APIMASH Hands-on Lab

Workbook 1: Getting Started & Getting Data

Lab version: 1.0.0

Last updated: 5/11/2013

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 first 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 (This workbook) – Getting started with APIMASH**
    - Choosing and API & Starter Kit
    - Create an API Developer Account
    - Understanding the Starter Kit
    - Understanding the API Call
- **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.

Consider a True Mashup

Many of the starter kits focus on a single API, and this can be a great place to start and quickly create your own app. You might also consider creating a **"Mashup"**, where the data from two or more APIs are

combined to create unique and useful experiences not provided directly by the original APIs. For example, connecting friend lists from a social media service to filter or identify reviews, events, or movies of particular interest.

Note that a few of the Starter Kits are already mashups!

Objectives

This workbook will show you how to:

- Get started with an APIMASH Starter Kit
- Understand the structure of the project
- See how the target API is called to return data
- Understand the project's data classes and how they are populated with the returned API data

System requirements

You must have the following to complete this workbook:

- Microsoft Windows 8
- Microsoft Visual Studio 2012 (any version)

Exercises

This workbook includes the following exercises:

1. Choose an API & Starter Kit
2. Create an API Developer Account
3. Understanding the Starter Kit
4. Understanding the API Call

Estimated time to complete this workbook: **45 to 60 minutes**.

Exercise 1: Choose an API & Starter Kit

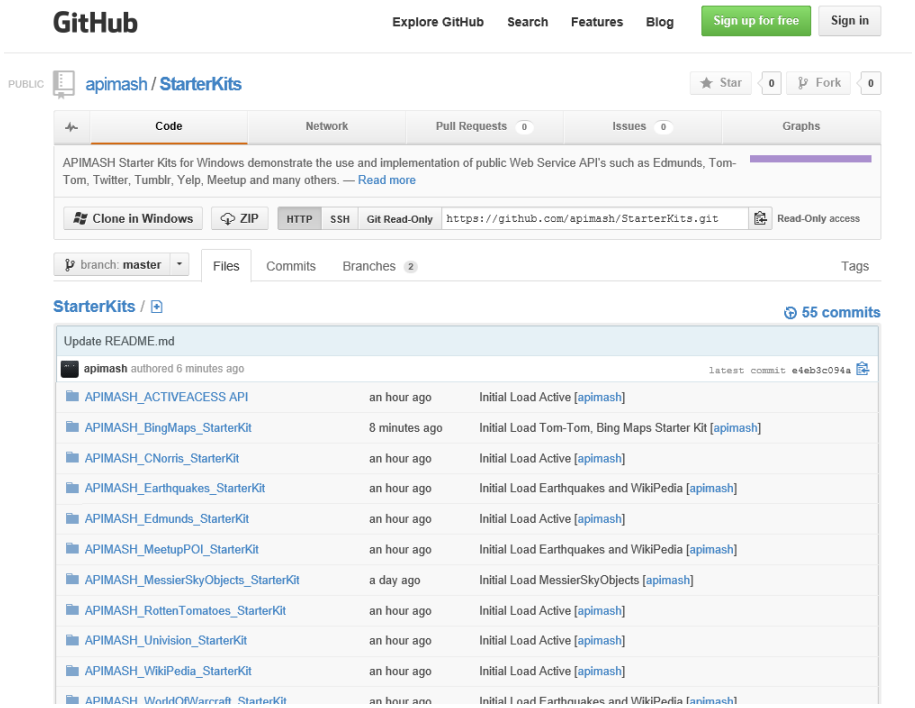
In the first exercise, you'll select the API to use, register for a developer account, and download the associated Starter Kit.

Task 1 – Select Your API

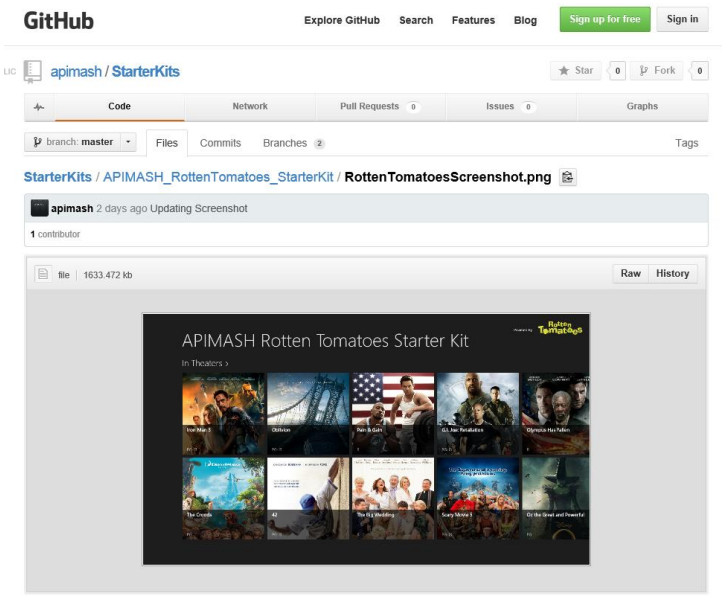
On the APIMASH site, you'll see many Starter Kits, each associated with a particular API or APIs.

2. Navigate to apimash.github.io/StarterKits

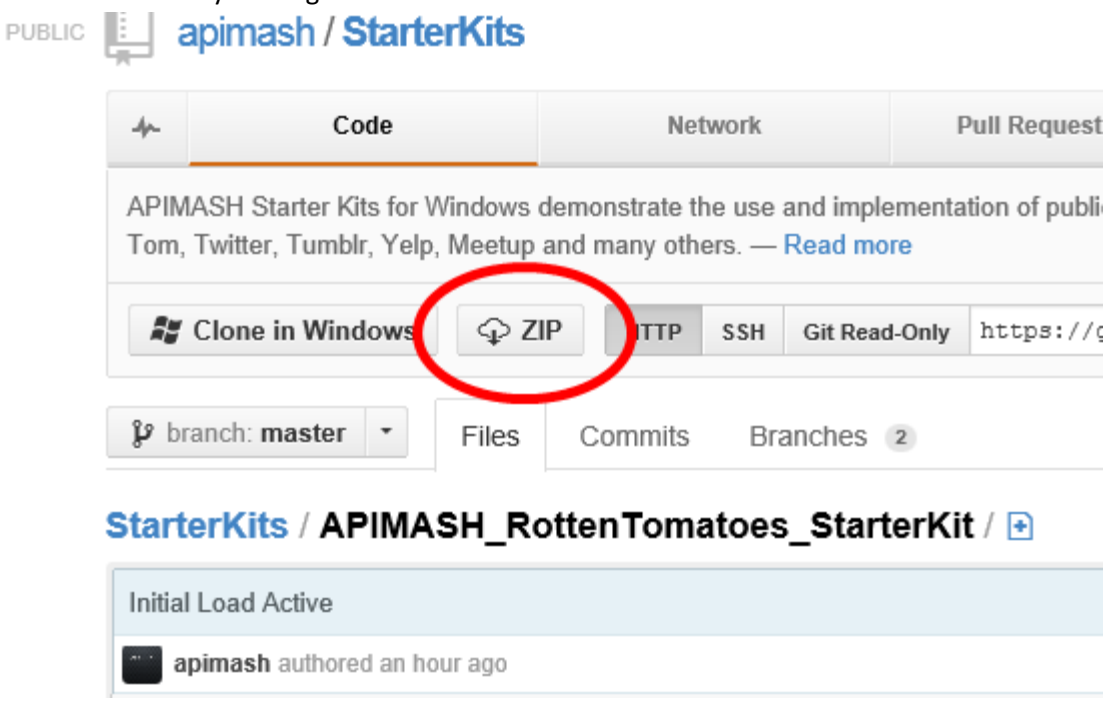
3. Click *"View on GitHub"* at the top right corner.
4. Take some time to review the listed Starter Kits (you can click into each to look at files.) Some are in XAML/C#, and others are in HTML5/JavaScript.



Note: Each Starter Kit has screenshots to show you what the app will look like when fully connected using an API developer account. Look in the root directory for each kit.



5. Download the kits by clicking the ZIP button:



6. Unzip to a folder (e.g. *C:\APIMASH*) – we'll return to this folder in a moment.

Exercise 2: Create an API Developer Account

APIs typically require that calls be made by registered developers and/or applications. The Starter Kits were developed using private accounts and those have been removed for publishing. Now you'll need to create your own API developer account and add they key(s) given to you into the Starter Kit before you'll be able to run the app.

Task 1 – Visit the API Website

Your Starter Kit is associated with an API. You should find the API's developer site listed in the Starter Kit's README file, but it should be easy to find on the website (typically a "Developers" link on the site's footer.)

1. Navigate to the API's website (e.g. http://developer.rottentomatoes.com).
2. Take the time to read and understand the developer and API usage policies.

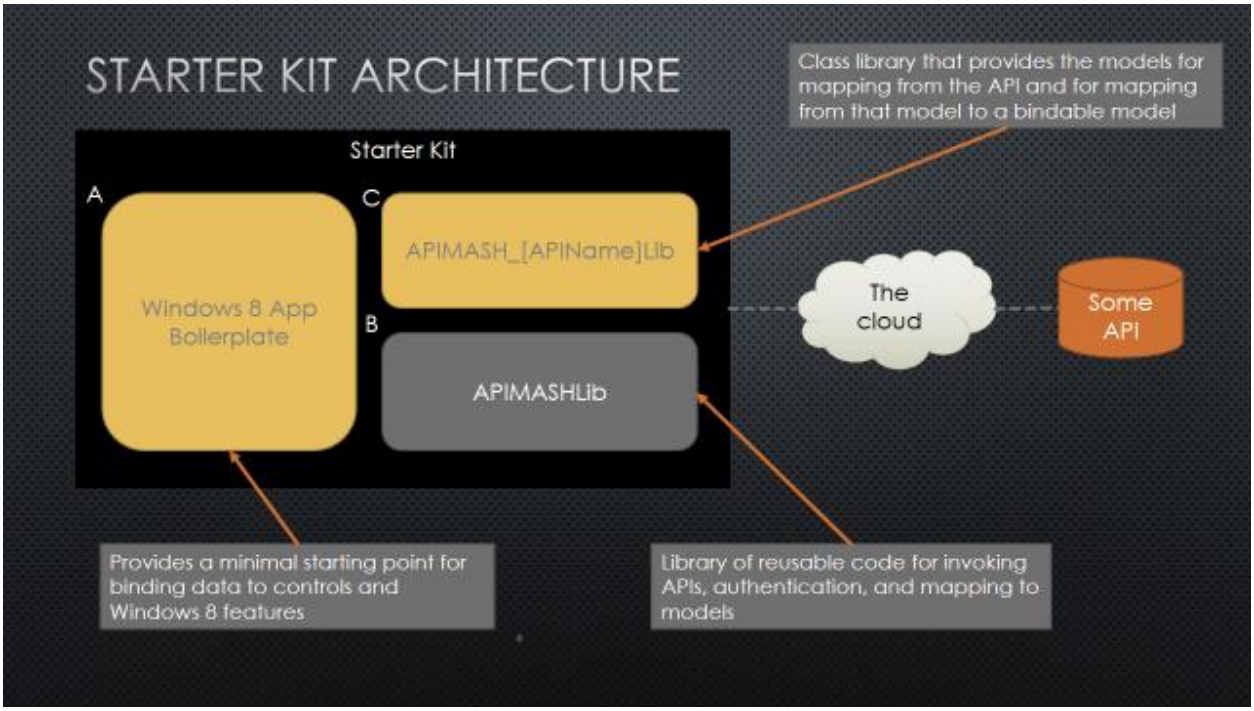
3. No **really**, do step 2. Their terms may limit your use, require certain presentation/attribution, stipulate that you cannot make money from your application, or other important items.
4. Create a developer/API account as instructed on the site.
5. Some APIs will immediately reply with your key, others may take longer (even days).
6. Keep your developer/API key handy. You will need to paste it into the Starter Kit later.

Exercise 3: Understanding the Starter Kit

Now that you've selected an API, registered for an account, and downloaded the associated Starter Kit, it's time to open it and have a look around.

Task 1 – The Starter Kit Structure

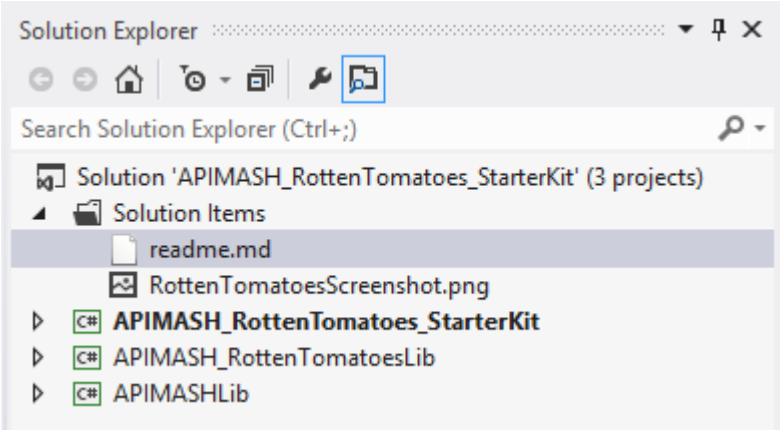
Each Starter Kit follows the same structure/architecture.



Open the Starter Kit to see this structure in action.

2. Navigate to your Starter Kit and open the Visual Studio solution file (e.g. *APIMASH_<StarterKitName>_StarterKit.sln*).

3. In Solution Explorer, you'll see a structure like this for XAML/C# projects:

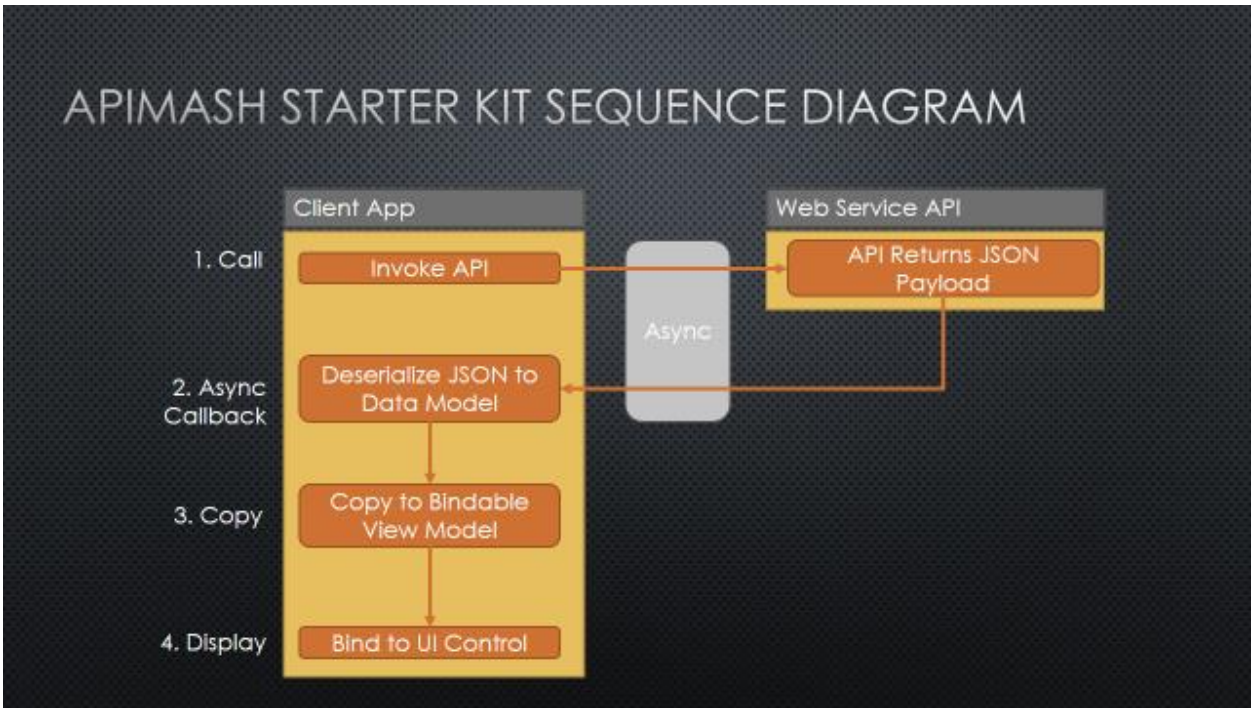


4. Compare these projects to the diagram above.
5. In this example, *APIMASH_RottenTomatoes_StarterKit* is the application/UI project (referred to as "Windows 8 Boilerplate" in the diagram.) Take a look around. This project will vary significantly from Kit to Kit, but minimally has:
    a. At least one screen with controls that are data bound to the Bindable Object Model. Some are based on simple controls built with the "Blank" Windows Store project template, others may use the "Grid" or "Split View" templates.
    b. A call to the target API(s) that use classes in the APIMASHLib project.
6. Open APIMASHLib and APIMASH_<StarterKitName>Lib and take a quick look around. We'll cover more of these in Exercise 4, but you'll see code to handle the API calls, process returned data, and copy to bindable objects used by the UI project.

Exercise 4: Understanding the API Call

You've seen the overall structure of the Starter Kits, now let's look at how the call to the API is made, and how the returned data is processed.

When each Starter Kit calls the target API(s), they follows the same flow:



The Client App represents the XAML or HTML application itself, along with the related class libraries, APIMASHLib and <StarterKitName>Lib. The Web Service API is the externally hosted third-party API web service.

The steps in each call are:

1. UI layer, typically in response to an event like a button click, forms the API call and makes the web request via the `APIMASHInvoke` class in APIMASHLib. The API process the request and returns a result as either JSON or XML.

   This call is made asynchronously (as all calls that can take longer than 50ms are in Windows Store apps), so response from the server will be handled in a separate method.

2. The callback method is invoked and determines if the request was successful or encountered an error. In order to use the data, the JSON/XML data is converted to instances of classes. The Kits use JSON.NET from Newtonsoft (http://json.codeplex.com) as a helpful library do this conversion.

Each kit has a set of classes, called the Object Model (OM), based on the structure of data returned by the API (e.g. Movies, Actors, Theaters, etc.) These are in the *<StarterKitName>Lib* project, in the *APIMASH_OM.cs* file. JSON.NET is used to create instances of these, populated with data from the API call.

3. However, following **Model View View-Model (MVVM)** design (see http://msdn.microsoft.com/en-us/magazine/jj651572.aspx for details), data used for the UI and controls may be *different* from the source data. In addition, some interfaces need capabilities to handle changes to the data via the UI. To support this, each Kit has a related Bindable Object Model, found in *APIMASH_OM_Bindable.cs*. These classes have methods that support copying from the original OM instances to the bindable OM instances.
4. Finally, the data (as copied to the bindable OM) is displayed in the app via *data binding*, where controls like GridView and ListView are paired with those data sources.

> **Note:** More on parts 3 and 4 above is in Workbook #2.

Task 1 –Starter Kit API Call

While each Starter Kit follows the above approach, each is slightly different due to API and data requirements. Now you will look at your Starter Kit to see how your specific kit works.

1. Open your Kit in Visual Studio (double-click the *<StarterKitName>.sln* file).
2. Expand the APIMASHLib project and open the APIMASHInvoke.cs file.

   The method `Invoke()` is key here. If you follow the code, you'll eventually find the *APIMASHMap* class' `LoadObject<T>()` method:

```
async public static Task<T> LoadObject<T>(string apiCall)
{
    var ws = new HttpClient();
    var uriAPICall = new Uri(apiCall);
    var objString = await ws.GetStringAsync(uriAPICall);
    return (T)DeserializeObject<T>(objString);
}
```

   This is where the real API work is done – calling the remote API, getting the response, and deserializing the result into Object Model instances. There's no real need dive through the code in *APIMASHLib*, as it's used as-is by the other projects and you shouldn't need to change anything in it.

3. Moving a layer up in the architecture, open the *APIMASH_<StarterKitName>Lib* project.
4. The *APIMASH_OM.cs* file contains the classes used in the `DeserializeObject<T>` call above to convert the JSON or XML data to class instances.

> **Note:** If you make changes to the API call such that it returns different data, you will need to change these classes (and potentially add new ones) to capture the different data.

5. Moving one more layer up, open the *APIMASH_<StarterKitName>_StarterKit* project.

To find the actual call(s) to the API, locate any lines of code using `APIMASHLibInvoke`. Here's one example, but each Kit may be different:

```
private readonly APIMASHInvoke apiInvokeReviews;

...

private void ReviewButton_Click(object sender, RoutedEventArgs e)
{
    var mi = (MovieItem)this.DefaultViewModel["Item"];
    apiInvokeReviews.OnResponse += apiInvokeReviews_OnResponse;
    var apiCall = mi.Reviews + "?" + Globals.ROTTEN_TOMATOES_API_KEY;
   apiInvokeReviews.Invoke<MovieReviews>(apiCall);
}
```

In this case, when the user presses the Review button, the code connects the `apiInvokeReviews_OnResponse` method to be called when the API call is complete. The API call is made by taking a root URL (e.g. http://api_site_here?option1=123&option2=456) and appending the developer/account key.

6. Find your Starter Kit's API response handling method and review the code. You should see code to check for success, copy object(s) to the bindable versions, and data bind those objects to the related UI controls.

Well done! By taking the time to understand what's happening in your Starter Kit, you'll be ready to think creatively, add new features, and get everything running smoothly!

Summary

In this workbook, you reviewed and selected a Starter Kit and API, registered for a developer key with that API, and reviewed the architecture and key methods of the Kit.

In the next Workbook, you'll take the next step by looking at the UI of your Starter Kit, understanding control options, and learning more about how data binding works so you'll be able to make changes to the appearance and functionality of what will be your Windows Store app!