APIMASH Hands-on lab

Workbook 2: Databinding and Controls

Lab version: 1.0.0

Last updated: 6/1/2013

CONTENTS

Overview

Welcome to the **APIMASH** program! This series will help you build useful Windows Store apps based on data and services provided by third parties via their own APIs. APIMASH features starter kits in XAML/C# and HTML/JavaScript that will help you quickly get started using these APIs, and into the Windows Store.

This is the last in a series of workbooks that will help you understand the starter kits, how they work, and how you can build your own Windows Store apps based on them:

- **Workbook 1 – Getting started with APIMASH**
    - Understanding the architecture
    - Calling APIs
    - JSON and Serialization
- **Workbook 2 – Controls, Data Binding, and Windows Store App Design**
    - Understanding data binding
    - Common Controls – GridView, ListView, and more
    - Windows store App features – Snapping, Sharing, Search, etc.
    - UI Design options
- **Workbook 3 – Getting Your App into the Windows Store**
    - Get a Windows Developer Account
    - Using the Dashboard
    - Submitting an App
    - Addressing Certification Issues

As you work though these workbooks, you will learn to access data via APIs, create an interface with controls, bind that data to controls, and submit your app to the Windows Store. Along the way, you'll see Windows 8 features to help your app shine, including sharing, searching, tiles, and more.

Objectives

This workbook will show you how to:

- Understand data binding
- Understand Common Controls – GridView, ListView, and more
- Learn Windows store App features – Snapping, Sharing, Search, etc.
- Explore UI Design options

System requirements

You must have the following to complete this lab:

- Microsoft Windows 8
- Microsoft Visual Studio 2012

Exercises

This Hands-on lab includes the following exercises:

Understanding data binding

Common Controls – GridView, ListView, and more

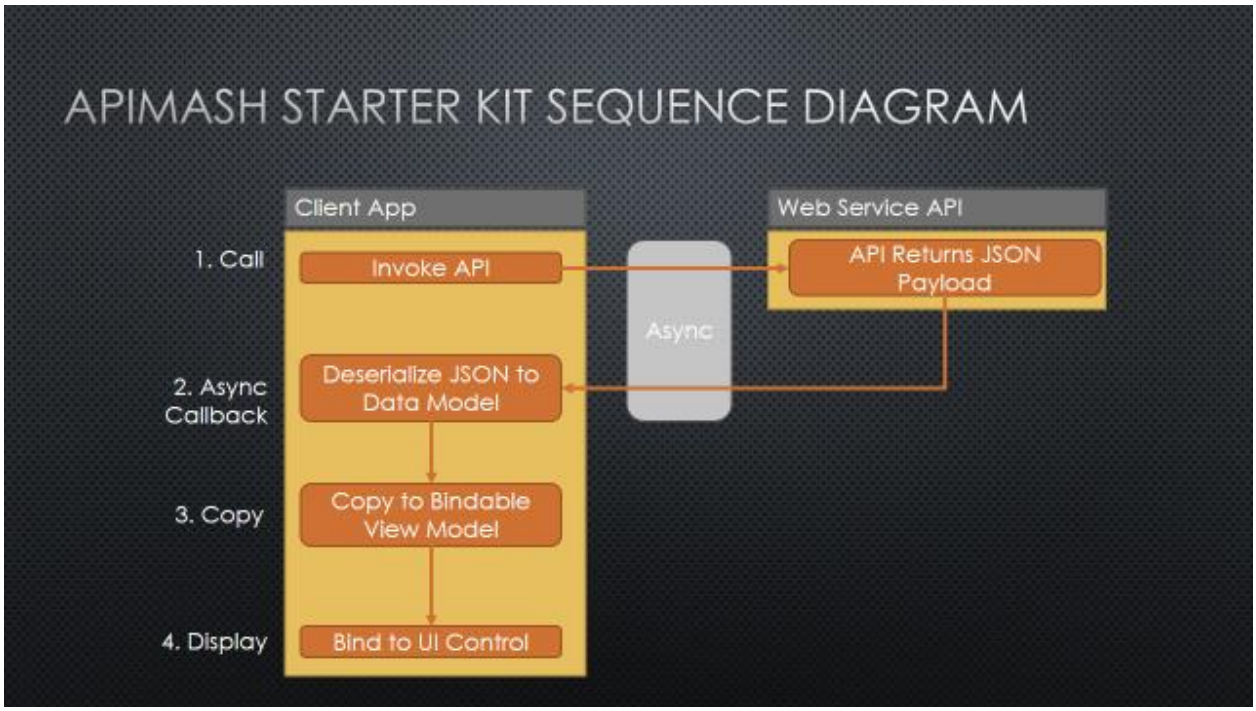
Windows store App features – Snapping, Sharing, Search, etc.

UI Design options

Estimated time to complete this lab: **50 to 70 minutes**.

Exercise 1: Understanding Data Binding

In the first exercise, we'll learn a bit about how databinding works, building on what we covered in the first workbook. After completing the first workbook, you should have a Data Model as seen in Step 2 of the diagram below:



This workbook will cover steps 3 and 4 in the diagram above.

Task 1 – Create the Bindable Object Model

Now that we have retrieved data from our chosen API, we have to choose how much of the data we will show in our app, and how to show it.

1. In Solution Explorer, expand the *APIMASH_OM.cs* file in the *<StarterKitName>Lib* project. JSON.NET is used to create instances of classes from the data returned by the API. In Solution Explorer, you can see all the classes we have to choose from and display in our project.

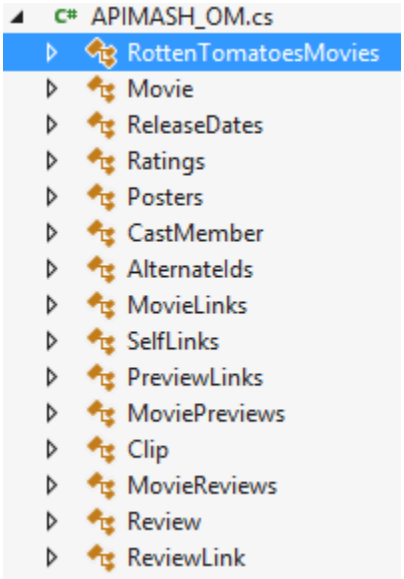


Figure 1

APIMASH Object Model from APIMASH_RottenTomatoes_StarterKit

2. In Solution Explorer, expand the *APIMASH_OM_Bindable.cs* file in the *<StarterKitName>Lib* project. This defines the format of the data that we want to display in our app.

Figure 2

APIMASH Bindable Object Model from APIMASH_RottenTomatoes_StarterKit

3. We handle the mapping of our Object Model to our Bindable Object Model when we get the response to our API call. One example of this is in the `apiInvoke_OnResponseInTheaters` method of our *GroupedItemsPage.xaml.cs* file that is called when the API call is complete.

```
        async private void apiInvoke_OnResponseInTheaters(object sender,
APIMASHEvent e)
        {
            var response = (RottenTomatoesMovies)e.Object;

            if (e.Status == APIMASHStatus.SUCCESS)
            {
                // copy data into bindable format for UI
                APIMASH_RottenTomatoesCollection.Copy(response,
System.Guid.NewGuid().ToString(), "In Theaters");
                this.DefaultViewModel["AllGroups"] =
APIMASH_RottenTomatoesCollection.GetGroups("AllGroups");
                _loaded = true;
            }
            else
            {
                var md = new MessageDialog(e.Message, "Error");
                bool? result = null;
                md.Commands.Add(new UICommand("Ok", new
UICommandInvokedHandler((cmd) => result = true)));
                await md.ShowAsync();
            }
        }
```

4. If you take a look at the Copy method in the *APIMASH_<StarterKitName>.cs* file in the *<StarterKitName>Lib*, you can see how it populates the objects defined in the *APIMASH_OM_Bindable.cs* file.

```
        public static void Copy(RottenTomatoesMovies response, string
groupId, string groupName)
        {
```

```
            try
            {
                MovieGroup mg =
APIMASH_RottenTomatoesCollection.GetGroupByTitle(groupName);
                if (mg != null)
                    mg.Items.Clear();
                else
                    mg = new MovieGroup(groupId, groupName,
response.Movies[0].Posters.Original);

                foreach (var mi in response.Movies.Select(t => new MovieItem(
                    t.Id,
                    t.Title,
                    t.MPAARating,
                    t.Ratings.AudienceRating,
                    t.Ratings.CriticsRating,
                    t.Links.Clips,
                    t.Links.Reviews,
                    t.Links.Cast,
                    t.Posters.Original,
                    t.Synopsis,
                    mg)))
                {
                    mg.Items.Add(mi);
                }
                _movieData._allGroups.Add(mg);
            }
            catch (Exception e)
            {
                throw;
            }
        }
```

Figure 3

Copy method in APIMASH_RottenTomatoes.cs

Exercise 2: Common Controls

Task 1 – Examine your Starter Kit UI & Controls

The first step is to take a look at the controls and UI that exists in your chosen Starter Kit. To continue with the example from Workbook 1 and Exercise 1 in this workbook, we'll use the APIMASH_RottenTomatoes_StarterKit which is based on the Grid App template in Visual Studio.

1. Open the *GroupedItemsPage.xaml* file in Visual Studio and see the XAML contains a GridView control and a ListView control.

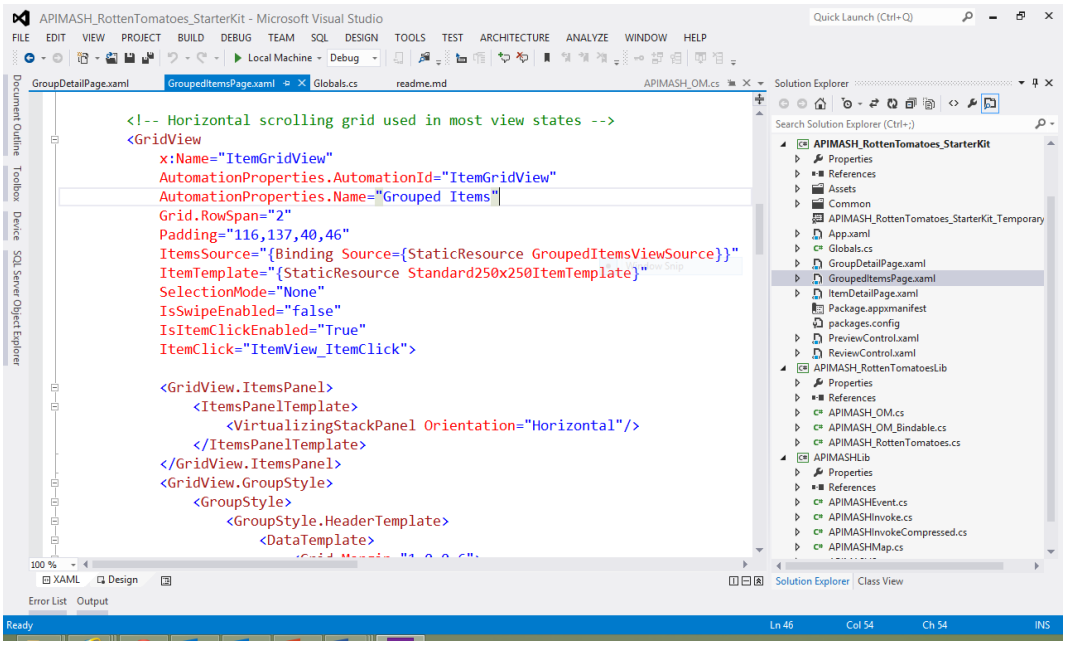


Figure 4

GropedItemsPage.xaml

2. See the XAML in the GridView, highlighted below, that binds the GridView to the GroupedItemsViewSource which is a resource defined above as a Page Resource.

```
<GridView
   x:Name="ItemGridView"
   AutomationProperties.AutomationId="ItemGridView"
   AutomationProperties.Name="Grouped Items"
   Grid.RowSpan="2"
   Padding="116,137,40,46"
   ItemsSource="{Binding Source={StaticResource GroupedItemsViewSource}}"
   ItemTemplate="{StaticResource Standard250x250ItemTemplate}"
   SelectionMode="None"
   IsSwipeEnabled="false"
   IsItemClickEnabled="True"
         ItemClick="ItemView_ItemClick">
   ...
```

Figure 5

Gridview Binding

```
<CollectionViewSource
      x:Name="GroupedItemsViewSource"
      Source="{Binding Groups}"
      IsSourceGrouped="true"
      ItemsPath="TopItems"
      d:Source="{Binding AllGroups, Source={d:DesignInstance
Type=rt:APIMASH_RottenTomatoesCollection,
IsDesignTimeCreatable=False}}"/>
```

Figure 3

Gridview Binding continued

3. Notice that the ListView control in *GroupedItemsPage.xaml.cs* follows the same pattern.

4. Lastly, the binding source object can be a single object, not just a collection of objects as we have seen used in the previous steps. In both the GridView and ListView controls in *GroupedItemsPage.xaml.cs* we see that a Button control containing a TextBlock is bound to the Title property of the DefaultViewModel.

```
<Button
      AutomationProperties.Name="Group Title"
      Click="Header_Click"
      Style="{StaticResource TextPrimaryButtonStyle}" >
      <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Title}" Margin="3,-7,10,10"
            Style="{StaticResource GroupHeaderTextStyle}" />
            <TextBlock Text="{StaticResource ChevronGlyph}"
            FontFamily="Segoe UI Symbol" Margin="0,-7,0,10"
            Style="{StaticResource GroupHeaderTextStyle}"/>
      </StackPanel>
</Button>
```

Figure 4

Single object databinding

Exercise 3: Windows Store Features

In this exercise we will examine a few features that will make your app shine on Windows 8. First we will look at Snapping, and then the Share, Search, and Settings contracts.

Task 1 – Snapping

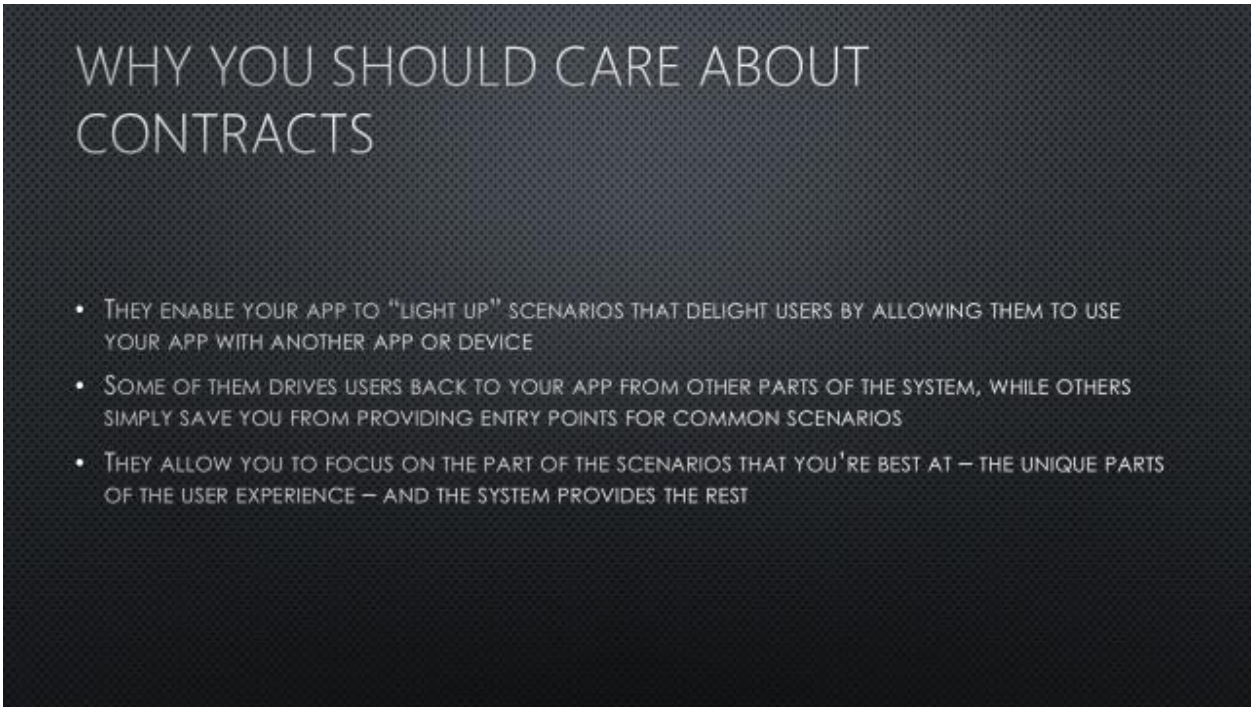
The reason we have both a GridView and a ListView in *GroupedItemsPage* is so that our layout adjusts accordingly when our app is in Snapped View. This allows our app to look and function properly for the user at all times. The VisualStateManager handles this process.

1. Scroll to the bottom of *GroupedItemsPage.xaml* to find where the **VisualStateGroups** are defined by the **VisualStateManager**.

2. Find the **VisualStateGroup** for “FullScreenPortait” and “Snapped”. This XAML contains animations that change the style of some of the controls (like PageTitle and BackButton) and switches the ItemListView and the ItemGridView from Visible to Collapsed, depending on the VIsualState.

3. For more on Snapping and VisualStates see: http://msdn.microsoft.com/en-us/library/windows/apps/hh465371.aspx

Task 2 – Search Contract



To continue with making our app shine on Windows 8, you may want to implement contracts. You can learn more about contracts and other extensions to your Windows Store app at: http://msdn.microsoft.com/en-us/library/windows/apps/hh464906.aspx

1. You can add a search pane to your app so users can search not only your app's content but content from other apps as well. Users can also transfer the search query itself to other apps. When you participate in this contract, you agree to make your app's content searchable by other participants and to present search results from those participants in your app. Participating in this contract helps you gain traffic and usage for your app. For more info, see: http://msdn.microsoft.com/en-us/library/windows/apps/xaml/JJ130767(v=win.10).aspx.

Task 3 – Share Contract

1. You can help users share content from your app with another app or service, and vice versa. Participating in the Share contract means that you do not have to write extra code or provide other developers with an SDK for your app just to share content. Apps that support the Share contract can automatically share content to and from any other app that also supports the contract. Participating in this contract helps you gain traffic and usage for your app. For more info, see: http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh871363.aspx.

Task 4 – Settings Contract

You can provide quick, in-context access to settings that affect the user's experience with your app. Participating in this contract ensures that your app is consistent with the Windows settings model. For more info, see: http://msdn.microsoft.com/en-us/library/windows/apps/hh770540.aspx.

1. This is the only Contract that is defined in the *APIMASH_RottenTomatoes_StarterKit*. We use the Settings Pane to display our About, Support and Privacy Policy information. In the *GroupedItemsPage.xaml.cs, GroupDetailPage.xaml.cs,* and *ItemDetailPage.xaml.cs* files you will find a method called `settingsPane_CommandsRequested` which creates settings commands and adds them to the Settings flyout. This is where you will add links to your About page and Privacy Policy.

```
        void settingsPane_CommandsRequested(SettingsPane sender,
SettingsPaneCommandsRequestedEventArgs args)
        {
            // update to supply links to your about, support and privavy
policy web pages
            var aboutCmd = new SettingsCommand("About", "About", (x) =>
Launcher.LaunchUriAsync(new Uri("")));
            var supportCmd = new SettingsCommand("Support", "Support", (x)
=> Launcher.LaunchUriAsync(new Uri("")));
            var policyCmd = new SettingsCommand("PrivacyPolicy", "Privacy
Policy", (x) => Launcher.LaunchUriAsync(new Uri("")));

            args.Request.ApplicationCommands.Add(aboutCmd);
            args.Request.ApplicationCommands.Add(supportCmd);
            args.Request.ApplicationCommands.Add(policyCmd);
        }
```

Exercise 4: UI Design Options

Even before we get data from APIs, decide what data to use and bind it to controls for display, we need to decide WHAT our app will do and the best way to display that data with controls. This planning is a vital first step to creating Windows Store apps. For more on Designing User Experience for apps, see: http://msdn.microsoft.com/en-us/library/windows/apps/hh779072.aspx. Below we will highlight some of the considerations to make when designing your apps.

Task 1 – Navigation Patterns

Learn how to organize the content in your Windows Store app so your users can navigate easily and intuitively. Using the right navigation patterns helps you limit the controls that are persistently on screen, such as tabs. This lets people focus on the current content.

1. *APIMASH_RottenTomatoes_StarterKit* uses a hierarchical pattern, for more patterns see: http://msdn.microsoft.com/en-us/library/windows/apps/hh761500.aspx.

Task 2 – Touch Interaction

Keep in mind that your users may be using either Keyboard and Mouse or Touch devices to interact with your app. Windows 9 supports both, and the design guidelines take touch into consideration.

1. Follow Touch Guidance found here: http://msdn.microsoft.com/en-us/library/windows/apps/hh465415.aspx.

Task 3 – Animations

Purposeful, well-designed animations bring apps to life and make the experience feel crafted and polished. Help users understand context changes, and tie experiences together with visual transitions.

1. Learn more about animations and when to use them: http://msdn.microsoft.com/en-us/library/windows/apps/hh975420.aspx

Task 4– Placing Commands

You have several surfaces you can place commands and controls on in your Windows Store app, including the app window, pop-ups, dialogs, and bars. Choosing the right surface at the right time can mean the difference between an app that's a breeze to use and one that's a burden.

1. Learn about context for commands: http://msdn.microsoft.com/en-us/library/windows/apps/hh761499.aspx

Summary

In this workbook, you learned how databinding works, whether to a single object or a collection of objects. We created a Bindable Object Model based on the data returned from an API. We took a look at some common controls available to display data. We examined some Windows 8 features to implement in your apps, like contracts and handling ViewStates. Lastly, we explored some UI guidelines, so you can customize your Windows Store app!

The the third and final workbook, you will see how to register as a Windows Store developer and learn how to prepare your apps for publication, then take your completed app and publish it to the Windows Store!