also contains two functions loosely related to using listboxes, but that apply to other controls as well so should probably be moved. One allows for a child to be located and returned from within its parent. The other returns the parent of the target.
9. **Animation Helper**
   1. This does not contain as much as one might think. I found that when I began coding animations, it overcomplicated the UI and made it gaudy. I like a smooth efficient interface with limited animations. I also found that the code tended to be pretty specific to what was being animated, though there is obviously some that I could have done. What really stopped me was that I think this code is actually more efficient and perhaps better located in the xaml, though I have not yet included any examples of that either.
   2. It does contain an example of how to create an animation, which is one I do use frequently, that will fade in or out any Framework element. I also provided a work around for a problem where the framework would not always evaluate what the default state of a control was if a control had an animation applied. An example would be fading in a control. Assume you have a control that is not visible, and then apply an animation to fade it in and then out. If later, you would like to then use the visible property, you would not be allowed because the control is being manipulated by an animation. The DefaultOpacityAnimation static property will allow you to reset the current animation easily.
10. **Database**
    1. This static class is meant to make coding database requests easier. It succeeds in many ways, but fails in some others. I have found this approach to not be conducive to executing commands on the same connection to be lacking and have provided better alternatives in working projects without bringing the changes back here. Feel free to use it but expect a better class in the near future.
    2. Several examples of how to use the syntax is available along with the other examples on MainWindow.xaml.cs. Some of the features are, automatic or easy closing and disposal of process object (DataReader and Connection), error notification, and easy data bridge construction with a single method in the model.
    3. Known Bug: If you attempt to use any construction that would allow for the connection event handlers to be attached, then execute a reader on the command using a while loop as in While(Reader.Read()) an error is thrown because the underlying connection is closed before this check is made, and the method no longer exists. A workaround is to instead evaluate While (command.Connection.State == ConnectionState.Open). This will act as expected as the connection was closed immediately after the last read. This may not properly address all concerns though as it could mean it’s usage in the data model is now limited in some ways.
11. **Main Window Helper**
    1. This is where most of the work that this framework does passes through. If the Main Window is used, this Helper allows most of the error reporting and user notifications to take place. If you would like to add to it, feel free but be very careful in what you alter as it could have wide spread implications.
    2. An example would be the tabbed notification area in the bottom left. While I would encourage you to figure out how it works, it is one of the more finicky things I have in this code. The timers can make your head hurt and mistakes can lead to unpredictable behavior and even crashes. I would not tamper with the timers until you fully understand what they do and the implication of your change. Not using this system, or using it as a guide to build one of your own are always options. It was introduced to fill a suspected role, show how controls can be stacked, and put some basic animations into play. It has some serious drawbacks (fixed position, real usefulness) and should be strongly evaluated before it is used.
12. **Modal View**
    1. WPF does a pretty good job of indicating the modal state, but I felt a little more was in order. For instance, a dialog always has the same border and control style. In my opinion, this should be more flexible. This code demonstrates how to make a new Window, how to control many of its properties, and how to add content. It also shows how to use one of the main windows features, which is a modal mask. This can be used in conjunction with the effect, blur, to make a pretty cool modal state. The function used to remove the icon can also be tracked back to show the only extension method I have in that helper.
13. **Window.Extentions**
    1. This only contains one method used to remove the icon from a window. Feel free to add your own here.
14. **Models**
    1. This is mostly here for application structure. Its contents are not intended to be permanent. However, it does contain one working data model as an example of usage and data bridge tie-in.
15. **Pages**
    1. Again, this is mostly here for application structure. Its contents are not intended to be permanent. The current contents are all examples. These examples can be toggled by toggling comments within the example region. Each page can be looked at as a separate form. Unlike winforms, WPF inherently likes relational structure. When designing your page, keep in mind it will always scale to fit the main window. You can take advantage of it, or place your layout in a grid that is flexible on the four sides with a fixed layout in the center, and your layout will be preserved and centered. Also keep in mind that only one page can occupy the main window at a time, but modals stack. However, there is nothing to stop you from creating a page with sectioned layout that contains other pages. In fact, I have a method in MainWindow.Helpers for this, but have to admit I have not tested it. If it fails it’s pretty darned easy and should work on the object directly.
16. **Resources**
    1. This folder is created by default with and WFP project and contains one file that I don’t believe is currently in use.
17. **Themes**
    1. This folder contains 7 themes that are freely available from the internet. Only one can be used at a time and syntax for toggling between them is available in App.xaml. It is assumed that one would be selected that is closest to the intended design, and then altered, while the rest are disposed of. Only one or zero themes need to be present. Make sure to adjust the App.xaml accordingly. If no theme is used comment or delete all theme references.
18. **App /Main Window**
    1. This is the starting point of the app. It contains several examples on syntax for using the helpers. It uses pages, modals, or features of the main window to demonstrate how common tasks can be executed using a combination of WPF libraries and the helpers. To get the most from the examples, you should get a feel for accessing the helpers and executing the example, but then go to the page also and see how it is being done. In many cases the real examples of efficient code and easy execution is there. However some, like the database class, the syntax is (to me) the highlight as getting data is pretty common place, but doing it this easily is usually less frequent.