[WPF Questions 2](#_Toc362556469)

[WPF Basics 2](#_Toc362556470)

[Dependency Concepts 4](#_Toc362556471)

[Layout/Controls Concepts 12](#_Toc362556472)

[Style/Template/Binding Concepts 19](#_Toc362556473)

[Converter Concepts 31](#_Toc362556474)

[Dispatcher Concepts 33](#_Toc362556475)

[Commands & Events Concepts 37](#_Toc362556476)

[Resources 41](#_Toc362556477)

[Smart Interview Questions 43](#_Toc362556478)

[Design Pattern Concepts 43](#_Toc362556479)

[WPF Architecture Questions/Localization 47](#_Toc362556480)

[Additional Concepts 51](#_Toc362556481)

[Testing Concepts 52](#_Toc362556482)

[Generics 53](#_Toc362556483)

[String Concepts 54](#_Toc362556484)

[Exception Handling 55](#_Toc362556485)

[Misc 56](#_Toc362556486)

[Threading Questions 56](#_Toc362556487)

[Basics 56](#_Toc362556488)

[Multi-Threading Techniques 57](#_Toc362556489)

[Synchornization Techniques 57](#_Toc362556490)

[Threading Issues 58](#_Toc362556491)

[Other Questions 59](#_Toc362556492)

[Data Structure Questions 59](#_Toc362556493)

[Performance Profiling Links 62](#_Toc362556494)

[Fakes and Mocks 63](#_Toc362556495)

WPF Questions

|  |  |
| --- | --- |
| WPF Basics | |
|  | What is the difference between Logical Tree and Visual Tree   * The logical tree exposed by WPF is a simplification of what is actually going on when the elements are rendered. * The entire tree of elements actually being rendered is called the visual tree. You can think of the visual tree as an expansion of a logical tree, in which nodes are broken down into their core visual components. For example, although a ListBox is logically a single control, its default visual representation is composed of more primitive WPF elements: a Border, two ScrollBars, and more |
|  | What is markup extension?   * Markup extensions enables a user to extend the expressiveness of XAML. * It can eval a string attribute value at runtime and produce an appropriate object based on string. * Few built-in markup extensions that are currently evaluated at compile time for perf reasons * They are defined by {} |
|  | What is the most common format for declaring a data binding in markup?  Markup extensions are the most common means of declaring a data binding. They are identified by the use of curly brackets ({}).The less common format is XAML. |
|  | What are the common markup extensions in WPF?   |  |  | | --- | --- | | **Common Markup Extensions in WPF** | | | Binding | The extension used for binding data. | | x:Null | Used for specifying a null value in XAML. | | x:Type | This extension is used for supplying a System.Type object. | | X:Array | This allows you to define an array of objects in XAML. | | StaticResource | * StaticResources are resolved at compile time. * This is used when the resource is not needed to be re-evaluated. | | DynamicResource | Similar to StaticResource, except that the data in the resource might change during runtime. | | RelativeSource | Allows you to bind source object on its relation to the target object which can be self, parent or template parent. | | Difference between StaticResource & Dynamic Resource   |  |  | | --- | --- | | Difference between them lies in how the resources are retrieved by the referencing elements. | | | StaticResource are retrieved only once by the referencing element and used for entire life of the resource. | DynamicResource are acquired every time the referenced object is used. | | Has lesser overhead as it is applied only once during the runtime of the application. | Has more overhead as it is applied whenever the resource value changes. | | Always get loaded the first time window/page is loaded irrespective of use during start up. | Are not loaded until it is actually used. So this results in better load time. | | Has to be defined before it is used. Hence, they can not be forward referenced. | They can use the forward references. | |  | Disadvantage - (Use-case where u cant use DR for sure) – It can only be set on a dependency property derived from a dependency object. | | | |
|  | Can you use Static Resource twice within the same panel. How do you address this?  X:shared=”false” will let you use the same StaticResource twice in the same panel |
|  | Is x:Static the same as a data binding?  x:Static is another markup extension that is different from the data binding markup extension. It is used to retrieve data from static value members on classes. |
|  | What is this syntax called    Property Element Syntax – an alternative syntax used for providing values for complex types |
|  | Can you elaborate the core classes of WPF |
|  | **What is System.Windows.UIElement?** UIElement class brings life. Here L mean “Layout behavior (measure/arrange)”, I mean “Input”, F mean “Focus” and E mean “Event”. In other words every class inherited from UIElements has support for layout, input, focus and event handling. |
|  | **What is System.Windows.FrameworkElement?**  The base class that adds support for:   * Additional input elements (e.g. tooltips, context menus) * Data binding * Styles * Storyboards * Property value inheritance * Support for the logical tree |
|  | What is Freezable?  The base class for objects that can be “frozen” into a read-only state for performance reasons. Freezables, once frozen, can be safely shared among multiple threads, unlike all other DispatcherObjects. Frozen objects can never be unfrozen, but you can clone them to create unfrozen copies. Most Freezables are graphics primitives such as brushes, pens, and geometries or animation classes. |
|  | What is the purpose of the x:Name attribute in XAML?  The x:Name attribute allows you to uniquely identify an instance of an object defined in XAML. The value of the x:Name attribute can be referenced in the associated C# or VB code |
|  | Difference between x:Key and x:Name  These are both the XAML attributes that are specified in the declaration of XAML elements. x is the namespace prefix     * Uniquely identifies elements that are created and referenced in a XAML-defined dictionary. * Using the Key element, we can refer the StyleSheet defined in <Style></Style> tag in some other <Style></Style> tag or in the declaration of some user interface element. * x:Key is not applicable for user interface elements like the Button      * It is used to mention the names of StyleSheets as well as the user interface controls * x:Name can also be accessed in the code-behind (.xaml.cs /.xaml.vb), but x:Key cannot be accessed in code behind   Additional Information  **For x:Key** : If you want to reference the same instance being used in XAML– then use FindResource in code behind to get the handle on that  **x:Name** when used – becomes a private field in the generated class (MainWindow.g.i.cs) and hence the element is then available in the code behind. |
|  | Describe 'lookless' controls  A Lookless control is a control that defines the behavior of a control without regard to how it will look when rendered. For example, a button is a control that has a normal state, a mouseover state and a clicked state. The appearance of the button could be anything—from the familiar grey box to an obnoxiously spinning 3D flaming corporate logo circa web design 1997 style.  When Microsoft set about building the Windows Presentation Foundation, one core goal was to create an environment for applications capable of both two-dimensional and three-dimensional content simultaneously. Using a traditional approach, this would have led to an abstract Button class followed by Button2D and Button3D subclasses that would override the painting rules. Tearing a page from the CSS-based Internet, Microsoft also wanted the look and feel of all controls to be controllable via styles that, when updated, would in turn update the look and feel of every component of that type in the application. |
|  | What are the phases in WPF Layout  Layout occurs in 2 phases:  **Measure Phase:**  Walk UI tree, determine preferred sizes: In the measure phase, WPF walks the entire visual tree and finds out how large each element needs to be. It does this by calling method on each element called measure. Constrained or Unconstrained: Constrained layout occurs when WPF knows how much space will be available or can make a good guess. Unconstrained layout occurs when the amount of space is unknown or unbounded. Occurs for both horizontally and vertically. Size to content: Elements are not allowed to ask an infinite amount of space during the measure phase so they must return a finite preferred size regardless of what is passed in.  **Arrange Phase**: Once the measure phase is complete WPF knows how big each element wants to be. In the second phase of layout WPF walks the tree a second time and calls the Range on each element passing in size and location. Ofcourse this may be impossible – element may ask for more space than available. In this case WPF will compromise and truncate the element. |
|  | Default Namepsaces |

|  |  |
| --- | --- |
| Dependency Concepts | |
|  | What is a dependency property? What does it provide more than CLR properties? OR What are the 3 features it provides over CLR properties  Dependency property is new property mechanism introduced by WPF. The purpose of dependency properties is to provide a way to compute the value of a property based on the value of other inputs.  These other inputs might include system properties such as themes and user preference, just-in-time property determination mechanisms such as data binding and animations/storyboards, multiple-use templates such as resources and styles, or values known through parent-child relationships with other elements in the element tree.  DP offer few additional features:   1. Change Notification 2. Property Value Inheritance 3. Support for multiple providers |
|  | What is Change Notification?  Whenever the value of the dependency property changes, WPF can automatically trigger number of actions depending upon the property’s metadata. One of the most important feature enabled by this built in change notification is property triggers, which enables to perform custom action when property value changes without writing any procedural code. |
|  | What is property value inheritance  This refers to flowing of property values down the element tree.  e.g. if FontSize(which is dependency property) value is set on to the window class, the same FontSize value will also be applied to all the children of the Window for which the FontSize value is not explicitly set. If for particular children, FontSize is set explicitly the parent settings will not be applied. |
|  | In which scenarios does property value inheritance doesnt seem to work?   * Not every dependency property participates in property value inheritance. (Internally, dependency properties can opt in to inheritance by passing FrameworkPropertyMetadataOptions.Inherits to DependencyProperty.Register.) * There may be other higher-priority sources setting the property value (eg: Status Bar) |
|  | From training  Window.Background is a dependency property but doesn’t have the inheritable option (metadata value is set to false). Hence setting it at Window doesn’t set the background of all the controls below. Whereas FontSize/Weight etc does have the inheritable option and hence all controls pick the value. |
|  | Support for Multiple Providers?  Multiple sources try to set the value of the dependency property however there is a defined mechanism based on which the final value is arrived at for dependency property. It is basically a five step process after going through which the value of the dependency property is finalized.   1. Determine the Base Value (Diagram Below) 2. Evaluate 3. Apply Animation 4. Coerce 5. Validate   More detailed explanation about the points if needed - <http://csharpsimplified.wordpress.com/2008/12/26/dependency-property/> |
|  | What is Dependency Object?  From the consumer point of view, DependencyObject is little more than a glorified Hashtable. It’s job is to store the current values of any DependencyProperty that is ever applied to it.   * DependencyObject contains 2 important methods called GetValue & SetValue which are used to retrieve values from the internal hashtable and used by Dependancy property system. * So the value you are giving goes & sits inside the DependencyObject’s hashtable. Important thing to note here is in many cases the value may not be explicitly set by you but the value may be derived from surroundings. * In order to have a dependency property – it needs to be derived from the class DependencyObject. In deriving from this class, you get all the infrastructure needed to participate in the WPF dependency property system. * It also offers Attached property hosting support. |
|  | What are the methods in DependencyObject in WPF?  GetValue  SetValue  OnPropertyChanged - Invoked whenever the effective value of any dependency property on this   DependencyObject has been updated. The specific dependency property that   changed is reported in the event data. |
|  | What is the value resolution strategy in WPF   |  |  | | --- | --- | | Every time you access a dependency property, it internally resolves the value by following the precedence from high to low. It checks if a local value is available, if not if a custom style trigger is active,... and continues until it founds a value. At last the default value is always available. |  | |
|  | More Details on Value Resolution  5. TemplatedParent template properties. An element has a TemplatedParent if it was created as part of a template (a ControlTemplate or DataTemplate). Within the template, the following precedence applies:  Triggers from the TemplatedParent template.  Property sets (typically through XAML attributes) in the TemplatedParent template.  10. Inheritance. A few dependency properties inherit their values from parent element to child elements, such that they need not be set specifically on each element throughout an application  <http://msdn.microsoft.com/en-us/library/ms743230.aspx>  <http://joshsmithonwpf.wordpress.com/2007/05/16/demystifying-dependency-properties/> |
|  | How did you create Dependency Property OR Can you walk through a Dependency property implementation   |  |  |  | | --- | --- | --- | | 1 | Declare the dependency property (static readonly) |  | | 2 | Register the property |  | | 3 | A. Net property wrapper (optional) |  | | 4 | A property changed callback (optional) |  | |
|  | Are dependency properties static?  Yes – all DependancyProperty fields are public, static, readonly and have a Property suffix  Public so that all property system calls including cross-assembly can access it.  The field you declare as static is only the identifier of a dependency property, not the value of the property. It is shared across all instances of the class, and is used to get/set the value of the property for each instance. So in essence - When you declare a DependencyProperty, you are declaring the definition of that property, rather than the storage space for the property's value (as would be the case with the regular property declaration).  Readonly - Because it makes it obvious, that the value of this property cannot be changed after initialization. |
|  | What information do you need to define a dependency property?   * Property Name * Type * ControlType that it applies to * MetaData (which can handle the change events and set a default value) |
|  | From training |
|  | Custom DependencyProperty Explanation   * The Window class derives from DependencyObject, so Window1 is already set up to handle dependency properties. * By convention, when you declare the dependency property field, the field name should end with the string Property. In the code, the dependency property name is SidesProperty. The property must be declared public static readonly. * The CLR wrapper should have the same name as the dependency property field, but without the Property suffix. * Notice that the CLR wrapper does not get or set the value of the dependency property field. Instead, it calls the GetValue and SetValue methods of the DependencyObject base class to have the WPF property system get and set the values. * Internally, at runtime, WPF bypasses the CLR wrapper and calls the methods directly. So, to make sure that the wrapper and the runtime calls remain equivalent, your wrapper accessors shouldn’t include any additional code other than the calls to the GetValue and SetValue methods. * I’ve now declared the dependency property field but haven’t yet assigned an instance to it. Since the dependency property field is a static field, it must be assigned to in a static constructor of the Windows1 class. That would be straightforward enough, except that there is no public constructor for the DependencyProperty class. Instead, to get an instance of the class, you must call the class’s Register method. Registering a dependency property connects it to the WPF property system and tells the property system how to manage its value. |
|  | What are read-only dependency properties?  Some dependency property of WPF controls are readonly. When registering your property, call the RegisterReadOnly method instead of the normal Register method for property registration. This makes DependancyProperty as readonly.  Example - . They are often used to report the state of a control, like the IsMouseOver property. Is does not make sense to provide a setter for this value. |
|  | What kind of properties in the UI can be updated by INotifyPropertyChanged?  DependencyProperties |
|  | How to find from where is a control’s dependency property value being set?  By using tools such as Snoop, WPF inspector.  By using the GetValueSource method. |
|  | **What is an attached property?**  **- An attached property is a special form of dependency property where the property is declared in   one class, but is used on the object of a different class.**  **- Attached properties are simply a way of connecting additional information to an element.**  **- The most common example of attached properties is found in the layout containers.  <Button x:Name=”myButton” DockPanel.Dock=”Top”/>** |
|  | Internals of how attached property works   * Instead of a CLR property wrapper, the DockPanel class has two methods called GetDock and SetDock, which, like the CLR wrapper on a regular dependency property, call GetValue and SetValue. * The major difference, though, is that the GetDock and SetDock methods call the GetValue and SetValue methods—not of their own object—but of the object using the attached property. * This effectively attaches a DockProperty dependency property to each button. Later when the DockPanel is cycling through its list of children, it can call the GetValue method on each one to retrieve the value associated with the element. |
|  | Example of attached behavior?   * Grid class defines the attached properties Row and Column * DockPanel defines the attached property Dock. * Canvas defines the attached properties Left, Right, Top, and Bottom |
|  | Difference between Attached Property and Dependency Property  The object on attached property is used is called the target object.   * Regular dependency properties have a CLR property wrapper around their calls to GetValue and SetValue. * Attached properties have two static methods of the form GetXXX and SetXXX, where XXX is the name of the dependency property field, without the Property suffix. * These methods call the GetValue and SetValue methods of the target object. * The target object must be derived from DependencyObject, in order for the * GetXXX and SetXXX methods to call its GetValue and SetValue methods. Besides these differences, attached properties are registered with the static RegisterAttached method, rather than the Register method. |
|  | Custom Attached properties |
|  | Is it possible for a control to define a handler for an event, that the control can't itself raise?  Yes, It's is possible for a control to define a handler for an event, that the control can't itself raise. These incidents are called attached events.  For example, a button control in a grid. The button class defines a click event, but the grid class does not. However, you can still define a handler for buttons in the grid by attaching the Click events of the button control. |
|  | In WPF via XAML through to my viewModel, I want to detect mouse move on a Canvas, and react to it only when the left mouse button is pressed. Specifically I want to get the event from a method in my viewModel, including mouse position and mouse button states. Can someone help me with this?  You can create an attached behavior of canvas that accepts the command and raise it and your view model will subscribe to that command and perform the action. |
|  | IPropertyChanged and INotifyCollectionChanged  WPF Binding generally needs to be notified when it is modified. The interfaces INotifyPropertyChanged and INotifyCollectionChanged are needed to update the UIElement which is bound with the data. So if you are creating a property which needed to update the UI when the value of it is modified, the minimum requirement is to implement the same from INotifyPropertyChanged, and for collection (like ItemsSource), it needs to implementINotifyCollectionChanged. ObservableCollection itself implements INotifyCollectionChanged, so it has support to update the control whenever new item is inserted to the list or any old item is removed from the string. |
|  | What is INotifyPropertyChanged used for?  For notifying any observer, usually a WPF control, that the property it is bound to has changed. |
|  | What is the importance of INotifyCollectionChanged in WPF   * This interface is similar to INotifyPropertyChanged, which is to be implemented to notify the WPF/Silverlight framework whenever any changes occur in a collection, like adding an object or deleting an object. * "ObservableCollection" is a built-in class which implements the INotifyCollectionChanged interface. |
|  | Explain how objects can notify the UI of changes.  There are three methods:   * Implement INotifyPropertyChanged and raise an event every time the property changes * Implement the property as a DependencyProperty (which automatically handle change notification) * Create an event named after the property named MyPropertyChanged. WPF automatically searches for this event if an object does not impliment INotifyPropertyChanged or DependencyProperty |
|  | POCO (Plain old CLR objects) versus Dependency Objects   * A DependencyObject has thread affinity – it can only be accessed on the thread on which it was created * The DependencyObject class overrides and seals the Equals() and GetHashCode() methods * DependencyObjects are not marked as serializable   Reference Material - <http://kentb.blogspot.com/2009/03/view-models-pocos-versus.html> |
|  | InotifyPropertyChanged versus DependancyProperties (which provides built in change notification)  **Implementation:**  InotifyPropertyChanged: Declaring that your class is implementing INotifyPropertyChanged adds an PropertyChangedEventHandler that you raise for every changes of the properties. We also add a little tricky method checkIfPropertyNameExists(String propertyName) which checks by reflection when debugging if the property name really exists ! You usually ends up with code like this :  DependancyProperty MSDN definition: a property that exists on a DependencyObject, is stored by the DependencyObject property store, and is identified by a DependencyProperty identifier on the owning DependencyObject.    **Which one to choose?**  Execution Times - DependencyProperty is 230 times faster than INotifyPropertyChanged for one binding and this number does not increase with the number of binded controls.!  Memory Usage - DependencyProperty seems (in my tests) to create less instance and to use less memory than the INotifyPropertyChanged system...  Inheritance issues   * To create a DependencyProperty your objects needs to inherit from DependencyObject. This is not always possible and then using INotifyPropertyChanged is the only way to make it Bindable-aware. * Also, by being a DependencyObject, your object will carry with it all the dependency engine stuff and these limitations: * only thread that the DependencyObject was created on may access DependencyObject directly. * DependencyObject seals Equals and GetHashCode(), * They are not marked as Serializable   Inheritance from a base class you do not have a grip on ?=> No DependencyProperty !  Coding Flexibility  Flexibility (easier code writing) needed ?=> Choose INotifyPropertyChanged !  Testing  When you perform testing on your object, you will be in trouble if you use DependencyObject : the test are not done on the same thread that created the object and then throws you a "System.InvalidOperationException: The calling thread cannot access this object because a different thread owns it".  Testing => No DependencyProperty !  Code Readability/writing  Some people argue that the use of DependencyProperties in code is extremely ugly. I myself think that this is exactly the same.  Reference Material -  <http://blog.lexique-du-net.com/index.php?post/2010/02/24/DependencyProperties-or-INotifyPropertyChanged>  <http://blog.quantumbitdesigns.com/2010/01/26/mvvm-lambda-vs-inotifypropertychanged-vs-dependencyobject/> |
|  | Difference between List<T>, ObservableCollection<T> and INotifyPropertyChanged   |  |  | | --- | --- | | List<T> represents a strongly typed list of objects that can be accessed by index. It provides methods to search, sort, and manipulate lists. The List<T> class is the generic equivalent of the ArrayList class. It implements the IList<T> generic interface using an array whose size is dynamically increased as required. |  | |

|  |  |
| --- | --- |
| Layout/Controls Concepts | |
|  | What are different types of Panels in WPF ? Explain them? OR Explain the types of Layout? |
|  | How placing controls in a WrapPanel differs from a Grid?  Grid is used to display the content in the form of rows and columns while Wrap panel is used to display the content one by one till the width size and then start with second row. |
|  | What is ContentPresenter?  In WPF there is an element called ContentPresenter, that is often used inside control templates, as well as inside the root application markup. The concept of ContentPresenter is quite simple – it is a placeholder for any XAML content and it can be used to insert content at runtime. |
|  | **What is the use of ContentControl class?**  **ContentControl is base for commonly used controls and offers a content property which accepts just one child element. But it is flexible to accept any element including other layout elements as child.** |
|  | **What is difference between ContentControl and ContentPresenter?**   * **Inside ControlTemplate use ContentPresenter** * **ContentControl is a base class for controls that contain other elements and have a "Content" property (for example, Button).** * **Outside of ControlTemplate (including DataTemplate and outside templates) try not to use any of them, if you must prefer ContentPresenter** |
|  | **What is the Difference between TextBlock and Label?**   |  |  |  | | --- | --- | --- | | **1** |  | **TextBlock derives directly from FrameworkElement and is not a control. <Label> internally calls Textblock for text rendering and hence is heavier to render than a textblock** | | **2** |  | **Label being control can be user interactive and can have an access key** defined by the Label to set focus to the other control. | | **3** |  | **TextBlock doesn’t have a behavior by default but Label is greyed out when its disabled.** | |
|  | Text vs Content property?  Text property will only accept simple text whereas content will accept simple text or any other element. |
|  | You need to display a list of items. Which controls are possible options? Of those, which would you use and why?  ItemsControl, ListBox, ListView, DataGrid.   * Make sure they know that ListBox and ListView can be selected while ItemsControl does not support selection. * ListView is more feature rich, as it inherits ListBox, which inherits ItemsControl, and adds features. * Make sure they understand what a DataGrid is and how it is different and why they would display a list in a DataGrid. |
|  | How would you create a custom panel control?  The custom panel we are going to write today is going to be extremely similar to the wrap panel, except that we are going to add animation into the picture. When items in the panel need to wrap to the next line because the size of the panel changed, they won't just switch positions - they will animate to the new position.   |  |  | | --- | --- | |  |  |  |  |  | | --- | --- | | Step 1: First thing we need to do is create the skeleton of the class for the new custom control. In my case, I called it AnimatedWrapPanel, and it will extend the base Panel class: |  | | Step 2: Technically, this is already a control, but it wouldn't actually do anything yet. To actually get it to do something useful, there are two very important methods that we need to override:  WPF uses a two pass layout system to determine the positions and sizes of parents and children. Each of these methods corresponds to one of those steps. In the first step, the Measure step, parents are supposed to ask their children what their desired size is, given an available size. In the second step, the Arrange step, parents position their children and tell their children how much size they are actually getting. |  | | Well in the MeasureOverride function, you are required to call Measure on every child. If you don't, then the child's desired size is never determined - and this will break the Arrange pass of the layout system. |  | | Again, you will want to call Arrange on every child, otherwise that child will not get placed/rendered. What we are doing here is placing every child at (0,0) and letting them be their desired size. |  | |
|  | What is the difference between ListBox and ListView?  ListView is a specialized ListBox (that is, it inherits from ListBox). ListView allows different views rather than a straight vertical list. It's basically the multi-column listbox. The most common view used within a ListView is called a GridView. In simple scenarios, a ListView can be used as a lightweight data grid (which was important in the initial release of the platform because there was no native DataGrid class). |
|  | How can I sort items in a ListBox (or any other ItemsControl)?  Sorting can be done via a mechanism on the ItemsCollection object, so it applies equally to all ItemsControls. ItemsCollection has a SortDescriptions property that can hold any number of System.ComponentModel.SortDescription instances. Each SortDescription describes which property of the items should be used for sorting and whether the sort is ascending or descending.  For example, below code should sort the listbox items in the descending order by fields ApplicatonName and InstanceName  listBox1.Items.SortDescriptions.Add(new SortDescription("ApplicatonName", ListSortDirection.Descending)); listBox1.Items.SortDescriptions.Add(new SortDescription("InstanceName", ListSortDirection.Descending)); |
|  | Difference between Visibility.Hidden and Visibility.Collapsed  **Visible**: Display the element.  **Hidden**: Do not display the element, but reserve space for the element in layout.  **Collapsed**: Do not display the element, and do not reserve space for it in layout. |
|  | What are the difference between CustomControls and UserControls in WPF?   |  |  | | --- | --- | | CustomControl (Extending an existing control)   * Extends an existing control with additional features. * Consists of a code file and default style in Themes/Generic.xaml. * Can be styled/templated. * The best approach to build a control library |  | | * Is Derieved from Control | | | This example of a "Numeric up/down" control is an extension of a textbox. The up and down buttons are defined in the default template of the control and wired up in the OnApplyTemplate() override in the logic part of the control. The ControlTemplate can easily be exchanged by another that has the up,down buttons aligned left for example. | |   UserControl (Composition)   * Composes multiple existing controls into a reusable "group". * Consists of a XAML and a code behind file. * Don’t need support for theming - Cannot be styled/templated. * Will not be shared across applications * Derives from UserControl.     This example of an "RGB user control" composes three labels and textboxes as well as a color field together to an reusable part. The logic in the code behind file adds a Color DependencyProperty that gets and sets the resolved color. |
|  | How to consume a usercontrol and/or custom control  Step 1: Add a reference in the MainWindow.xaml through xmlns:controls    Step 2:   |  |  | | --- | --- | | For UserControl |  | | For Custom Control |  | |
|  | Different methods for implementing absolute positioning in WPF (positioning of controls)   1. Controls not Nested in a Canvas (Use Margin)  |  |  | | --- | --- | |  |  |   You have to adjust your control’s Margins to specify the exact location of your control. In order to do that, I first must specify the default Horizontal and Vertical alignment.     1. Controls Nested in a Canvas      1. Transforms   Explanation Below |
|  | What are LayoutTransform and RenderTransform  The WPF layout system comprises, of two steps, followed by the rendering of the user interface (UI):   * Measure * Arrange * Render   In the Measure step, the DesiredSize of each element is computed;  in the Arrange step the position of child elements within their parents is determined;  finally, in the Render step, the result user interface is rendered to the screen.  Layout transforms and Render transforms are computed at different stages of the layout/render process:  **LayoutTransform**  Measure  Arrange  **RenderTransform**  Render  As a result, any transformations associated with an elements LayoutTransform property will have an impact on the subsequent Measure and Arrange steps. Whereas a RenderTransform will not have any impact on the layout process and will only effect rendering. The difference is probably best illustrated by an example:   |  |  | | --- | --- | |  |  | |  |  | | *A pair of Borders is created, each containing a single image. A RotateTransform is assigned to the LayoutTransform property of each border. This causes each Border and its contents to be rotated slightly, the first by positive 10 degrees and the second by negative 12 degrees. Margins, as defined in the imageBorder Style, between the Borders is maintained since the transform is applied at layout time. The Borders are fully contained within the containing StackPanel, being clipped to within the panel’s dimensions.*  *The Borders are repeated on the second row. This time RenderTransfom is used instead of LayoutTransform. Notice that the two Borders now overlap and are not clipped to the containing StackPanel. In this case the Borders were positioned and clipped in an untransformed state, i.e. unrotated. Then the RotateTransform was applied to each Border with the results shown here.* | |   In the above examples, it can be seen that when a LayoutTransform is applied, neighboring elements are re-positioned to accommodate the transformed elements, whereas with the RenderTransform they are not. One common use of WPF transforms is to rotate ListView column headings. With the above examples it should be obvious that a LayoutTransform is required to achieve this effect.  When possible, use a RenderTransform instead of a LayoutTransform.  A LayoutTransform can be a very useful way to affect the content of a user interface (UI). However, if the effect of the transform does not have to impact the position of other elements, it is best to use a RenderTransform instead, because RenderTransform does not invoke the layout system. LayoutTransform applies its transformation and forces a recursive layout update to account for the new position of the affected element.  Transform is an abstract class, and WPF provides several concrete implementations that can be applied to the two aforementioned properties in XAML and code. Some of them are:   |  |  | | --- | --- | | **RotateTranform**: Rotates an element by a specified Angle. You can specify the Angle of Rotation and the element will be rotated in 2D space.  **ScaleTransform**: ScaleTransform allows you to scale the element means increase/decrease the size of the element in the 2D space.  **SkewTransform**: It skews the element by specified angle. Skew stretches elements in a NonUniform manner and thus the element will be transformed so as in 3D space.  **TranslateTransform**: This transformation will make the element move by a specified X and Y Co-Ordinates |  | |
|  | What’s the difference between ComboBox’s IsEditable and IsReadOnly properties?  Setting IsEditable to true turns ComboBox’s selection box into a text box. IsReadOnly controls whether that text box can be edited, just like TextBox’s IsReadOnly property. This means that IsReadOnly is meaningless unless IsEditable is true, and IsEditable being true doesn’t necessarily mean that the selection text can be edited. Below in the table sums up the behavior of ComboBox based on the values of these two properties.  The Behavior for All Combinations of IsEditable and IsReadOnly   |  |  |  | | --- | --- | --- | | IsEditable | IsReadOnly | Meaning | | false | false | The selection box displays a visual copy of the selected item and it doesn’t allow the typing of arbitrary text. (The default behavior.) | | false | true | Same as above. | | true | false | The selection box displays a textual representation of the selected item and it allows the typing of arbitrary text. | | true | true | The selection box displays a textual representation of the selected item but it doesn’t allow the typing of arbitrary text. | |
|  | When the SelectionChanged event gets raised, how do I get the new selection?  The SelectionChanged event is designed to handle controls that allow multiple selections,  so it can be a little confusing for a single-selection selector such as ComboBox. The  SelectionChangedEventArgs type passed to event handlers has two properties of type  IList: AddedItems and RemovedItems. AddedItems contains the new selection and  RemovedItems contains the previous selection.  e.g.  void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)  {  if (e.AddedItems.Count > 0)  object newSelection = e.AddedItems[0];  }  Like this code, never assume that there’s a selected item! Besides the fact that ComboBox’s  selection can be cleared programmatically, it can get cleared by the user when IsEditable is  true and IsReadOnly is false. In this case, if the user changes the selection box text to  something that doesn’t match any item, the SelectionChanged event is raised with an  empty AddedItems collection. |
|  | Selector.IsSynchronizedWithCurrentItem Property?  You can set the Selector.IsSynchronizedWithCurrentItem property to true to ensure that the item selected always corresponds to the CurrentItem property in the ItemCollection. For example, suppose that there are two ListBox controls with their ItemsSource property set to the same source. Set IsSynchronizedWithCurrentItem to true on both list boxes to ensure that the selected item in each ListBox is the same.  Any ItemsControl binds to the ICollectionView of the ItemsSource property. The ICollectionView interface contains a member called CurrentItem. What the IsSynchronizedWithCurrentItem does is: whenever an item is clicked on the ItemsControl, it sets the CurrentItem for the collection view. The ICollectionView also has two events: CurrentItemChanging and CurrentItemChanged. When the IsSynchronizedWithCurrentItem property is set, the ItemsControl will update the SelectedItem based on what the ICollectionView's CurrentItem is. |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Style/Template/Binding Concepts | |
|  | Key Terms Map   |  |  | | --- | --- | | **Triggers**  - Property Trigger  - Data Trigger  -- MultiDataTrigger  - EventTrigger | **Styles** | | **Templates**  - Control Templates  Key Terms – Content Presenter. TemplateBinding - Data Templates  -- HierarchialDataTemplate  Key Terms – ItemsTemplate, DataTemplateSelector | |  | - ItemsPanelTemplate  Note: Trigger Properties are not available for this Template | |
|  | Explain Style vs. Template?  A style, represented by the System.Windows.Style class, is a pretty simple entity. Its main function is to group together property values that could otherwise be set individually. The intent is to then share this group of values among multiple elements.  Style uses a collection of Setters to set the target properties. Creating a Setter is just a matter of specifying the name of dependency property (qualified with its class name) and a desired value for it.   |  |  | | --- | --- | |  |  |   Template explanation is below. |
|  | What is Style Inheritance?  Styles can even inherit from one another. It is implemented through BasedOn property of the Style class |
|  | How can I apply multiple styles to a control  You cant apply multiple styles per se – but you can do this through Style Inheritance using BasedOn property |
|  | What are Triggers? Name the types  A trigger is a collection of Setter objects that are executed if the specified condition is fulfilled.  Property triggers—Invoked when the value of a dependency property changes  Data triggers—Invoked when the value of a plain .NET property changes  Event triggers—Invoked when a routed event is raised  Triggers are only available in styles and templates. |
|  | What information do you need to define a property trigger?  **Example 1**   * A target element (the element for which the trigger is defined) – TextBox (in this case) * A property to monitor (and trigger) – IsFocussed (in this case) * A value for the property to initiate the trigger event - True (when the Textbox gets focus) * A collection of Setters that are applied when property being watched has the specified value  |  |  | | --- | --- | |  | Before    After the Property Trigger is fired |   **Example 2**   * A target element (the element for which the trigger is defined) – Rectangle (in this case) * A property to monitor (and trigger) – Fill (in this case) * A value for the property to initiate the trigger event - LightGreen (in this case) * A collection of Setters that are applied when property being watched has the specified value  |  |  | | --- | --- | |  |  | |
|  | Different Ways to Set Triggers on Controls  See example G:\04 - Net Technical Material\!Visual Studio 2010 Projects\02-WPF\aMine\09-InterviewPracticeExamples\02-Triggers \01 - TextBox\_Ex, 02 - Button\_Ex, 03 - Border\_Ex   * You can set it through Property Triggers within Style (as shown in example 1) * You can set it through Control Template Triggers (as shown in example 2) * You can set it through EventTriggers (generally used with Animation) |
|  | Data Trigger Explanation  Data triggers are just like property triggers, except that they can be triggered by any .NET property rather than just dependency properties. (The Setters inside a data trigger are still restricted to setting dependency properties, however.)  The following TextBox has a Style that triggers the setting of IsEnabled, based on the value of its Text property, which is not a dependency property. When Text is the string “disabled”, IsEnabled is set to false.   |  |  | | --- | --- | |  |  |   It is important that trigger property is defined only in control template and data template, but not in items panel template. Because hierarchical data template is inherited by data template, therefore this will also have trigger property. Here is a class diagram of DataTemplate class.    A data trigger allows you to trigger off of a property on your data item by leveraging a Binding. This is most often used in the context of a DataTemplate, but don’t overlook the power of data triggers in other scenarios.  **Example of DataTrigger within DataTemplate**   |  |  | | --- | --- | |  | In this case, we are creating a DataTrigger that acts more like a property trigger by using a FindAncestor binding to get to a source object. Namely, we are triggering off of the IsSelected property on an ancestor ListBoxItem. This will allow us to draw a pink border around our template whenever it is selected. |   MultiDataTrigger  Now suppose our data item is a person and we only want a pink border when decorating a female. If the person is male, we instead want a blue border. We could achieve this by adding a second trigger, as shown in the following template:   |  |  | | --- | --- | |  | Now we have added a MultiDataTrigger with two conditions. The first contains a Binding to a property on the data item that identifies the gender of the person. The second is the same binding we used earlier to get at the IsSelected property on the ancestor ListBoxItem. Now if the person is male and they are selected, the border will be blue. |   Important Observations  The above example illustrates another important aspect of triggers and setters… namely that the order of triggers and setters is important - **if multiple setters target the same property, the last setter wins!**  The final thing I will point out about this example is that it demonstrates how to combine both property triggers and data triggers by leveraging a MultiDataTrigger. |
|  | Event Triggers  An event trigger, as the name implies, can be used to start an action in response to an event. More specifically, an event trigger executes in response to a routed event. Here’s a example of a Button that uses an event trigger to begin an opacity animation when the Button is first loaded:   |  |  | | --- | --- | | In this example, the event trigger is added directly to the Triggers collection of the framework element (the Button), itself. Again, the FrameworkElement.Triggers collection can only contain event triggers. If you wish to use property or data triggers, they must be within the Triggers collection of a style or template.  Note that you cannot trigger off of a standard CLR event because the mechanism for supporting event triggers depends on the WPF event routing engine, which intelligently routes events to only the elements that have registered to receive them. |  | |
|  | Triggers Behavior Notes  With a property or data trigger, the condition that causes the trigger to be applied is based on state. As such, the condition will be true for a period of time. As long as the condition evaluates to true, the setters are applied to their target properties. When the condition later becomes false, the setters are no longer applied and the target properties fall back to their un-triggered values.  There is no such concept of state with an event trigger. An event is fleeting. The event fires and then it’s gone. Since the condition that invokes an event trigger is instantaneous, an EventTrigger does not contain a collection of setters. Rather, it contains actions which execute in response to the event. These actions allow you to control (start, stop, pause, resume, etc) the execution of storyboards. The example above uses a BeginStoryboard action in response to the FrameworkElement.Loaded event. |
|  | What are Templates and its type in WPF?  A template, on the other hand, allows you to completely replace an element’s visual tree with anything you can dream up, while keeping all of its functionality intact. They all derive from the abstract FrameworkTemplate class.   |  |  | | --- | --- | | **ControlTemplate** – supplies a visual representation of a UI control, such as a Button or ListView.  **DataTemplate** - Used to create a visualization of a non-visual object, such as a business object.  **ItemsPanelTemplate** – ItemsControl, and all of its subclasses (such as ListBox), create the layout panel that hosts their child elements via an ItemsPanelTemplate.  **HierarchicalDataTemplate** - Data template that has knowledge of how to display a data object’s child objects, such as in a master-detail situation. |  | |
|  | What is a ControlTemplate?   |  |  |  | | --- | --- | --- | | A ControlTemplate is used to declare the visual representation for a control. Controls in WPF are separated into logic that defines the states, events and properties and template that defines the visual appearance of the control. The wire-up between the logic and the template is done by DataBinding.  Each control has a default template. This gives the control a basic appearance. The default template is typically shipped together with the control and available for all common windows themes. It is by convention wrapped into a style, that is identified by value of the DefaultStyleKey property that every control has.  Note that similar to a style, the ControlTemplate class exposes a TargetType property. This property must always be set.  The template is defined by a dependency property called Template. By setting this property to another instance of a control template, you can completely replace the appearance (visual tree) of a control. | |  | |  |  | |  |  |  | | --- | --- | | When you create a custom control template and you want to define a placeholder that renders the content, you can use the ContentPresenter. By default it adds the content of the Content property to the visual tree of the template. To display the content of another property you can set the ContentSource to the name of the property you like. |  | |
|  | Difference between Style and Control Template  Styles set properties on controls.  ControlTemplate is a property shared by most controls that specify how they are rendered.  To elaborate, you can use a style to group settings for a bunch of properties so you can re-use that to standardize your controls. Styles can be set explicitly on controls or applied too all of a certain type.  Control Templates can be set by a style or set explicitly on a control to change the way it appears. All controls have default templates (and styles for that matter) that are embedded in the .net wpf assemblies.  Important: Values set in the template can only be replaced by replacing the template while values in a style can be replaced by setting the value when using the control. That is why is better to use the properties of the control by using TemplateBinding instead of coding values. |
|  | What is DataTemplate  Data Template are a similar concept as Control Templates. They give you a very flexible and powerful solution to replace the visual appearance of a data item in a control like ListBox, ComboBox or ListView. In my opinion this is one of the key success factory of WPF.  If you don't specify a data template, WPF takes the default template that is just a TextBlock. If you bind complex objects to the control, it just calls ToString() on it. Within a DataTemplate, the DataContext is set the data object. So you can easily bind against the data context to display various members of your data object   |  |  | | --- | --- | |  |  | |
|  | Q. What is the use of Item Templates?  Gets or sets the DataTemplate used to display each item.   |  |  | | --- | --- | | The basic ItemsControl syntax will look like this:    By default, this will create a Vertical StackPanel, then it will loop through each item in MyCollection, add it to a TextBlock, then add that TextBox to the StackPanel. |  | | Quite often you don’t want to simply display an item as a TextBlock, and that’s where the ItemTemplate comes in. |  | |
|  | Difference between ControlTemplate and DataTemplate  Very basically a ControlTemplate describes how to display a Control while a DataTemplate describes how to display Data.  Typically a control is rendered for its own sake, and doesn't reflect underlying data. For example, a Button wouldn't be bound to a business object - it's there purely so it can be clicked on.  A ContentControl or ListBox, however, generally appear so that they can present data for the user.  A DataTemplate, therefore, is used to provide visual structure for underlying data, while a ControlTemplate has nothing to do with underlying data and simply provides visual layout for the control itself.  A ControlTemplate will generally only contain TemplateBinding expressions, binding back to the properties on the control itself, while a DataTemplate will contain standard Binding expressions, binding to the properties of its DataContext (the business/domain object or view model). |
|  | DataTemplateSelector  Provides a way to choose a DataTemplate based on the data object and the data-bound element. |
|  | HeirarchialDataTemplate  HierarchicalDataTemplate is an extension of DataTemplate with an additional ItemSource property. It is used to render hierarchical data in controls like treeview or menu. HierarchicalDataTemplate ends with a Data Template. |
|  | DataTemplate vs HierarchicalDataTemplate  DataTemplate – see answer above  HierarchicalDataTemplate is an extension of DataTemplate with an additional ItemSource property. It is used to render hierarchical data in controls like treeview or menu.  Hence HierarchicalDataTemplate which is a data template that has knowledge of how to display a data object’s child objects, such as in a master-detail situation. |
|  | ItemsControl vs ItemsTemplate vs ItemsPresenter  ItemsControl is conceptually a control that houses items. Try to simply think of this control as a holder for zero or more objects.  The default layout defined for the ItemsControl is a StackPanel so the three buttons in the snippet will be displayed one above the other, separated by a rectangle. It is very common to have a vertical list of items in the page and most of the time it suffice but in other cases you can have the need to display the items in a completely different layout. The default layout can be changed easily using the ItemsPanel property and ItemsPanelTemplate.  ItemsPresenter is part of the ItemsControl template that will define where the items are placed within it. Your ItemsControl's template can be anything you like, say a Grid with some pretty pictures around it, inside this template, you would place the ItemsPresenter where ever you want your items to be, say right in the middle of your grid. |
|  | What is the difference between ItemsControl, ItemsPresenter, ContentControl, ContentPresenter, DataTemplate, HierarchicalDataTemplate  ControlTemplate – A ControlTemplate is used to declare the visual representation for a control.  ContentControl – **ContentControl is base for commonly used controls and offers a content property which accepts just one child element. But it is flexible to accept any element including other layout elements as child.**  ContentPresenter – A stub in the ControlTemplate for where the content from the Content property of a ContentControl should be inserted  ItemsControl – Control that displays a collection of items  ItemsPresenter – An ItemsControl.Template can contain a ControlTemplate. The ItemsPresenter is the stub for where the items should be rendered in the ControlTemplate  DataTemplate – Defines how to visualize a business object (as opposed to a ControlTemplate that defines how to visualize a control)  HierarchicalDataTemplate – Defines how to visualize different levels of hierarchical data |
|  | Customize a DataGrid  <http://blog.smoura.com/wpf-toolkit-datagrid-part-ii-custom-styling/>  <http://stackoverflow.com/questions/12545512/style-a-wpf-datagrid>  <http://www.codeproject.com/Articles/586132/WPF-DataGrid-Custommization-using-Style-and-Templa> |
|  | Datagrid Real working example  <http://blogs.msdn.com/b/jaimer/archive/2009/02/10/m-v-vm-training-day-sample-application-and-decks.aspx> |
|  | Describe Resource Dictionary  It‘s a xaml file with bunch of property values which can be combined at application or window level to build resource collection.  The items in a ResourceDictionary are not immediately processed when application code is loaded by a XAML loader. Instead, the ResourceDictionary persists as an object, and the individual values are processed only when they are specifically requested.  The ResourceDictionary class is not derived from DictionaryBase. Instead, the ResourceDictionary class implements IDictionary but relies on a Hashtable internally. |
|  | Resource Dictionary – Further Details  <http://blogs.msdn.com/b/wpfsldesigner/archive/2010/06/03/creating-and-consuming-resource-dictionaries-in-wpf-and-silverlight.aspx> |
|  | DataBinding?  WPF provides a simple and powerful way to **auto-update data** between the business model and the user interface. This mechanism is called DataBinding. Everytime when the data of your business model changes, it automatically reflects the updates to the user interface and vice versa. This is the preferred method in WPF to bring data to the user interface.  Databinding can be **unidirectional** (source -> target or target <- source), or **bidirectional** (source <-> target).  The source of a databinding can be a normal .NET property or a DependencyProperty. The target property of the binding must be a DependencyProperty and should be bindable  To make the databinding properly work, both sides of a binding must provide a change notification that tells the binding when to update the target value. On normal .NET properties this is done by raising the PropertyChanged event of the INotifyPropertyChanged interface. On DependencyProperties it is done by the PropertyChanged callback of the property metadata  Databinding is typically done in XAML by using the {Binding} markup extension. The following example shows a simple binding between the text of a TextBox and a Label that reflects the typed value:   |  |  | | --- | --- | |  |  | |
|  | What are the different Data Binding Modes in Windows Presentation Foundation?  OneWay, TwoWay, OneTime, OneWaytoSource, Default |
|  | What is 'One-way-to-Source ' binding property?  In One-way-to-Source binding when the target property changes, the source object gets updated. |
|  | UpdateSourceTrigger property?  It’s a Binding property which drives change flow behavior from target to source.  “UpdatesourceTrigger” property is very important in “TwoWayBinding” and “OneWayToSource” binding modes. It defines how and when source should get updated.  Value of UpdateSourceTrigger could be any of the value from UpdateSourceTrigger Enumeration.  Enumeration values are:  PropertyChanged – The source is updated whenever the target property value changes.  LostFocus – The source is updated when target property changes and target object looses focus.  Explicit – The source is updated when explicit call is made to update using “BindingExpression.UpdateSource”.  (For most properties, the default behavior is PropertyChanged although the TextBox.Text property has a default behavior of LostFocus which saves overhead of updating source object while user is typing).   |  |  | | --- | --- | |  |  | |
|  | How to ensure that a value is displayed always even if binding fails?  Fallbackvalue attribute – use when binding will fail |
|  | Explain Binding vs. TemplateBinding in WPF   * TemplateBinding is a markup extension. * Template bindings are similar to ordinary data bindings, but they’re lighter weight because they’re specifically designed for use in a control template. * TemplateBindings are evaluated at compile time against the type specified in the control template whereas the binding markup is resolved at runtime. * One more thing - TemplateBindings don't allow value converting. They don't allow you to pass a Converter and don't automatically convert int to string for example (which is normal for a Binding). |
|  | TemplateBinding Example  This example shows how a Control Template is used for customizing Tool Tip:    This control template is too rigid. What if another team member needs to create a special ToolTip in just one place with a background color other than NormalControlBackground? They would probably write some markup like this:   |  |  | | --- | --- | |  | Sadly, they would discover that it wouldn't work. The control template doesn't know anything about the properties we set on the ToolTip in the markup. Everything is defined in the control template and it's self contained. We really want the template to be able to use the properties that are set on the control. |   The template above can be rewritten to do that:    Using the template bindings allows the control template to pick up the values that are supplied directly on the element, or on the style applied to the element.    How to handle default values when using Template Binding  If you use a control template with template binding, and don't provide any values on the control, what happens? The default values of the bound properties are used. There are many cases where this is not what you want though. If your intent was to have a global look for tool tips based on our first example, then you want the template to use some specific brushes as well a few other things.  In this case, you can use a style to set the desired defaults. |
|  | Here is a TextBox. Change it so that it will bind to the ‘Text’ property of any DataContext.  <TextBox />  <TextBox Text=”{Binding Path=Text}” /> |
|  | You want to use a custom Binding but the control does not support binding in the area you want. What are your options? Which would you choose and why?  - Inherit object and add a Dependency Property – If the object is not sealed this is likely an easier option.  - Create an attached property – The object might be sealed. |
|  | How do you bind a button’s Command property to a method?  You don’t. You bind to an ICommand. You create a class that implements ICommand and in that object you connect to your method. |
|  | What happens if the value you are binding to does not exist?  The error is ignored. If running from Visual Studio, a message is logged in the Output window. |
|  | What is the Binding syntax for binding to a Static value?  {x:Static s:MyStaticClass.StaticValue2} |
|  | What is the correct syntax for binding to a property of a Singleton?  {Binding Source={x:Static sing:MySingletonClass.Instance}, Path=SomeProperty} |
|  | **DataContext property?**  **Every WPF control derived from FrameworkElement has a DataContext property. This dependency property is meant to be set to the data object it visualizes. If you don't explicitly define a source of a binding, it takes the data context by default.**  **The DataContext property inherits its value to child elements. So you can set the DataContext on a superior layout container and its value is inherited to all child elements. This is very useful if you want to build a form that is bound to multiple properties of the same data object.** |
|  | MultiBinding   |  |  | | --- | --- | | It is a binding that enables you to bind to multiple items and return a single new value using a converter. This is extremely useful if a control in your interface needs to be affected by a number of backend property changes - for instance, any sort of aggregator. Below is a screenshot of the example we are going to build today - and as you can see, we will be doing some aggregating. We will be using a MultiBinding to sum the values of the three sliders and report the total. |  | |
|  | Provide an example of when and why you would use MultiBinding.  - When you have multiple values that your which to combine.  - When you want to use String.Format in XAML. |
|  | CollectionViewSource   * WPF has a powerful data binding infrastructure. It allows you to bind almost any kind of collection directly to a view. But when it comes to sorting, filtering and grouping the support of the collections is rare. That's the point where the CollectionView comes into play. * It is the primary Data object for any WPF list controls (like ComboBox, ListBox, ListView etc) that allows flexibilities like Sorting, Filtering, Grouping, Current Record Management etc. Thus it ensures that all the related information like filtering, sorting etc is decoupled from the actual control. It is been very popular to those working with data object because of inbuilt support for all WPF List controls.  |  |  | | --- | --- | | * A collection view is a wrapper around a collection that provides the following additional features: * Navigation * Sorting * Filtering * Grouping |  |  * CollectionViewSource is NOT a view - it doesn’t even implement ICollectionView – a requirement for a class to be considered a “view”. CollectionViewSource is simply a class that once given a collection (by setting its Source property) creates and exposes the corresponding view (through the View property) * Collection view classes work by wrapping an underlying collection of items so that they can provide automatic selection tracking and sorting, filtering, and paging for them. An instance of these classes can be created programmatically or declaratively in XAML using the CollectionViewSource class. * Collection view classes can be used by the view model to keep track of important state information for the underlying collection, while maintaining a clean separation of concerns between the UI in the view and the underlying data in the model. In effect, CollectionViews are view models that are designed specifically to support collections. * Therefore, if you need to implement filtering, sorting, grouping, or selection tracking of items in the collection from within your view model, your view model should create an instance of a collection view class for each collection to be exposed to the view. You can then subscribe to selection changed events, such as the CurrentChanged event, or control filtering, sorting, or grouping using the methods provided by the collection view class from within your view model. * The view model should implement a read-only property that returns an ICollectionView reference so that controls in the view can data bind to the collection view object and interact with it. All WPF and Silverlight controls that derive from the ItemsControl base class can automatically interact with ICollectionView classes.   Implementation:  Example 1: <http://wpftutorial.net/DataViews.html>  Example 2: <http://www.abhisheksur.com/2010/08/woring-with-icollectionviewsource-in.html>  Example 3 - <http://www.zagstudio.com/blog/387#.UUntLhzvsmM> |
|  | IDataErrorInfo  <http://codeblitz.wordpress.com/2009/05/08/wpf-validation-made-easy-with-idataerrorinfo/>  <http://tarundotnet.wordpress.com/2011/03/03/wpf-tutorial-how-to-use-idataerrorinfo-in-wpf/>  Extra Info - When “IDataErrorInfo” interface implemented and bound value to the control is invalid, control will have a red border around it to indicate user about the validation failure.    The problem of not being able to see all such decoration is absence of Adorner layer. All these error templates and focus styles etc. are drawn inside the AdornerLayer. The WPF window has an AdornerLayer by default, so we can always see validation errors in WPF |

|  |  |
| --- | --- |
| Converter Concepts | |
|  | Type Converter  Markup as an extension of XAML cannot impose restriction over a data element. That means we can only specify string data for attributes of any object in XAML. But XAML provides a flexibility to create your Type converter which allows you to impose restriction on the data. Thus even primitives like Single or Double could not have restrictions while you describe in your XAML. Type Converters plays a vital role to put this restriction to XAML parser.  XAML parser while parsing any value of an attribute needs two pieces of information.   * Value Type : This determines the Type to which the string data should be converted to. * Actual Value |
|  | What is IValueConverter used for?  Its an interface that provides a way to apply custom logic to a binding. If you want to associate a value converter with a binding, create a class that implements the IValueConverter interface and then implement the Convert and ConvertBack methods. |
|  | How to implement the IValueConverter?  Step 1: Declare the class which implements the IValueConverter (having Convert and ConvertBack methd    Step 2: Add it as a resource in your XAML – so you can access it    Step 3: Use the converter for conversion |
|  | Why is it better to use an IValueConverter instead of performing the conversion in the ViewModel?  This follows the Don’t Repeat Yourself (DRY) principle. It also follows the Single Responsibility Principle (SRP).  Because you may have to perform the same conversion between multiple View and ViewModel combinations. If each ViewModel has the code, the code is duplicated. If the code is in an IValueConverter, it exists in one place and is resusable. |
|  | How can binding values from data objects be changed when bound to WPF controls? For example, changing the format of a DateTime string  Using value converters (that implement IValueConverter) |
|  | Provide an situation in which you have or would use a ConverterParameter.   * Anytime you have a converter that returns something different based on a parameter.I have a converter that convert date to string. I used converter parameter to define if It's Departure date or arrival date to return single string in the required format * I have a converter used by many user controls each user control represents specific class type. I used converter parameter to hold the class type so I can control the conversion behavior based on the parameter type |
|  | MultiBinding and MultiValueConverter  Multiple binding enables target to bind with multiple sources. It aggregate multiple bindings together and out single value for target. Multibinding requires converter class which is responsible to combine all different types and values from different binding class and return single type and value for target. The converter class must implement IMultiValueConverter interface. This interface quite similar to IValueConverter and contains two methods Convert and ConvertBack. Convert Method contains array of values as parameter and ConvertBack method contains array of types as parameter.    Example - Sometimes we have to convert value depends on more than one variables. In that case we are going to use the multi value converter. Lets suppose we want to define the control template of progress bar and instead of displaying its current value, we want to display the current progress in percentage. |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Dispatcher Concepts | |
|  | How is dispatcher used in WPF  <http://weblogs.asp.net/pawanmishra/archive/2010/06/06/understanding-dispatcher-in-wpf.aspx> |
|  | What is a dispatcher Object?  Application Dispatcher – what is it and how does it work?   * Most classes in WPF derive from DispatcherObject * It emulates a Single Thread Apartment behavior. * The thread that creates a WPF UI element owns the elements and other threads can not interact with the UI elements directly, this is known as thread affinity. * WPF objects that have thread affinity derive from DispatcherObject at some point in their class hierarchy * At construction, a DispatcherObject stores a reference to the Dispatcher linked to the currently running thread. In effect, the DispatcherObject associates with the thread that creates it.   **Background**   * All WPF applications start out with two important threads, one for rendering and one for managing the user interface. The rendering thread is a hidden thread that runs in the background, so the only thread that you ordinarily deal with is the UI thread. * WPF requires that most of its objects be tied to the UI thread. This is known as **thread affinity**, meaning you can only use a WPF object on the thread on which it was created. Using it on other threads will cause a runtime exception to be thrown. * The thread affinity is handled by the Dispatcher class, a prioritized message loop for WPF applications. Typically your WPF projects have a single Dispatcher object (and therefore a single UI thread) that all user interface work is channeled through. * In addition to using the Dispatcher's message loop to channel items of work through the user interface thread, every WPF object is aware of the Dispatcher that is responsible for it (and therefore, the UI thread that it lives on). This means that any attempts to update WPF objects from secondary threads will fail. This is the responsibility of the DispatcherObject class.   **Dispatcher Object**  The DispatcherObject class has two chief duties:   * to provide access to the current Dispatcher that an object is tied to and * provide methods to check (CheckAccess) and verify (VerifyAccess) that a thread has access to the object (derived from DispatcherObject).   1. CheckAccess() : This method provides access to current dispatcher that an object is tied to . This returns a Boolean value as true if the current thread has access to use the object and returns false if the current thread can not use the object.  2. VerifyAccess() : The purpose of this method is to Verify if the thread has access to the object. If the thread does not have access to the object, an exception is thrown.  By providing this basic functionality, all the WPF objects support being able to determine whether they can be used on a particular thread—specifically, the UI thread.    If you are working on a non-UI thread, you'll need to use the Dispatcher to update DispatcherObjects.  **Using the Dispatcher**  If only one thread can modify the UI, how do background threads interact with the user? A background thread can ask the UI thread to perform an operation on its behalf. It does this by registering a work item with the Dispatcher of the UI thread. The Dispatcher class provides two methods for registering work items: Invoke and BeginInvoke. Both methods schedule a delegate for execution. Invoke is a synchronous call – that is, it doesn’t return until the UI thread actually finishes executing the delegate. BeginInvoke is asynchronous and returns immediately.   * Dispatcher is an instance of the class System.Windows.Threading.Dispatcher. * Dispatcher maintains a prioritized queue of work items for a specific thread. * The Dispatcher class provides a gateway to the message pump in WPF and provides a mechanism to route work for processing by the UI thread. This is necessary to meet the thread affinity demands. * When the Dispatcher is created on a thread, it becomes the only Dispatcher that can be associated with the thread, even if the Dispatcher is shut down. * If you attempt to get the CurrentDispatcher for the current thread and a Dispatcher is not associated, a Dispatcher will be created. * If a Dispatcher is shut down, it cannot be restarted. * The Dispatcher class provides access to invoke code on the UI thread directly. * In WPF, a DispatcherObject can only be accessed by the Dispatcher it is associated with. Note that every visual (Textbox, Combobox etc) inherits from DispatcherObject. For this reason, a background thread cannot update the content of a Button that is associated with the Dispatcher on the UI thread. This is accomplished using either Invoke or BeginInvoke. Invoke is synchronous and BeginInvoke is asynchronous. The operation is added to the queue of the Dispatcher at the specified DispatcherPriority.   Synchronously   * Dispatcher.Invoke   Asynchronously   * Dispatcher.BeginInvoke method of the Dispatcher to asynchronously queue up a work item for the UI thread * Background Worker class: Introduced in Windows Forms 2.0, a class for non-UI thread handling to simplify the development model for UI developers   Extra Reading - <http://msdn.microsoft.com/en-us/magazine/cc163328.aspx> |
|  | How can worker threads update the UI?  WPF is very similar to Windows Forms in terms of its multi threaded capabilities. While it does use multiple threads as part of the rendering pipeline as far as a user of WPF is concerned the UI is single threaded. So the old rule that we must do all of our UI updates from the main thread still applies.  In Windows Forms we would use Control.Invoke to send work back to our main thread. WPF is very similar. It provides the Dispatcher which allows us to Invoke back onto the main thread.  Its reasonably straight forward to use. We provide a delegate that we would like to run on the main thread, and ask the Dispatcher to run it using either Invoke (for synchronous running) or BeginInvoke (for async running). |
|  | CheckAccess and VerifyAccess  “DispatcherObject” class v-voffers two methods namely CheckAccess and VerifyAccess.  CheckAccess : Determines whether the calling thread has access to this DispatcherObject. Only possible return values are true and false.  VerifyAccess : VerifyAccess is more stringent compared to CheckAccess. It checks whether the calling thread is the one which created the DispatcherObject. If not it throws “InvalidOperationAccess” exception. Getting back to the example which I gave in the beginning, in that we were trying to update the UI through code which was running on a different thread. Internally VerifyAcess is invoked and once it verifies that calling code is different from the one which controls the application, it throws the “InvalidOperationError” exception. |
|  | BackgroundWorker in WPF (and how Dispatcher works behind the scenes in this)  In Windows Forms 2.0, Microsoft introduced a class for non-UI thread handling to simplify the development model for user interface developers. This class is called the BackgroundWorker. Code below shows typical usage of the BackgroundWorker class.    The BackgroundWorker component works well with WPF because underneath the covers it uses the AsyncOperationManager class, which in turn uses the SynchronizationContext class to deal with synchronization. In Windows Forms, the AsyncOperationManager hands off a WindowsFormsSynchronizationContext class that derives from the SynchronizationContext class. Likewise, in ASP.NET it works with a different derivation of SynchronizationContext called AspNetSynchronizationContext. These SynchronizationContext-derived classes know how to handle the cross-thread synchronization of method invocation.  In WPF, this model is extended with a DispatcherSynchronizationContext class. By using BackgroundWorker, the Dispatcher is being employed automatically to invoke cross-thread method calls. The good news is that since you are probably already familiar with this common pattern, you can continue using BackgroundWorker in your new WPF projects. |
|  | What is the relationship between threads and Dispatchers?  Dispatcher / Thread relationship is one to one, i.e. one Dispatcher is always associated with one thread and can be used to dispatch execution to that thread. Dispatcher.CurrentDispatcher returns the dispatcher for the current thread, that is, when you call Dispatcher.CurrentDispatcher on a worker thread you get a dispatcher for that working thread.  WPF application by default has only one Dispatcher. The dispatcher is the only thread that will allow you to interact with UI elements.  Dispatchers are created on demand, which means if you access Dispatcher.CurrentDispatcher and there is no dispatcher associated with the current thread, one will be created. |
|  | Running WPF Application with Multiple UI Threads  <http://eprystupa.wordpress.com/2008/07/28/running-wpf-application-with-multiple-ui-threads/> |
|  | How can I create a ProgressBar and use an animation to simulate the progress of an operation? |
|  | WHAT IS A FREEZABLE? What is its purpose?  A freezable object is one that has a mechanism that allows you to "Freeze" it. This locks downs all the state and makes the object immutable. This makes the object more performance to use and safer to share between threads. |
|  | How can I determine whether a Freezable Is Frozen?  This example shows how to determine whether a Freezable object is frozen. If you try to modify a frozen Freezable object, it throws an 'Invalid Operation' Exception. To avoid throwing this exception, use the "IsFrozen" property of the Freezable object to determine whether it is frozen. |
|  | How can I obtain a writable copy of a Read-Only Freezable?  This example shows how to use the Clone method to create a writable copy of a read-only Freezable. After a Freezable object is marked as read-only ("frozen"), you cannot modify it. However, you can use the 'Clone' method to create a modifiable clone of the frozen object. |
|  | Why do elements inherit from the DispatcherObject?  <http://joel.inpointform.net/software-development/net-interview-questions-wpf/> |
|  | How can you do update UI in WPF in Async multi-threaded environment? |
|  | What is the difference between WPF Dispatcher.Invoke and Windows Forms Control.Invoke |
|  | What is the WPF Dispatcher? how does this use frames? |
|  | What is the equivalent of Winforms Control.InvokeRequired in WPF? |
|  | What is the difference between Winforms Control.Invoke Requiredand Dispatcher.CheckAccess |
|  | What if you want to clear the update queue on the Dispatcher? |
|  | How do you queue UI updates on the Dispatcher? |
|  | Dispatcher in cross thread communication  <http://10rem.net/blog/2012/01/10/threading-considerations-for-binding-and-change-notification-in-silverlight-5> |
|  |  |
|  |  |
|  |  |

<http://leecampbell.blogspot.com/2009/01/responsive-wpf-user-interfaces-part-2.html>  
<http://tech.pro/tutorial/800/working-with-the-wpf-dispatcher>  
<http://msdn.microsoft.com/en-us/magazine/cc163328.aspx>  
<http://blog.decarufel.net/2009/03/good-practice-to-use-dispatcher-in-wpf.html>

|  |  |  |
| --- | --- | --- |
| Commands & Events Concepts | | |
|  | | What are WPF Commands and Events and differences between them  **Events**  Events represent low level interactions.  Events are faster than commands.  **Commands**  Commands respond to actions. Commands represents high level input interactions like print, copy, paste, Which commands are associated to which action.  Commands can be called from Menu item, Toolbar button, Keyboard, etc. Commands are more standard way to integrate Events.  **Distinction**  Although Commands and Events can be overlapping, they are two different things. Commands say "do this!", while events say "this just happened!". So you might have a CloseWindowCommand for closing a window, but the window might have a ClosingEvent that tells subscribing objects that is is closing.  **Advantages of Commands**  - Commands do not link to the caller, so commands are independent .  - A command can tell us if an action is possible or not. This can be done by implementing the CanExecute method. A button can attach CanExecuteChanged event and will be disabled if CanExecute returns false. The same button can be enabled if CanExecute returns true. So Commands provide support for enabling/disabling all related UI controls based on this status of command.  Its best practice to use commands over events at real project, especially if you use MVVM design pattern. |
|  | | What are Types of events in WPF OR What is Event Routing or a Routed Event? OR What are the event routing strategies?  Event Routing or A Routed Event is a type of event that is has the ability to invoke handlers not only on the object that raised the event but also on multiple listeners in an element tree.  Events that can traverse the logical tree and can be handled at higher or lower levels elements in the tree to write effective event handling code  Types of routed events –  Tunneling - Move lower down the element tree  Bubbling - Move higher up the element tree  Direct - Routed events that gets fired only at the element level and don’t tunnel or bubble.    **Event Tunneling**  The order of events firing for the controls in Event Tunneling is exactly opposite to the event Bubbling. Imagine that we have Stack Panel inside Window, a rectangle control inside the StackPanel. If you we fire a tunneled event on a Window, it will be first fired on the Window, then StackPanel and then on rectangle. ex: PreviewMouseDown   * All the Preview events tunnels down   **Event Bubbling**  It starts from Target and then bubbles up to the root . Imagine that we have Stack Panel inside Window , a rectangle control inside the StackPanel. If you we fire a bubbled event on a rectangle , it will be first fired on the rectangle , then StackPanel and then on window. ex: MouseDown   * Most routed events use the bubbling routing. * Bubbling routed events are used to report input changes from other UI elements. * All Main events Bubble.   In WPF, if an event is defined to tunnel and bubble, tunnelling occurs first (as a preview event) and then once reaching the source, the event bubbles back to the root  **Advantages in RealTime**  Routed events support a class handling mechanism in which the class specifies static methods that handles routed events before registered instance handlers can access them. This is concept is useful in control design as custom class can enforce event-driven class behaviors. |
|  | | What is difference between Event Bubbling Vs Event Tunneling? When to apply what?  If you put an image on a button, and that image is clicked, the tunnel event for that image will fire. But it is highly likely that you would also want to handle that click as if the button itself was clicked, so a bubble event is also fired which bubbles up to the click event on the button itself (because the button owns the image), so that you can treat the image click like a button click, using the usual event handler for the button.  I would also like to add that the event for both (Tunnel and Bubble) are different. For tunnel, we use PreviewXXX (eg: PreviewMouseDown) and for bubble event we use XXX (eg: MouseDown) event. By sequence, Tunnel event gets fired first starting from the root of the application and ends at the source. And the Bubble event is triggered. It makes perfect sense to have both these events. |
|  | | What is the use of EventManager class?  Provides event-related utility methods that register routed events for class owners and add class handlers.  To define a new even we first need to use the EventManager class to register the new event.  This is how the system knows how to handle the event and in particular what sort of routing is to be applied. The RegisterRoutedEvent returns a RoutedEvent object which has to be stored as a static field for later use within the new control.  Code Example - <http://msdn.microsoft.com/en-us/library/system.windows.eventmanager(v=vs.85).aspx> |
|  | | Routed Events and relationship to Class and Instance Listeners  Routed events consider two different types of listeners to the event: class listeners and instance listeners. Class listeners exist because types have called a particular EventManager API, RegisterClassHandler, in their static constructor, or have overridden a class handler virtual method from an element base class. Instance listeners are particular class instances/elements where one or more handlers have been attached for that routed event by a call to AddHandler.  Things are pretty simple: you can register handler at instance (object) level, or at class level.  The difference is, when you register event at class level, it will get called first, before any instance level handlers (of course tunneling or bubbling still takes place before, if handling class is lower/higher in logical tree). So you can handle this event at class level and filter whether this event should be handled by instance or not (by setting e.Handled = true you will stop event for going through other handlers). It may be useful in some cases, but for now I have no example in my mind to share.  This sample will register event handler that will be called only when event was raised for specific instance of element:    And this will create event handler, that will be called each time any DockPanel will get Button.Click event, before instance handler of this DockPanel will get called:    If methods were:    This would create output:    But if in class level handler you would set event args to handeled (e.Handled = true;), it would filter out this event for instance level handler (and bubbling up in logical tree). |
|  | | What is the RoutedCommands?  A routed command is a command object that does not know how to accomplish the task it represents. It simply represents the task/feature. When asked if it can execute and when told to execute, it simply delegates that responsibility off to somewhere else.   * They provide a method that allows events for multiple elements share the same command * The command is defined in a CommandBindings element. Any of the child elements of the main container can reference the commands |
|  | | What does ICommand Interface  2 methods (CanExecute and Execute) and 1 event (CanExecuteChanged) |
|  | | Difference between regular Command (ICommand) and Routed Command  The ICommand interface supports the command pattern. That allows you to abstract user actions into a re-usable class.  Routed commands are a particular implementation of ICommand that search through the visual tree for handlers. They are particularly useful for commands that can be implemented by many different controls, and you want the current control to handle it.  The key difference is that RoutedCommand is an ICommand implementation that uses a RoutedEvent to route through the tree until a CommandBinding for the command is found, while RelayCommand does no routing and instead directly executes some delegate.  Think copy/paste. There could be a whole bunch of controls that might handle it, but by using the routed command, the routed command system will automatically find the correct control to handle the command based on focus.  In a M-V-VM scenario a RelayCommand (DelegateCommand in Prism) is probably the better choice all around. |
|  | | Why use Routed Commands?  You might be wondering why it is a good idea to use routed commands at all. Why bother? What’s wrong with just hooking a Button’s Click event and doing things the “normal” way?  Well, you do not have to use routed commands if you do not want to. You certainly can just hook a Button’s Click event and go to town. By extension, why bother with a data layer and a business layer? Why not just stick your whole application into Window1.xaml.cs and be done with it? That would be much easier, right?  Advantages   1. All controls using the same RoutedCommand will automatically be disabled when the command cannot execute. 2. There is less event handler code to write since most of the wiring is provided for you by the commanding system. You do not have to add event handlers for each UI element that executes the same command, whereas using events directly off of UI elements requires many handlers that all basically do the same thing. 3. Using routed commands makes it possible to decouple the Software Engineering team from the Visual Design team. The developers don’t have to worry about what type of element is consuming application functionality, just as long as the UI executes the right commands all is well. This also frees the designers from having to worry about such details so that they can focus on creating a great user experience. 4. If you use my implementation of Model-View-Controller or Structural Skinning, using routed commands is an absolute must. |
|  | | Difference between RoutedCommand and RoutedUICommand?  RoutedUICommand derives from RoutedCommand and adds a text field used to provide a description of the command. This is useful when the command is bound to be accessed from the UI. |
|  | | How to create Custom Commands?  You will typically need to create custom commands if the ones you get out of the box from WPF is insufficient.  System.Input.ICommand interface should be implemented which defines a CanExecute property, Execute method and CanExecuteChanged event handler. |
|  | | What does this command do and when would you use it? CommandManager.InvalidateRequerySuggested()  It causes the any commands, such as a binding to Button.Command to check again if it can execute.  It is used when the command runs and the button should be enabled when the command completes, but the button stays disabled. This often occurs when running the command on a thread or by using a BackgroundWorker. |
|  | | You have a panel with 3 buttons in it. The panel has the attribute Button.Click=”HandleButtonClick”. When any of the buttons in the panel are clicked, HandleButtonClick is called. Explain why this works.  This is based on event bubbling. When the event travels from the source to the root, it reaches the panel containing the 3 buttons. As it passes each container, it checks to see if the container defines the button’s attached click event defined. In this case, the panel has the event defined and therefore can handle the button click for any child element. |
|  | | In routed event, if you handle event at one handler, does it still bubble up? If No, is there a way you can force it?  Yes – by default all routed events bubble up all the way to the parent even if e.Handled=true is called at the child control. Using that makes it seem that that the event is not bubbling up but if we use AddHandler methodology – then you can catch events at any level (even if it handled) |
|  | | From Training  AddHandler methodology (instead of +=) and when to use  AddHandler adds a routed event handler for a specified routed event, adding the handler to the handler collection on the current element. Specify handledEventsToo as true to have the provided handler be invoked for routed event that had already been marked as handled by another element along the event route. |
|  | |  |
| Design Pattern Concepts | | |
|  | Design Patterns used besides MVVM in WPF –  1) Command Pattern  2) Attached Behavior Pattern  3) Mediator Pattern  4) Singleton  5) Observer pattern  6) Adapter  7) Dependancy Injection  8) How/when to do a Weak Event Pattern with IWeakEventListener and subclassing WeakEventManager.  <http://www.cours.polymtl.ca/inf3700/divers/nonSoftwareExample/patexamples.html>  <http://stackoverflow.com/questions/182707/what-design-patterns-have-you-used-in-your-wpf-efforts-and-which-do-you-like>  <http://stackoverflow.com/questions/6441143/design-patterns-used-in-wpf>  MVVM - <http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx> | |
|  | What is the purpose of MVVM?   1. To provide a design pattern that allows for separation of concerns. The UI is independent of the data and the data is independent of the UI. 2. Easier Automated Testing 3. Abstractation-Now the view can be really abstract and you can use just a generic.xaml file and then give a style template to your model. | |
|  | Describe MVVM.  It is probably easiest to describe it in relation to MVP and MVC.  **Common Entities**  Model –   * Domain entities & functionality * Knows only about itself and not about views, controllers, etc. * For some projects, it is simply a database and a simple DAO * For some projects, it could be a database/file system, a set of entities, and a number of classes/libraries that provide additional logic to the entities (such as performing calculations, managing state, etc)   View –   * The interface the user interacts with * Code that handles the display * Note that view related code in the codebehind is allowed   Controller , ViewModel, and Presenter - These all provide methods for separating the view from the model  **Controller (ASP.NET MVC)**   * One-to-many relationship between the Controller and the views. One controller will be responsible for initiating the display of many different views. * There are numerous variations of how the views should or should not communicate with the models in MVC. With ASP.NET MVC, all requests are intercepted and redirected by a controller.   **ViewModel (WPF, knockout js)**   * One-to-one relationship between the view model and the views * Views interact with the view model by binding (for example, WPF data binding) to the view model’s properties/functions   **Presenter (WinForms)**   * One-to-one relationship between the Presenter and the views * There is no mechanism to bind the views to a view model so instead, each view implements an interface providing the means for the presenter to interact with the view. In addition, each view has a reference to the presenter. This allows views to forward messages to the presenter.     More Details if needed - <http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/> | |
|  | Q: When should you use “code-behind” in MVVM.  Note: Don’t except ‘never’ as an answer.  - When the code only involves the View or WPF Controls in the View.  - When you need to implement events that do not support binding to ICommand. | |
|  | What gotchas have you experience while using the MVVM pattern?  No real answer here…just that they have experienced problems proves their experience. Example issues:  - The ViewModel just becomes a place for all the code-behind.  - Determining whether the View can, should, or should not reference the ViewModel.  - Determining if View objects can, should, or should be in the ViewModel.  - When breaking Views down, how small to go.  - When breaking ViewModels down, how small to go. | |
|  | Bad things with MVVM  Double click event – easier to do outside MVVM | |
|  | What object do most ViewModels inherit from:  Either ObservableObject or ViewModelBase. | |
|  | How can I handle the input event through XAML?  By creating property for that event in viewmodel class . | |
|  | MVVM Frameworks Comparison  <http://www.japf.fr/2009/10/a-quick-tour-of-existing-mvvm-frameworks/> | |
|  | How do you implement binding a button click to a method?  Create a class that implements ICommand, often called RelayCommand or RoutedCommand.  Add an ICommand property to your ViewModel and instantiate the property using your ICommand implementation, RelayCommand or RoutedCommand. | |
|  | Details on Command Pattern  The command pattern is a common software design pattern that allows you to easily encapsulate the logic needed to defer execution of a method. It's commonly used to handle control actions in a unified manner.  Much of the power of commands comes from the following three features:   * WPF defines a number of built-in commands. * Commands have automatic support for input gestures (such as keyboard shortcuts). * Some of WPF's controls have built-in behavior tied to various commands.   The command pattern is well suited for handling GUI interactions. It works so well that Microsoft has integrated it tightly into the Windows Presentation Foundation (WPF) stack. The most important piece is the ICommand interface from the System.Windows.Input namespace. Any class that implements the ICommand interface can be used to handle a keyboard or mouse event through the common WPF controls. This linking can be done either in XAML or in a code-behind.  A command is any object implementing the ICommand interface (from System.Windows.Input), which defines three simple members – first 2 being delegates and third being an event:  **Execute**:The method that executes the command-specific logic  **CanExecute**: A method that evaluates the state of command and returning true if the command is enabled or false if it is disabled  **CanExecuteChanged**: An event that is raised whenever the value of CanExecute changes  Built in Commands Within WPF –  ApplicationCommands: Close, Copy, Cut, Delete  ComponentCommands: MoveDown, MoveLeft, MoveRight, MoveUp  EditingCommands: AlignCenter, AlignJustify, AlignLeft, AlignRight  NavigationCommands: BrowseBack, BrowseForward, BrowseHome  MediaCommands: ChannelDown, ChannelUp, DecreaseVolume  Each of these properties does not return a unique type implementing ICommand. Instead, they are all instances of RoutedUICommand, a class that not only implements ICommand, but supports bubbling just like a routed event.  Extra read (Detailed step by step on implementing ICommand) - <http://www.rhyous.com/2010/11/29/wpf-databinding-to-methods-encapulated-in-an-icommand/> | |
|  | Details on Attached Behavior Pattern  The Attached Behavior pattern encapsulates "behavior" (usually user interactivity) into a class outside the visual hierarchy and allows it to be applied to a visual element by setting an attached property and hooking various events on the visual element.  One rough analogy to behaviors is that of extension methods in C# 3.0. Those are at the method level. Behaviors are similar, but are live components. More specifically, a behavior is a component that encapsulates some functionality and can be attached to another component to extend its built-in functionality, without creating a derived class.  The fact that behaviors are attached components that implies a composition-like metaphor, offers two advantages whenever derivation is non-essential:   1. Allows "extending" multiple component types - in the TextBox example, the behaviors can be attached not only to TextBox but also to WatermarkTextBox. 2. A single TextBox can be associated with multiple behaviors as needed, rather than requiring the developer to build and use a kitchen-sink style control that has to anticipate every possible feature.   Almost all behaviors handle events and encapsulate a set of event handlers into a reusable component that can then be instanced declaratively. Often this leads to a more declarative style of programming (something I love) and reduced code-behind clutter.  **Example 1**: Lets say you're implementing a typical form with a TextBox and a search Button. You'll probably want to allow users to press the Enter key after typing in some search keywords to kick off the search rather than force them to click the button explicitly. What you'd normally do is write a couple of event handlers to handle both the Click event from the Button and the KeyDown event from the TextBox (to look for the Enter key) in your code-behind, and trigger the search logic from each of them. This is more complicated and messy than it needs to be.  Ideally you simply want to handle a single event - the button's Click event. In fact, in HTML forms, this is built in into the notion of a Form with a submit button. However, it isn't so in WPF, where a Form concept is baked in. This is where this behavior comes in.  **Example 2**: The M-V-VM pattern uses Commands instead of events in many cases, but WPF controls don't trigger Commands in all the interesting cases. Button for example has a Command property, but it only fires when the Button is clicked, not double-clicked or right-clicked. ClickBehavior defines attached properties like DoubleClickCommand of type Command. When you set these properties the behavior registers for the MouseRightButtonUp event and in the handler for that event invokes the Command in the DoubleClickCommand behavior. Interestingly, this same behavior can be used to attach Commands to \*any\* UIElement, even a rectangle. This turns out to be useful for adding interactivity to parts of the UI without the extra cost of a full-blown Control.  **Example 3**: You can do a lot with the Attached Behavior pattern. Drag and drop is an obvious candidate for a behavior. A single DragDropBehavior class can register for all the basic input events and invoke Commands like OnDrop and OnDrag, allowing you to make any visual a DropTarget or DropSource without sub-classing or adding substantial code to your page. | |
|  | Difference between Attached Properties and Attached Behaviors  "Normal" attached properties are there as metadata for some other piece of code (i.e. a container control like a Grid or a ToolTip) to modify its own behavior based on the presence of that attached property. When used like that, attached properties are like attributes in C# or VB code - they don't do any work themselves, they are just there to influence behavior implemented somewhere else.  When an attached property has a change handler that acts on the exposed API of the DependencyObject to which it is attached, it is an attached behavior - in which case it is much more like an extension method in C# - a chunk of code that can be associated with that object to supplement that object's behavior or functionality without that object's knowledge.  Behaviors are a way of supplementing the functionality, or behavior, of XAML elements. A behavior is a chunk of code you write that can be used in XAML by attaching it to some element through attached properties. The behavior can use the exposed API of the element to which it is attached to add functionality to that element or other elements in the visual tree of the view. In this way, they are the XAML equivalent of C# extension methods. Extension methods allow you to define additional methods for some class in C# without modifying the class definition itself, and they work against the exposed API of the class. Behaviors allow you to add functionality to an element by writing that functionality in the behavior class and attaching it to the element as if it was part of the element itself.  In the context of MVVM, they are a great way to bridge from things that are happening in the view due to user interaction and getting the information and execution into a view model.      Different Article        Code Sample - <https://github.com/RobertHedgate/Behavior-lab-xaml>  One more article –  <http://www.japf.fr/2008/08/thinking-in-wpf-attached-properties/>  <http://www.japf.fr/2008/09/thinking-in-wpf-more-attached-properties/> | |
|  | Mediator Pattern  The Mediator class can be used to provide loosely-coupled message-based communication between various entities, such as ViewModel objects.  Creating a class that allows various objects to send messages to each other is trivial. It becomes more challenging when you try to create a reusable class like that which does not leak memory. Since the Mediator object can live indefinitely, but the objects registered to receive messages might have a shorter lifespan, this could easily become a hotspot for memory leaks to occur. In an ideal world, any object that registers itself to receive messages should unregister itself when it no longer needs to receive messages. However, in practice, it’s not always easy to know when an object should unregister itself. The design of the Mediator class needs to take this into consideration.  Example – Messenger class implementation of Josh Smith | |
|  | What is PRISM? What are its advantages of using it in real time applications?  <http://beyondrelational.com/modules/2/blogs/61/posts/11247/what-is-prism.aspx>  <http://www.getproductprice.com/Ads-what-is-prism-its-advantage-and-components.aspx> | |
|  | More Design Pattern Examples  <http://code.msdn.microsoft.com/site/search?f%5B0%5D.Type=User&f%5B0%5D.Value=Srigopal%20Chitrapu>  <http://www.remondo.net/category/patterns-and-practices/> | |
|  | Dependancy Injection Pattern Example  <http://code.msdn.microsoft.com/Dependency-Injection-with-5702acaf> | |
|  | Data Virtualization  <http://stackoverflow.com/questions/1389769/wpf-virtualizingstackpanel-for-increased-performance>  <http://www.zagstudio.com/blog/498#.UVnTchzvsmM> | |
|  |  | |
|  |  | |

|  |  |
| --- | --- |
| WPF Architecture Questions/Localization | |
|  | What is the architecture of WPF?   |  |  | | --- | --- | |  |  |  1. Presentation Framework - Most of the important WPF elements are in Presentation Framework. Holds top level WPF types which includes Window, Controls, Styles, and Layout Panels etc. The code and controls written in WPF Application is mostly interacting with this layer. Also provides high-level services like layout, data binding, command handling. 2. Presentation Core - Holds base types such as UI Element and Visual. Almost all the controls you are directly interacting with are derived from these types. Presentation Framework uses most of the types defined in this layer. It provides .NET API that uses rendering services for the MIL. During graphics programming we work with Core API. 3. Composition Engine - Is also called MIL, Media Integration Layer. It takes Bitmaps, Vectors, and media render them to the DirectX. MIL sits as unmanaged layer to minimize CPU usage. 4. MilCore – Media Integration Library is core rendering system. MIL is unmanaged code. This layer converts WPF elements into the format that Direct3D expects. Windows7 and Windows Vista uses this assembly to render its Desktop. 5. WindowsCodecs– provides supports for imaging like image processing, image displaying and scaling etc. |
|  | What is XAML  - It is Declarative Markup Language.  - An element in XAML is an instance of an object and attributes are properties on that object.    - XAML is not dependent on WPF or WPF is not dependent on XAML. |
|  | What the first class that is loaded by any new WPF Application project?  App.xaml and App.xaml.cs. |
|  | App.xaml  It represents the application itself. It is not visual and is primarily used to store resources that will be used throughout an application. App.xaml also defines which window opens when the application launches. |
|  | When you apply x:Name what happens in the background? How does setting x:Name help you reference the element in the cs page  Because it becomes a field in the generated class within mainwindow.g.cs (auto-generated file by MS which has a partial class and ties with partial class of MainWindow.xaml.cs)  MainWindow.xaml  Mainwindow.g.i.cs (location is under obj folder) |
|  | What tool should you use sketch a mock of your WPF application?  SketchFlow |
|  | What is the Application Lifetime in WPF?  <http://www.dotnetfunda.com/interview/exam4763-what-is-the-application-lifetime-in-wpf.aspx> |
|  | Does XAML file compiled or Parsed?  By default XAML files are compiled ,But we do have options to let it be parsed. |
|  | Where can adorners be used?  Adorners can be used for following scenarios:   * Adding functional handles to a UIElement that enable a user to manipulate the element in some way (resize, rotate, reposition, etc.). * Provide visual feedback to indicate various states, or in response to various events. * Overlay visual decorations on a UIElement. * Visually mask or override part or all of a UIElement. |
|  | What are the different types of brushes that WPF offers?  There are six different types of brushes   * SolidColorBrush * LinearGradientBrush * RadialGradientBrush * DrawingBrush * VisualBrush * ImageBrush |
|  | How to apply tooltip over a disabled element  We have to use the ShowOnDisabled attached property of the ToolTipService class From  XAML and this would look like the following on a Button:- |
|  | How to access XAML objects in behind code |
|  | What are common Localization/Globalization practices for Localizing/Globalizing WPF?  1) Use Resources.resx  2) Use BAML  3) Use a ResourceDictionary to manage localizable strings |
|  | What are some best practices when using WPF if you plan to build a Localizated/Globalizated application?  - Write UI in XAML  - Avoid sizing and positionings, but let objects automatically flow.  - Enable TextWrapping. |
|  | ResourceManager Class  Represents a resource manager that provides convenient access to culture-specific resources at run time.  The ResourceManager class looks up culture-specific resources, provides resource fallback when a localized resource does not exist, and supports resource serialization.  The ResourceManager class retrieves resources from binary resource (.resources) files. Typically, these resource files are embedded in an assembly either by a language compiler or by the Assembly Linker (AL.exe), although the CreateFileBasedResourceManager method allows a ResourceManager object to retrieve resources directly from a .resources file that is not embedded in an assembly. |
|  | Design a Scroll ticker (Marquee control in WPF) |

|  |  |
| --- | --- |
| Additional Concepts | |
|  | CustomaMarkupExtension  Steps to create a custom markup extension   1. Create a class (new cs file from Project) which derives from MaxValueExtension. The naming convention is that the class name should ends with 'Extension' .Eg:TypeDescriptorExtension 2. If we need to accept a default parameter with extension write a constructor 3. Add other properties as well. 4. Override ProvideValue Method and return the appropriate value after processing.     public override object ProvideValue(IServiceProvider serviceProvider)   {  Computational Logic goes here   }  XAML Code Snippet leveraging CustomMarkupExtension |
|  | InputBinding within Commands  WPF also allows us to hook up commands with Input Gestures. It’s an abstract class, and we will be dealing with two types of gestures, namely KeyGesture and MouseGesture. Examples of mouse gestures are left button click, scroll wheel click and key gestures speak for themselves. Both kinds of gestures allow us to also define an optional control modifier, e.g. Control + Left Button Click, or Shift + C key. |
|  | Debugging Tip – set the output type to console application |

|  |  |
| --- | --- |
| Testing Concepts | |
|  | Testing Frameworks - TDD etc  <http://stackoverflow.com/questions/3856837/c-sharp-wpf-testing-strategies>  <http://en.wikipedia.org/wiki/Test-driven_development>  <http://www.codeproject.com/Articles/9715/How-to-Test-Private-and-Protected-methods-in-NET> |
|  | The Microsoft UI Automation Library  There are several techniques you can use to test the user interface of a Windows® application. For example, you can use System.Reflection classes to test Microsoft® .NET Framework applications or you can test both .NET and native applications by calling Win32® API functions such as FindWindow—either with unmanaged C++ or with C# or Visual Basic® by using the P/Invoke mechanism.  We get started with UI test automation using the new Microsoft UI Automation library, which is included in the .NET Framework 3.0 as part of Windows Presentation Foundation (WPF). You can think of it as a successor to the Microsoft Active Accessibility (MSAA) library, which was originally designed only for accessibility features but was somewhat coincidentally found to be useful for and adapted to UI automation.  <http://msdn.microsoft.com/en-us/magazine/cc163288.aspx> |
|  |  |
|  |  |