Windows Phone 7.1: a Quick Start

Introduction 1

Target audience 2

Pre-requisites 2

References 2

Overview 2

Hardware 2

API 2

GUI 3

Silverlight 3

Application Startup 3

The XAMLs 4

GUI Building Blocks 6

Localization 12

Random Notes 12

Application design 12

FAQ 13

Third Party 14

IDE Setup 15

Tips and Tricks 15

Final Words 16

Introduction

When I started to write this article, I thought it'll be 3-4 pages at most. However, time shown that I severely underestimated the amount of the information, so the article became much longer than I initially expected.

I'm writing this article because I was unable to find a high-level overview of the WP7 development. There're some good "how to do this specific small thing" articles. The reference documentation is good as well. However, the higher-level concepts are hidden in various video podcasts and records from conferences. And I hate watching video instead of reading text.

Usually, the backend code of a WP7 application is normal .NET, with very few phone-specific features. This whole article however is phone-specific, that's why the primary focus is on the rich UI, and the UI logic.

Target audience

Developers new to Windows Phone and Silverlight.

Pre-requisites

Descent PC (at least 3 GB RAM, at least DirectX 10 class GPU, at least Vista SP2), Visual Studio 2010, Windows Phone 7.1 SDK.

If you want to test on your physical device, developer account (priced 100 USD / year) is required.

References

MSDN library.

Charles Petzold, programming windows phone 7.

Wikipedia, Google, StackOverflow.com, bunch of standalone blogs posts.

Overview

Hardware

Here's the complete list of the devices on the market.

As you can see, all devices are single-core, with frequency ranging from 1 to 1.4GHz.

There're 256 to 512 MB RAM, an application must not exceed 90 MB of RAM usage (on devices with more than 256 MB, this requirement is relaxed).

All screens are 480x800 pixels. There're LCD models (i.e. the white LED backlight is illuminating the RGB sub-pixels), there're OLED models (i.e. the sub-pixels are self-illuminated). The main difference between the two — different color rendition. And by the way: by default, an application is using 16-bits color mode R5G6B5, so if you need many colors in your app, you should configure 32-bits color in the application manifest XML.

There're also optional compass and gyroscope, at least 1 camera, and mandatory GPS.

API

Common services

WP7 applications are managed. They are executed by the special ARM version of the .NET runtime, running special version of compact framework.

Such things as math, text and XML manipulation, events, delegates, regular expressions, threads, synchronization primitives, CLR thread pool, and LINQ are the same.

Collections are changed only slightly compared to the desktop .NET. For example, there's no HashSet but it can be trivially implemented given the Dictionary is here.

Some exception classes removed, e.g. ApplicationException.

Compact SQL server works OK, it uses the isolated storage (mentioned in the next paragraph) to hold your data. Entity framework works OK. I've used code-first approach, with records/columns are marked with attributes from the two *.Data.Linq.Mapping namespaces (where "*" stands for Microsoft.Phone and System.Data). I have only tested under the small load, though. But after all, this is a phone in your pocket, not a ProLiant.

It's possible to use Async CTP. I'd recommend to consider it, if your application does some background processing, e.g. talking to some other system over the Internets.

Phone-Specific Services

The lifecycle of the applications is somewhat strange. Before you'll start to design the dataflow between various parts of your app, be sure to read about the concept of "tombstoning".

The writeable area of phone's file system is wrapped deep inside of the "Isolated storage". The storage files implements the IStream interface, so most readers/writers/serializers are available.

A phone has compass and GPS, and of course there's an API for that. The compass is a bit tricky to use because the system sometimes sends you a message saying "Achtung! The compass needs calibration" that you must handle properly by displaying the UI for that.

GUI

There're several ways to write GUI for WP7.

For 3D games, there's XNA. There's also HTML5/JavaScript. But for the application I'm developing (a bank client), the Silverlight was the obvious choice.

The rest of this article will focus on the Silverlight technology.

The framework is a subset of Silverlight 4. The offline documentation is available. The documentation features a zillion of small blue phone icons meaning "this method/property/event is supported on a phone". Unfortunately, it lacks the phone-only part of the framework, which is documented in another CHM, "Windows Phone Developer Tools Documentation".

If you're familiar with the Silverlight, it helps a lot. Before now, I only used Silverlight for 3 days, few years ago, to complete an evaluation task while searching for a new job.

Silverlight

Application Startup

Just like with desktop version of the Silverlight, after you've successfully build your application, you'll get a XAP file. The file is actually a ZIP file. If you'll unpack it, you'll see a bunch of DLLs, XML and resource files.

One of the DLL files (the one specified as the "EntryPointAssembly" in the AppManifest.xaml file) is the main executing assembly of your application, containing the entry points, and usually large part of the resources.

In short, the Silverlight runtime reads the AppManifest.xaml and WMAppManifest.xml files, loads the entry point assembly, and creates the instance of your application class (inherited from System.Windows.Application).

The XAMLs

Being a GUI framework, Silverlight provides a way to define the GUI using a special markup language (XML subset) called XAML. You should know (to some extent) basic concepts of XML, especially the namespaces.

Besides providing the runtime environment, there are also 2 completely different design-time environments. One is Visual Studio 2010. Another one is Microsoft Expression Blend 4.

Visual studio allows you to develop code, write raw XML, it also has basic WYSIWYG editor. Blend allows you to edit the same XML visually, in a sophisticated WYSIWYG environment. I use both for different tasks, and alt+tab frequently between them.

In runtime, the XAMLs of your applications are combined into the visual tree, containing the controls visible on the screen. All nodes of that visual tree are the instances of classes, usually inherited from some system-provided framework elements. The direct analogy of visual tree is the windows hierarchy in Win32, as visualized by Spy++ tool. Unlike Win32, there's no such things as WM_PAINT or custom drawing: every pixel visible on the screen comes from some framework element[s] (if you want to, it could be an image displaying the raster bitmap you've generated programmatically, but beware the CPU load required to generate a bitmap).

XAML Bindings

Silverlight provides different mechanisms to communicate between the XAMLs and the code.

Partial Class a.k.a. Code-Behind

The XAMLs containing pages and user controls are compiled producing a partial class. In the XAML, the elements having the x:name attribute are compiled to the internal fields of your page / user control inherited class. So you can define in xaml a text block with "x:name=tbMessage", and then in the code of your page/control write this.tbMessage.Text = "Hello".

You can also bind controls-generated events to the event handlers, and process those events by the code behind class (i.e. by the other half of that partial class).

I have a feeling that the code-behind was created mostly to make it easier for developers to migrate from WinForms to Silverlight. Some advanced Silverlight's features are only available when you do data binding.

Data Binding

Data binding is general technique that binds two data/information sources together and maintains synchronization of data. In the context of the WP7, the data binding is a mechanism to bind the properties of the visual tree (instantiated from a bunch of XAML resource files) to the underlying classes.

Data bindings can be one way (i.e. when the data is travelling from the underlying data to the UI) or two-way (when the data travels both directions so that the underlying data source receives changes generated by the user input).

Data bindings can be one time only, or the visual tree may subscribe to change notifications, and update itself in real-time as soon as the data changes.

The best part is, with this approach it's possible to provide design-time data for the Expression Blend application.

On the model's side, data binding requires public properties (fields won't work). If you want two ways binding, your property must contain public setter. And if you want the GUI to subscribe for change notifications, you must implement an interface for that, INotifyPropertyChanged.

On the XAMLs side, data binding requires a dependency properties attached to a framework element. Most often you'll bind to the system-provided dependency properties, such as "Text" property of class TextBlock, and "Visibility" of everything. In some cases however (while creating controls, behaviors or triggers), you'll want to define your own dependency properties.

Change Notifications

Besides INotifyPropertyChanged interface, there're other ways to notify the GUI that you've modified the data. If you're visualizing a list of items, it's sometimes a good idea not change the entire list, but use ObservableCollection (or a derived class); so that you can add/remove items and the GUI will receive more specific notifications about that.

Beware of the threading. Visual tree can only be modified from the main (GUI) thread, so it's a good idea to marshal the notifications. You can do it in some base ViewModel class:

```
/// <summary>If the current thread is a GUI thread, execute the action right away.
/// If however it's some other thread, schedule the action to be executed by the GUI thread
and immediately return.</summary>
/// <param name="act">The action to execute.</param>
public static void startOnGuiThread( Action act )
{
	var disp = Deployment.Current.Dispatcher;
	if( disp.CheckAccess() )
		act();
	else
		disp.BeginInvoke( act );
}

protected override void RaisePropertyChanged( string propertyName )
{
	Global.startOnGuiThread( () => base.RaisePropertyChanged( propertyName ) );
}
```

Dependency Properties

If you want your custom/user control to have a property that participates in data binding, styling, and/or animation, you need to create not just a property, but a dependency property.

Here's more information "why", and here's more information "how".

Binding Converters

Sometimes you need to display not the raw value from the data source, but rather the value converted somehow. For example, in the data source you have a field of type "enum eSex { Unknown, Male, Female }",

and want to show one of the 3 icons '?', '♂' or '♀'. Or you want to show some part of the UI depending on whether some property is set.

For such cases, you can write a converter (which is a public class implementing System.Windows.Data.IValueConverter interface), and specify it in the data binding string.

GUI Building Blocks

The Silverlight is a huge framework. In this section, I'll try to write a brief overview of what's in there. For the detailed information, read the "Programming Windows Phone 7" book by Charles Petzold.

Frame

The application frame is the one and the only root of your application's visual tree, the outermost container. The frame is set up in app.xaml.cs file, in the constructor.

Page

A page represents the screen of the content. Only one page can be visible at the single time (however the system doesn't kill the old pages, so with some easy tweaks you can have translucent pages). The runtime provides a way to navigate between pages and also maintains the stack of the recently visited pages, to process the hardware back button.

A page has a XAML file, and a code-behind CS file.

A phone application has the default page, i.e. the page that is opened upon startup. The default page is specified in the WMAppManifest.xml configuration file.

I suggest you to read the article "Application Page Model for Windows Phone" for more high-level information on pages and navigation.

User Control

A user control is a piece of the visual tree implemented in the separate XAML. I use them in the following two cases:

- For the pieces of the UI that should appear on more than one page and/or in several instances on the single page.
- Just to break the overly complex UI and/or UI logic across several XAMLs.

There's good support for creating user controls in Blend.

Just like a page, a user control has a XAML part with the markup describing the visual tree, and partial code-behind class.

Custom Control

A custom control differs from user control in two ways:

- It has no XAML part. However, you should define the control template for similar effect.
- It can be styled and themed.
- It can host arbitrary content inside.

    - If it's derived from ItemsControl, it can be used to represent a collection of items. Framework-provided ListBox and MapItemsControl are an example of such controls.
    - If it's derived from ContentControl, it can be used to host an arbitrary XAML content inside. Framework-provided Border and Button are an example of such controls.
- It has no code behind. You only have your single class implementing a control, with no compiler-generated part.

I have created a few ContentControls to host heterogeneous set of items in a consistent manner. It's sometimes a good decision to define new dependency properties of that controls.

For example, let's say you're creating an app that sells pizzas and burgers. So, in your model you have:

```
abstract class ItemOrder
{
        public string name { get; }
        public int Count { get; set; }
}

class PizzaOrder: ItemOrder
{
        float diameter { get; set; }
}

class BurgerOrder: ItemOrder
{
        float weight { get; set; }
        bool hasMustard { get; set; }
}
```

And you want to create two controls to display pizzas and burgers. You want the controls to have a common border and title (that includes the name label, and count control), but different content inside (diameter slider for pizza, and some other controls for burger).

For this case, you may create a custom ContentControl, named "ItemOrderControl" with two custom dependency properties, name and count. Then you can use it in the following way (for the pizza):

```
<local:ItemOrderControl name="{Binding name}" count="{Binding Count, Mode=TwoWay}">
        <StackPanel>
                <TextBlock Text="The diameter in inches:" />
                <Slider Value="{Binding diameter, Mode=TwoWay}" Minimum="7" Maximum="25" />
        </StackPanel>
</local:ItemOrderControl>
```

With this approach, you can reuse the frame (with name label and count editor) with many types of the enclosed content. Here's the detailed article on creating custom content controls.

Custom controls take more time to develop then e.g. user controls. For example, for the above example where you only have 2 simple items, I wouldn't bother creating a custom content control, instead I'd just defined styles for border, name and count, and copy-pasted the XAML. Only create them for the containers that you'll reuse a lot across your application.

Various Templates

Control Template

The system controls are customizable via styling; the mechanism is covered in the next section. In short, you can set up basic properties (like fonts and colors) in style.

However, sometimes you'll want to change visual properties that are not exposed as the control properties. For example, by default buttons has 12 px padding that comes from system resource of type Thickness, named "PhoneTouchTargetOverhang", which is not always desired.

In another cases you might want to replace the look of the control completely. For example, in my application I have checkboxes that look like "outlined heart shape" unchecked and "filled heart" when it's checked.

Unlike competing platforms (I'm looking at you, Apple), in WP7 it's possible to customize *every* visual aspect of the system controls, by providing what is called "control template". Visual states are also a part of the control template, so you can even design animated transitions between them. For example, when my customized checkbox is [un]checked, the heart shape is filled by [un]blending the filled image for 0.11 seconds.

Don't create control templates by editing XAMLs in visual studio. Instead, right click on the control in Expression Blend, click "Edit Template", then "Edit a copy…" The Blend will then take a framework-provided copy of the control template, and paste it into the resource dictionary of you choice.

Content Template

Content template is a template for the content controls, which allows you to visualize your view model objects by feeding their properties into the appropriate places in the XAML. I seldom use content templates outside of the item controls.

Item Template

There're several system provided controls that can display the collections of your items. For example, an ItemControl displays your items in the stack panel, ListBox does the same but allows used to select items, while MapItemsControl displays your items on the top of the map.

To use them, you provide a collection of items to visualize (an ObservableCollection if your collection changes dynamically and you want to reflect those changes in UI), and you also provide an item template. An item template is a content template for individual item.

List Box Templates

So, for list box you'll have 2 templates that define the look of your items:

1. Control template for item, you provide it in ItemContainerStyle attribute for list box. In this template, you can specify different appearance based on item's visual state. However, you have no access to the data item. The container is the outermost part of an item.
2. Item template. In this template, you can access the properties of the items of the data source, however you don't have access to the visual states. This template defines what's inside of the item.

You ask “What if I want to customize container properties based on the item?” The answer is — it’s fairly easy. Create a class that derives from a ListBox, and override a single method called PrepareContainerForItemOverride. When this method is called by the framework, you have both the container, and the data item, and you can do whatever you want with their properties. Don’t forget to call the base class implementation.

And by the way, I found it’s a good idea to place ItemContainerStyle in a globally-available resource dictionary, and reuse them across various list boxes in your application. See the “Resource Dictionaries” section below for more details about that.

Styles

Styling is another powerful technique in the Silverlight. Use styles and/or other globally defined resources every time you want your controls to look similar on different pages/screens.

For example, in the application I’m developing the designer painted some buttons in orange, and scattered them across the application’s UI. Of course, I’ve defined the style for orange buttons, and reuse it heavily across the whole application.

A style can inherit from another style, overriding some properties.

A styles can be defined literally anywhere: globally, in separate XAML (see the next section “Resource Dictionaries”), or in the XAMLs with controls.

A style can be named, or unnamed. Unnamed styles are applied to all elements of some type that has no style specified. I usually style frequently-used elements, such as text boxes, using the following approach:

- I create a style named “textboxBase” with basic setters such as font family.
- I create an unnamed style for text boxes, based on textboxBase style, with no setters at all.
- This way, default style is applied automatically to all textboxes, and I can create further styles based on the default one. Moreover, in specific page I can create unnamed style for them (that applies automatically to all text boxes on that page) that’s based on the textboxBase style.

There’s good support of styling in Expression Blend, e.g. applying an existing style to a control is just a few mouse clicks (the first one is right click). However, I’ve found it’s easier to edit styles in XML, using the Visual Studio.

Resource Dictionaries

This is probably the simplest concept in the Silverlight. Basically, a resource dictionary is just a Dictionary<string, object>, with values declared in the XAML files. You can put styles there, control templates, brushes, images, data templates, and your own custom objects.

The resource dictionary in app.xaml is globally available to all pages of your application. For example, in the application I’m developing, on different screens there’re several translucent panels. To make them be of the same color, I’ve defined a SolidColorBrush resource in my resource dictionary, and use it as the backgrounds of the translucent panels. The panels are of different types — grid, border, etc., so I can’t just define a single style for all of them.

Unless you manually copy-pasting your resources across pages (you really shouldn't), soon you'll notice the app.xaml file grows bug and unmanageable. Fortunately, it's possible to include an external resource dictionary XAML in the app.xaml, using the `<ResourceDictionary.MergedDictionaries>` element. In my application, I have 3 resource dictionaries included in the app.xaml:

1. BaseControlStyles.xaml — styles for basic controls such as buttons and text boxes.
2. ListStyles.xaml — styles and templates for list boxes and list box items.
3. Converters.xaml — a bunch of my own small classes implementing IValueConverter interface. I decided that since they mostly relate to presentation logic not appearance, they deserve their own resource dictionary.

System Controls

The framework provides some system-implemented controls. I won't describe individual controls here, since there's a lot of information on the internet.

I'd like to note however, that look and feel of the system controls can be changed rather radically by using styling and tempting. For example, here's the article describing how to make checkbox to look like a toggle switch, with the correct appearance and animations.

Animations and Visual States

Visual state is common property of a view model.

For example, if you're developing a client-server application, you may have 3 distinct states of a view model: loading, normal, and failed. While it's loading, you want user to see a progress indicator. While it's normal, you want to present your data. And it the operation failed, you want the user to see an error message, possibly with the "retry" button.

Of course, you could write some code somewhere (in code behind of view model) to manually show/hide parts of the GUI. But then your designer shows up, and says "I want this panel to be colored gray while it's loading, and I want the error panel to slide from the right edge of the screen".

Luckily, the Silverlight has a feature called "Visual state manager", which allows you to define visual states in the XAML markup, and even animate the changes.

Here's how I usually manage visual states of my pages and user controls:

1. Copy-paste the code implementing attached behavior called "Visual State" from stackoverflow. It allows you to databind to the current visual state.
2. Inside the view model class, define the enumeration with the possible visual states.

```
public enum VisualState
{
        stateNormal,
        statePanelExpanded,
};
```

3. In your view model, create a property "visualState" of type `VisualState`, that fires a PropertyChanged event when modified (use inpc snippet to save typing).
4. Use Blend to create the visual states, name them exactly like your `VisualState` members. Set up state-specific visual changes and animations using all power of the sophisticated editor.

5. In your XAML, write `VisualStates.CurrentState="{Binding visualState}"` on the control that has the visual states defined.

Behaviours

Behaviour is a new way of adding interactivity without writing code.

When I did my first demo applications in WP 7.1, I've used code behind to handle events.

Then when I started to use MVVM, the logic "we clicked the button, should now do something" moved to the view model class.

The good things about the behaviors is that for many cases, that logic has already been written by, and being supported by, Microsoft. So you don't have to. For example, by using the NavigateToPage action, you can implement navigation across pages with no code at all.

It's also possible (and sometimes worth it) to write custom behaviours. I'll give you an example.

In the application I'm developing there're many different kinds of delay-loaded items, with completely different data and appearance, which share the same typical usage scenario:

1. User sees an item with some information.
2. On click/select/whatever, application issues a network call, and presents a progress bar under the item.
3. If the network call failed, user sees an error message. If it succeeded, user is taken to another page presenting the data that has been downloaded.

Here's how I approached the problem:

- An abstract base class for the view model called "DelayLoadingViewModel". It has public Load() method, and protected abstract Task loadImpl() method. It also exposes the visualState dependency property, of type enum {notFetched, requestSent, failed, fetched}.
- Custom content control, with visualState dependency property of the same type. In the template, I have a progress bar (initially hidden), and 4 visual states, e.g. for requestSent state a progress bar is visible, and the content opacity is set to 50%.
- Custom trigger, with 2 dependency properties: the view model object to fetch (of type DelayLoadingViewModel), and the URI to navigate on success. If the network call fails, it displays a message box. If succeeds, it navigates to the XAML page specified in the URI. Later, I've added the 3-rd property, "forceReload" of type bool, to force network call even when the object is already in the fetched state.

After I've created those 3 building blocks, it became much easier to develop the rest. Both UI and logic that implement the delay-loaded mechanism is reusable, and is separated from both the underlying data source (non-abstract child of the DelayLoadingViewModel class implementing the loadImpl method to actually load something), the UI (content created in Blend residing inside my <local:DelayLoadedPanel visualState="{Binding visualState}"> content control), and the interactions (event triggers created in Blend that uses my <local:DelayLoadingAction> trigger to start the loading process).

Custom behaviours usually take more time to develop than code the same logic in e.g. code behind. But once you're done, attaching behaviour to a control is a matter of 2 clicks. Just like the custom controls, only develop a custom behavior for the UI logic you're going to reuse a lot across your application, otherwise the time you've spent on them won't pay up.

Localization

Unfortunately, the localization support is rather poor.

- Localizing an application tile and application name requires building an unmanaged resource-only DLL, here's the guide.
- To setup which languages are supported by your application, you have to manually modify the SupportedCultures element in your .csproj file, the IDE doesn't support that.
- You can use localized strings in your XAMLs, however you'll have to type those long binding strings manually: there's no support for using localized strings in Blend. I found easier to expose read-only properties with localized strings from my view model: in this case the IntelliSense helps typing those IDs.
- I had problem with date picker control (from the Silverlight Toolkit third-party library) full screen page, here's the solution that worked for me (scroll to the comment posted by Scordo).

Random Notes

Application design

I hope, by now you have some information on the features available to you.

I know the amount of the information might be overwhelming. And unlike e.g. iOS, Microsoft provides almost no default look, and gives you the freedom to choose approaches to different problems. In this section, I'll try to give my answers to some higher-level design problems I've encountered.

Code-behind or MVVM?

Here's the opinion I mostly share.

I prefer MVVM when it fits well.

Fortunately, the two approaches can co-exist within the same application just fine. And sometimes combining code-behind with MVVM works best.

For example, on some screens of the application there's a Bing map, displaying items from either local DB or a web service. A translucent panel with the details of the item slides into view when user taps an item.

Here's what I did:

- My MapControl is not data bound. It has 2 visual states, normal and details. In the details state, the panel is visible.
- Item details are implemented using a separate user control, ItemDetailsControl, which is data bound. This way I can design it in Blend.

- In the code behind of MapControl, I have a couple of event handlers, pushpin_Tap and map_Tap. In the handlers, I switch the visual state, and assign the DataContext of the ItemDetailsControl instance.

This way I'm enjoying both the benefits of MVVM, and simple event handling of the code behind.

With pure MVVM, I wouldn't be able to handle the events properly, because:

1. They come from different objects.
2. In my pushpin_Tap I set the e.Handled = true; otherwise the same tap event will also be received in the map_Tap and will cause the panel to hide. Such low-level event handling is unavailable in MVVM.

Static items or items control?

Depends on how similar are the items you're displaying, and how many of them.

If you have many items, and/or the count of items is unknown in design time, and/or all items looks completely similar and can be visualized with a single item template without even a data template selector — use an item control.

If you only have a few items mostly known at a design time — it's usually easier to design them in Blend, by styling and copy-pasting. It's sometimes easier to show/collapse a few panels by using properties of your view model, than create a data bound items control.

Animations: VSM or code?

VSM. Maybe the first 1-2 times you're using the VSM you'll find it hard: changing the visual states can be tricky sometimes, and you'll initially spend some time learning how to edit them in Blend. After you'll get used to the Blend's UI, and create a few utility classes to help you trigger the state transitions, you'll discover the VSM is a huge time saver. Especially on the later stage of the project, when you have to change something you've did before. **NB!** To simplify debugging, always check for return value from `ExtendedVisualStateManager.GoToElementState` call.

Note on User Controls

In OO design, there's a well-known "god object" anti-pattern. In WP7 application design, it's easy to start creating "god user controls": overly complex user controls, with different display modes. Don't do that. Keep them simple and blendable, don't be ashamed to copy-paste a few lines of XAML when you need to, and use global resource dictionaries with styles and templates to maintain the consistent look and feel across them.

FAQ

Q. The ListBox items don't occupy the whole width like I want to. What to do?

A. There're several places to check. First one is item container style. Another one, if you're using a custom data template selector, you must set `HorizontalContentAlignment="Stretch"` on your DataTemplateSelector XAML element. BTW, the "XAML Spy" software can resolve such questions really fast.

Q. After I've moved the animations from a page to a resource dictionary, the project can't be built saying "property 'IsOptimized' was not found in type 'ColorAnimation'".

A. Add the line to your resource dictionary XAML:
```
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```
Q. I did that, now the application crashes at startup saying "XamlParseException occurred in System.Windows.dll: The property 'IsOptimized' was not found in type 'ColorAnimation'". Blend works OK.

A. Add two more lines to your resource dictionary XAML:
```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
```
Q. After I did something, I now receive an exception on page load, from deep inside the system DLL, with lots of "FrameworkElement.MeasureOverride" methods on a call stack. It happens both in runtime and Blend. There's no single MeasureOverride in my code. WTF?

A. Most of the times, I've eventually discovered there're errors in my binding strings syntax. Sometimes I tried to bind to an unbendable property, other times I tried to use unsupported types of bindings. To troubleshoot that, start commenting out the elements that contain attrib="{Binding …", attrib="{TemplateBinding …", etc.. Remember the XAML is a subset of XML, so the comments "`<!-- <Something /> -->`" work OK.

Q. I have a view model for a data bound list box, with selectedItem property bound to a property of my VM. In the setter, I open a new page. After I press back, I can't select the same item once more.

A. The behavior is by design, because the item stays selected. In the setter, you can revert the selected item to null, and raise a property changed event.

Q. I have a view model for a data bound list box, with selectedItem property bound to a property of my VM. But this time I want the selection to persist. After I click an item, it becomes selected. However, I'm unable to set the initially selected item of the list.

A. ListBoxItem has 3 states in the SelectionStates group: Unselected, Selected, and SelectedUnfocused. After you click on an item, it becomes Selected. However when you've set the initial selectedItem through data binding, the visual state becomes SelectedUnfocused. Ensure your SelectedUnfocused visual state is the same as the Selected state.

Q. I'm trying to implement visual states as you suggest, however I don't see any state changes, and the `ExtendedVisualStateManager.GoToElementState` method returns false.

A. Ensure you're raising PropertyChanged event when your visualState property changes. And, make sure you have the following 3 lines in your XAML with visual states:
```
<VisualStateManager.CustomVisualStateManager>
       <ec:ExtendedVisualStateManager/>
</VisualStateManager.CustomVisualStateManager>
```

Third Party

I'm using the following ones: Silverlight Toolkit, Sharp Zlib, Image Tools, Newtonsoft JSON, MVVMLight.

Beware the quality of the Silverlight Toolkit code is somewhat worse than the quality of the system-provided parts of the SDK. Be prepared to spend some extra time on debug/troubleshooting. There're also problems with globalization of those controls. It's no surprise the toolkit is not included into the official Microsoft's SDK.

IDE Setup

Snippets and Templates

One drawback of using MVVM light, you'll often find yourself writing repetitive code, or making same changes to different newly-created classes/XAML pages.

To address the first problem, I recommend getting familiar with the "code snippets" feature of the Visual Studio IDE. By "getting familiar with" I mean not only using, but also modifying. The most frequently used is "inpc" snippet, I've tuned the snippets provided with MVVM light to fit my coding style.

To address the second problem, I recommend changing the new item templates.

I've made a ZIP file with my IDE customizations; see the readme.txt inside for more verbose description, as well as installation instruction.

Other Settings

In VS 2010, disable "Edit and Continue" (Tools/Options, Debugging/Edit and Continue, uncheck "Enable Edit and Continue"). It's counter-intuitive, but since WP7 doesn't support E&C, when E&C turned on, it prevents the code to be edited at all while the debugger is attached. I find it convenient to fix small bugs while looking at them on the simulator, which is usually located on my second display. Before I turned off E&C, I often tried to edit read only source code files. Unfortunately, the XAML files are still read-only while the debugger is active. Related: visual studio's bug ID=534915.

By default, VS 2010 loads XAML files in WYSIWYG editor. The Expression Blend is much better WYSIWYG editor, you'll mostly open them in visual studio to edit XAML source. Besides, visual studio's editor opens too slowly. To fix, VS 2010 project explorer, right click on a XAML file, choose "Open With…", then select "Source Code (Text) Editor", and press "Set as Default" button.

Tips and Tricks

Designing a View Model

When creating a view model, you don't need to raise property changed for the properties that don't change while the view is visible. The "inpc" snippet helps you save the typing; however the less code you have to support is better, and unneeded notifications are sometimes bad for performance.

If some property changes really often (e.g. updated by the compass many times a second), create a static readonly PropertyChangedEventArgs instance to send when this property changes, doing so will save you many memory allocations.

Don't expose commands in your view model. Creating a property of type RelayCommand requires you to write some setup code. Instead, you can apply and setup CallMethodAction built-in behavior with a few mouse clicks. This way, your view model only needs to implement a public method to handle the event.

Other

Resource access is sometimes tricky. To reference a resource, you'll need to include it in your project, and specify a special URL in either XAML or code. You must also choose the correct "Build Action". In short – in most cases, the correct one is "Content" for images and other resource files, "page" for XAMLs with UI, and

“resource” for XAMLs with resource dictionaries. Beware: the URL to access the resource will vary based on the build action.

If you’re troubleshooting some GUI-related issue on a complex screen, and can’t figure out e.g. “why the hell is this thing centered instead of being stretched” – download the demo of http://xamlspy.com/ (I ended up buying it, BTW). This excellent tool can help you really quick. It’s an equivalent of Spy++ application for the Win32 platform, but for the Silverlight. It even allows you to edit properties of the visual tree elements in runtime, and instantly see the changes.

Speaking about GUI issues; if you’ve made some GUI-related changes in visual studio, switched to blend, pressed “Build Project” (Ctrl+Shift+B) and not seeing any differences — there's a chance the problem is not on your side. Launch your app on emulator or device to verify. If it’s the Blend who’s wrong, quit the expression blend and launch it again. Rebuilding or reopening the project doesn’t help.

Copy-paste control templates from the Internet with care. There’re different versions of WP7 and Toolkit, the default control templates change with versions. Not just controls. It’s possible that some source code you’ll google is no longer relevant because the functionality you need is now a part of the SDK.

When implementing an IValueConverter interface, ignore the last “CultureInfo culture” argument, in most cases it has nothing with the current localization. Use current culture instead.

Unfortunately, Expression Blend doesn’t support editing of the custom control templates. There’s a workaround: right click on the control, choose “Edit template / Edit a copy”, edit the template however you like, then move the control template’s XAML back to the “Themes/generic.xaml” to update the control’s appearance on another pages.

As I said at the beginning of the article, the developer account costs you 100 USD per year. There’s a gotcha however. If you’re an individual, it takes 10 minutes to get yourself an account. If however you’re representing some legal entity and would like to register a corporate account, the paperwork could easily take weeks instead of minutes. Needless to say, before it’s done, you won’t be able to test your application on the actual hardware.

If your application uses bing maps silverlight control, when submitting your application to the marketplace, make sure you’ve **cleared** the checkbox “China” under the “the countries where your application is available” section. Otherwise your application won’t be allowed to the marketplace.

If your application uses compass, it’s a good idea to stabilize the compass reading by averaging a dozen of recent compass readings. To properly average a set of angles, read this note, and use `Math.Atan2` API.

I have a decent computer, a laptop with second generation Core i5 CPU and 8GB RAM. However, Visual Studio 2010 & Blend 4 both load storage subsystem with random read/write requests. Eventually I’ve bought a fast SSD drive, it helped a lot.

Final Words

I hope, by now you have some image of how to get started developing a rich UI application for Windows Phone 7.

I started this article in August 2012. Now it's November, and Windows Phone 8 is out. However, I think this article is still relevant, due to the following 2 reasons:

1. The WP8 is backward compatible with Windows Phone 7. As far as I know, Microsoft is planning to keep WP7 on the market for some time, as a platform for mid-range phones.
2. I didn't yet look at the WP8 SDK. However I've heard that while the underlying API has changed from Silverlight to WinRT, the higher-level functionality (like the XAMLs, MVVM and Blend) stays more or less the same.