Bethany Beery

CIT275 – Instructor Jeff Straw

Northwestern Michigan College

SELF DIRECTED PROJECT

Ever since I purchased a new lap top with Windows 8 I have been curious about Window Store Apps. What does the code behind look like? What was used for the touch capabilities? How does development differ from WinForms and WPF? I choose to follow my curiosity and teach myself how to create a Window Store Application as my self-directed project.

With Bob Tabor on Channel 9 as my guide I attempted my first cookbook application by going through the Window Store apps for Absolute Beginners with C# video series. However, I discovered that the series was written for Windows 8.0. Changes to code functionality and syntax in Windows 8.1 make the demo code no longer useable. Layout Aware Page and the Callisto control suite have been depreciated so the code will not compile.

**THE BIG PICTURE - GETTING STARTED**

**Languages**

An amazing feature Microsoft offers with developing an application as a Windows Store Application is the ability to code in a variety of different languages with the same end result. Window Store Applications run on top of a new API called WinRT. Introduced in Windows 8, WinRT is a set of object oriented APIs that provide a shift in Windows development. One can pick from Visual C#, Visual Basic, JavaScript, or Visual C++ for the code behind language and XAML or HTML as the corresponding graphical user interface language. Note that only JavaScript uses HTML. For my research I have chosen to focus on applications written in my favorite language, Visual C#.

**Types of application**

There are three types of Window Store Applications. A Universal app is created for both Windows and Windows Phone platforms, Windows targets Windows desktops and tablets, and Windows Phone Apps will only run on Window phones. Visual Studio makes it relatively easy to extend your project to both platforms later on should you decide to include both versions.

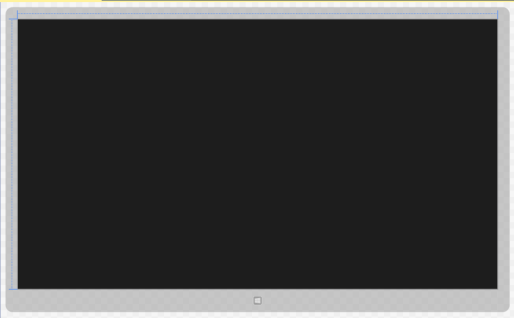
**User Experience**

Window Store applications have a different user experience than traditional desktop applications. Desktop application navigation has been reliant on an always present menu bar. Window Store applications have the option to hide navigation until a user needs it. There are multiple new ways to navigate through the application, such as horizontal and vertical swipes, app bars and charms. Interaction is also different in that event are integrated into the app page instead of using modal dialogs like a message box.

**Templates**

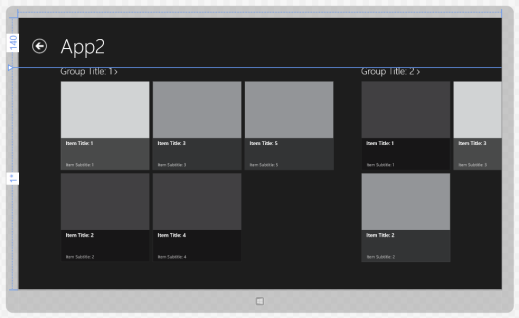
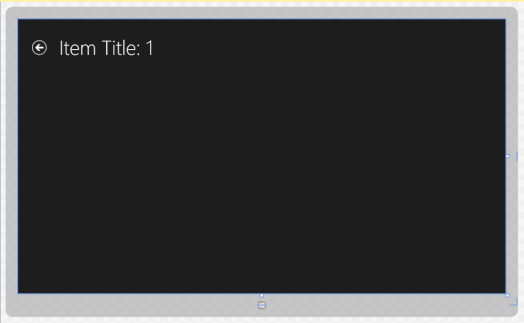
Each language has a selection of templates to choose from as a starting point to make development easier, faster, and visually consistent. The selection of templates varies slightly from language to language however, they have three in common; blank app, grid app, and slip app. The blank app template presents the designer with one page and an empty page as the name would suggest. Grid app template is more complex with three pages and a group of data in grid format.

**Blank Template**



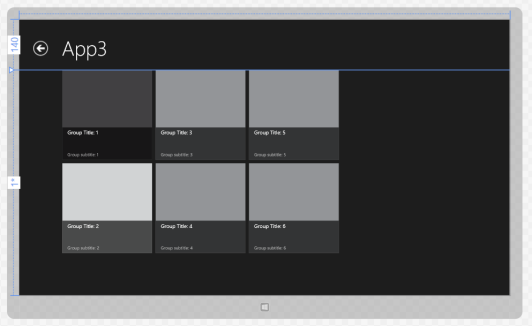
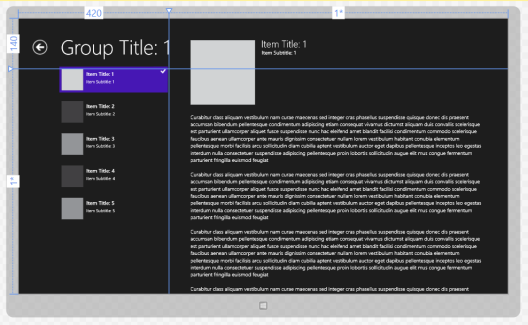
MainPage.xaml

**Grid Template**

GroupedItemsPage.xaml GroupDetailPage.xaml ItemDetailPage.xaml

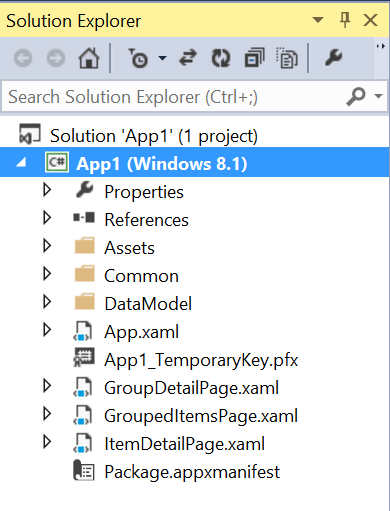
**Split Template**

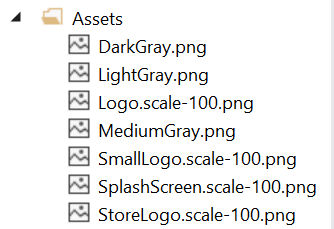
ItemsPage.xaml Split Template.xaml

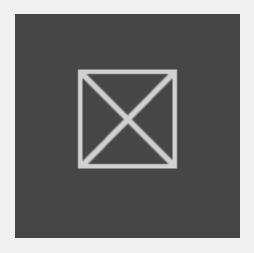
## STRUCTURE OF A NEWLY CREATED WINDOWS STORE APP -> GRID APP PROJECT

**Assets Folder**

 Contains png image files for the tile display of your app on the start screen

and the image displayed when the app launches – the splash screen.

 Default logo appearance



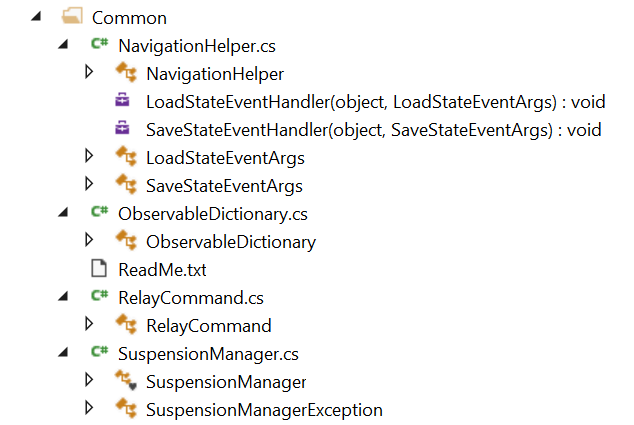
**Common Folder**

Contains 4 classes, NavigationHelper.cs, ObservableDictionary.cs,

RelayCommand.cs, and SuspensionManager.cs, to support the templates.

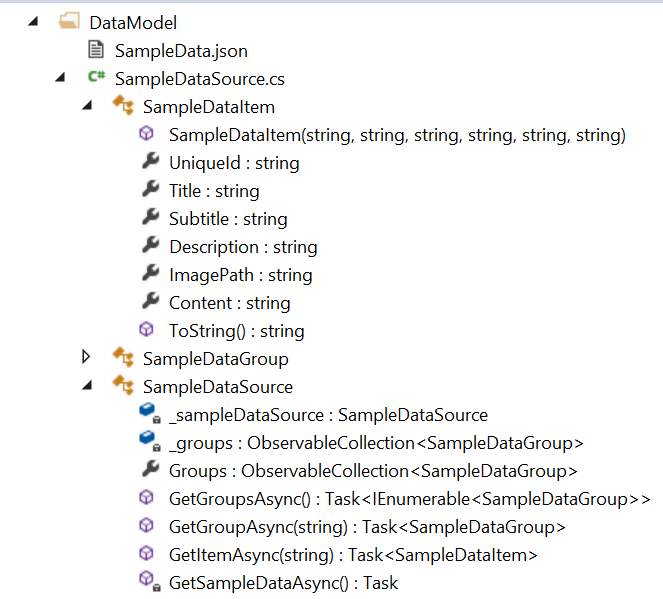
The ReadMe text file contains a warning that chaning the contents of the

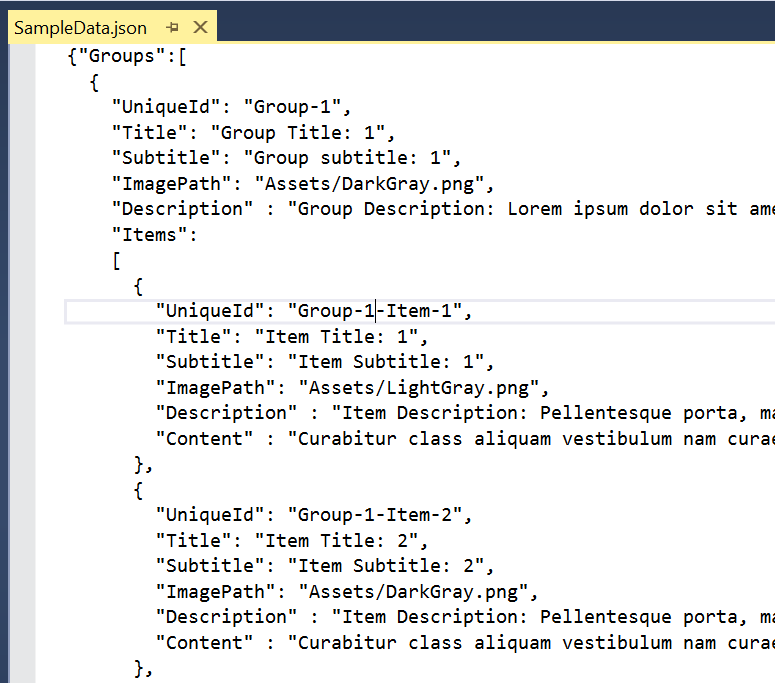
files in this folder will break the project template.



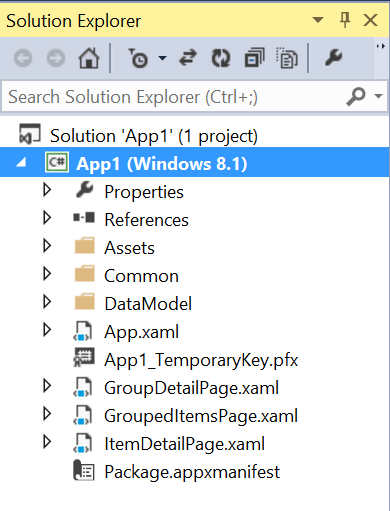
**DataModel Folder**

The class contains two public classes that define a generic sample data item and a generic sample data group. They both contain string properties, a constructor, and a to string method. It is intended that the programmer will replace this with their own data source. SampleDataSource is a sealed class that creates a collection of groups and items with content read from a static json file.



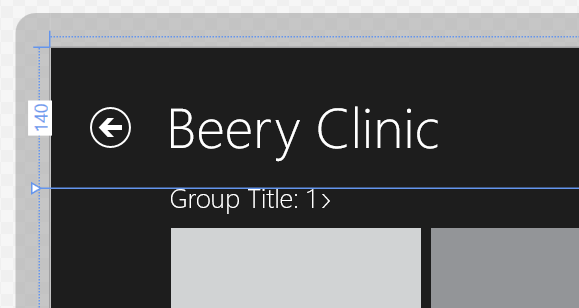


|  |  |  |
| --- | --- | --- |
|  |  |  |

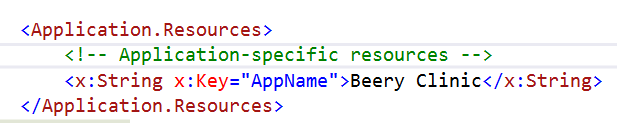
 **App.xaml**

App.xaml is the entry point to start the application. It is a cross between ASP.NET’s global.asax, WPF’s app file, and the main method used to boot WinForms and Console application executables.

In the App.xaml page contains only xaml, with no designer window. It also sets the application’s page title in the GroupedItemsPage. The xaml of GroupedItemsPage refrences the appname stored in app.xaml.



App.xaml

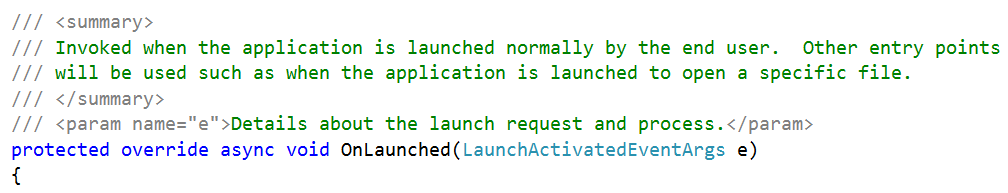


GroupedItemsPage.xaml

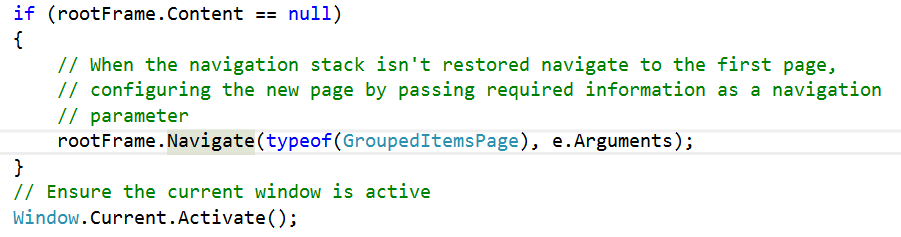


In the code behind we can set what page will initialize at start up. Change the reference in the OnLaunched method.

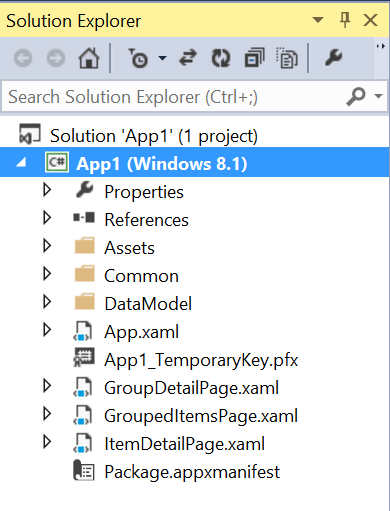
**App.xaml.cs**



// some code here…



Change your start page here

 **App1\_TemporaryKey.pfx**

Used to temporarily sign your application for local testing. You will be issued a new certificate created by Microsoft when publishing to the Store.

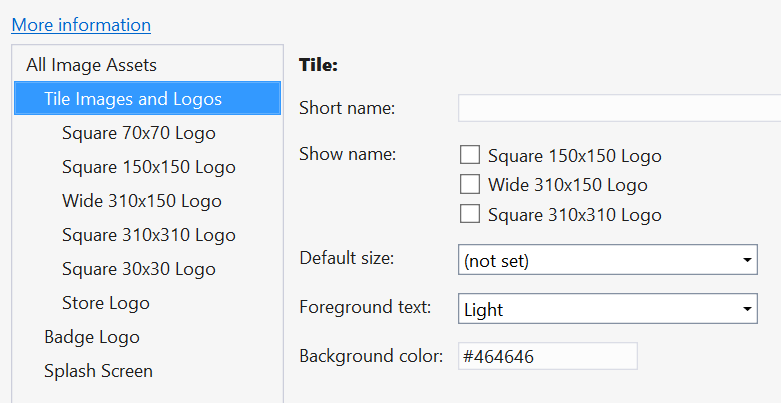
The certification process verifies the authenticity and integrity of an application.

**Package.appxmanifest**

A XAML based file containing manifest configuration settings governing how the application will integrate with Windows. The settings and properties contained in this file are required when you submit the application to the Windows Store. It is divided into six packages: application, visual assets, capabilities, declarations, content URIs, and packaging.

**Application** contains settings to set the title, splash screen content, and screen orientations.

**Visual Assets** are where you set your logo and splash screen to represent your application.

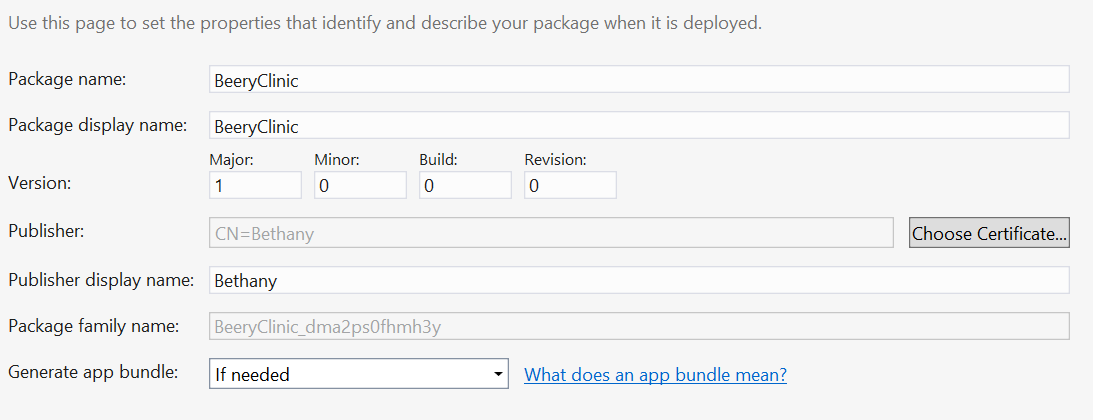


**Capabilities** indicate what system resources your application intends to use, like webcam, internet, location, microphone, music library, and several others. Forgetting to declare this will result in exception errors.

**Declarations** specify what interactions the application will support with other applications through contracts and extensions.

**Content URIs** contain the HTTPS URIs that your application uses to access geolocation devices and the clipboard.

**Packaging** sets deployment details.



**Navigation to another page**

To instantiate new pages from an event handler add the following code. The call to navigate is what instantiates a new NameofPage page.

if (this.Frame != null)

{

this.Frame.Navigate(typeof(NameofPage));

}

**Page definitions**

**Declaring xml namespace definitions, a map to rules elements need to adhere to**

**Default View Model**

The default view model is a way to pass data between the code behind and xaml. A view model is a special collection of data used exclusively by the view (xaml).

**CollectionViewSource**

CollectionViewSource is a collection class used as a go between data in view model and user interface elements on a XAML page. The class includes grouping and current-item support. Located

C#: public sealed class CollectionViewSource : DependencyObject

XAML:<CollectionViewSource .../>

In the code behind of GroupedItemsPage we have navigationHelper\_LoadState method used to populate the page with content passed during navigation. Bind the default view model to a collection of groups in xaml called sampleDataGroups.

DataContext="{Binding DefaultViewModel, RelativeSource={RelativeSource Self}}"

In the XAML, GroupedItemsPage.xaml:

Define a CollectionViewSource as a XAML resource and bind to it using the StaticResource markup extension.

<Page <!--Groups are bound into DefaultViewModel in code behind navigationHelper\_LoadState -->

DataContext="{Binding DefaultViewModel, RelativeSource={RelativeSource Self}}"

</Page>

<!--Groups in binding groups is coming from datacontext in the default view model -->

<CollectionViewSource

x:Name="groupedItemsViewSource"

Source="{Binding Groups}"

IsSourceGrouped="true"

ItemsPath="Items"

d:Source="{Binding Groups, Source={d:DesignData Source=/DataModel/SampleData.json,

Type=data:SampleDataSource}}"/>

<GridView

x:Name="itemGridView"

ItemsSource="{Binding Source={StaticResource groupedItemsViewSource}}"

**GridView control references CollectionViewSource class by x:Name for it’s data**

Code behind, GroupedItemsPage.xaml.cs:

Set source of CollectionViewSource XAML resource to a supported collection type

private async void navigationHelper\_LoadState(object sender, LoadStateEventArgs e)

{

var sampleDataGroups = await SampleDataSource.GetGroupsAsync();

// Bind to collection of groups in xaml called sampleDataGroups - in our case, gridview

this.DefaultViewModel["Groups"] = sampleDataGroups;

}

OR

Teams teams = new Teams();

var result =

from t in teams

group t by t.City into g

orderby g.Key

select g;

Groups.Source = result;