

Bing Search API – Quick Start and Code Samples

October 2012

# Introduction

With the Microsoft Bing Search API, you can use data that is collected by the Bing search engine in your desktop application, webpage, or other programming environment that can issue an HTTPS request.

This document describes how you can get started using the Bing Search API followed by a collection of code samples. First, it provides two tutorials: one you can follow to create a simple PHP-based web application and one you can follow to create an application using Microsoft .NET and C#.

Contents

[Introduction 1](#_Toc338721219)

[Getting started 1](#_Toc338721220)

[Account keys 1](#_Toc338721221)

[Tutorial: Creating a simple PHP application 1](#_Toc338721222)

[Set up the HTML 1](#_Toc338721223)

[Create the PHP file 2](#_Toc338721224)

[Processing the Bing API search request 2](#_Toc338721225)

[Tutorial: Creating a simple .NET Framework application 3](#_Toc338721226)

[Bing Search API parameter reference 3](#_Toc338721227)

[Bing Search API option parameters 4](#_Toc338721228)

[Options 4](#_Toc338721229)

[WebSearchOptions 4](#_Toc338721230)

[Code Samples 5](#_Toc338721231)

[Prepare Code Sample for Service Operation 5](#_Toc338721232)

[Executing a Web Service Operation 5](#_Toc338721233)

[Executing an Image Service Operation 6](#_Toc338721234)

[Executing a Video Service Operation 6](#_Toc338721235)

[Executing a News Service Operation 7](#_Toc338721236)

[Executing a SpellingSuggestions Service Operation 7](#_Toc338721237)

[Executing a RelatedSearch Service Operation 8](#_Toc338721238)

[Executing a Composite Service Operation 9](#_Toc338721239)

[Creating a jQuery / AJAX web application 9](#_Toc338721240)

[Creating a C++ App for Windows 8 12](#_Toc338721241)

[Creating an Objective-C Console Application 25](#_Toc338721242)

# Getting started

To get started, first create an account in the Windows Azure Marketplace and subscribe to one of the Bing Search API services. Windows Azure Marketplace is a cloud-based data service that enables you to find and consume published data sets and web services.

You can subscribe to one of the following subscription types.

|  |  |
| --- | --- |
| **Subscription type** | **Capabilities** |
| [Bing Search API](https://datamarket.azure.com/dataset/5BA839F1-12CE-4CCE-BF57-A49D98D29A44) | Gives you access to Bing web, image, video, news, related search results, and spelling suggestions. |
| [Bing Search API - Web Results Only](https://datamarket.azure.com/dataset/8818F55E-2FE5-4CE3-A617-0B8BA8419F65) | Gives you access only to Bing web results. |

With both subscription types, you can select a service level to meet the needs of your organization. Service levels are based on a monthly maximum of transactions. The lowest level (5000 transactions per month) is free.

**Note:** The tutorials in this document use the Bing Search API subscription type to get both web and image results from the Bing search engine.

# Account keys

When you create an account in the Windows Azure Marketplace, you get a default account key, which you can’t change or remove. However, you can create additional account keys.

You must use one of your account keys to access your Bing Search API dataset subscription. It’s better to use an account key that you created instead of the default one. That way, if your key is compromised, you can remove it and create another one in its place.

As soon as you have an account key, you’re ready to start creating a simple application.

# Tutorial: Creating a simple PHP application

Here is a list of what you’ll accomplish in this tutorial:

* Set up a simple HTML page with a search form
* Create a PHP file that uses submitted values to create a request to the Bing Search API
* Process the Bing API search request and parse and display the results

## Set up the HTML

The following is example code for a simple HTML page that contains a basic search form. You can use this form to choose the service operation (web or image) and enter the text of your query.

|  |
| --- |
| **Basic Search Form** |
| <html>  <head>  <title>Bing Search Tester (Basic)</title>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  </head>  <body>  <h1>Bing Search Tester (Basic)</h1>  <form method="POST" action="bing\_basic.php">  <label for="service\_op">Service Operation</label><br/>  <input name="service\_op" type="radio" value="Web" CHECKED /> Web  <input name="service\_op" type="radio" value="Image" /> Image  <br/>  <label for="query">Query</label><br/>  <input name="query" type="text" size="60" maxlength="60" value="" /><br /><br />  <input name="bt\_search" type="submit" value="Search" />  </form>  <h2>Results</h1>  {RESULTS}  </body>  </html> |

Fairly standard. One thing to note is that the page specifies UTF-8 encoding. You need this to correctly display the results from the Bing Search API. Another thing to note is the *{RESULTS}* placeholder. Using string substitution, you’ll insert the results from the API and avoid a lot of the PHP echo and HTML mash up. Save the sample HTML as bing\_basic.html.

## Create the PHP file

Next, copy the following code and save it as bing\_basic.php. Replace the value of $acctKey with your account key.

|  |
| --- |
| **PHP.1 – Outline of the application** |
| <?php  /\*\*\*\*  \* Simple PHP application for using the Bing Search API  \*/  $acctKey = 'YourAccountKey';  $rootUri = 'https://api.datamarket.azure.com/Bing/Search';  // Read the contents of the .html file into a string.  $contents = file\_get\_contents('bing\_basic.html');  if ($\_POST['query'])  {  // Here is where you'll process the query.  // The rest of the code samples in this tutorial are inside this conditional block.  }  echo $contents;  ?> |

The code in PHP.1 contains an outline of the basic tasks you’ll accomplish with this application, namely:

1. Get the HTML from the bing\_basic.html file.
2. Process the query (if one was entered in the search form) by calling the Bing Search API and using string substitution on the {RESULTS} placeholder.
3. Send everything back to the browser with the final echo statement.

## Processing the Bing API search request

The rest of this tutorial focuses on what you’ll do to complete step #2, that is, how you’ll proceed when the **query** field of the form contains a value. The rest of the code examples in this tutorial go inside the conditional block that checks if the **query** field contains a value.

To complete step 2, first, prepare the final request URI by adding the service operation and the encoded query to the root URI.

|  |
| --- |
| **PHP.2 – Preparing the request URI** |
| // Encode the query and the single quotes that must surround it.  $query = urlencode("'{$\_POST['query']}'");    // Get the selected service operation (Web or Image).  $serviceOp = $\_POST['service\_op'];  // Construct the full URI for the query.  $requestUri = "$rootUri/$serviceOp?\$format=json&Query=$query"; |

Here are the examples of your final URI to the Bing Search API:

**Web:**

<https://api.datamarket.azure.com/Bing/Search/Web?$format=json&Query=%27Xbox%27>

**Image:**

<https://api.datamarket.azure.com/Bing/Search/Image?$format=json&Query=%27Xbox%27>

Next, encode your account key by using Base64 encoding and create a stream context that you’ll use with the Bing Search API request.

|  |
| --- |
| **PHP.3 – Creating the stream context and getting the response from Bing** |
| // Encode the credentials and create the stream context.  $auth = base64\_encode("$acctKey:$acctKey");  $data = array(  'http' => array(  'request\_fulluri' => true,  // ignore\_errors can help debug – remove for production. This option added in PHP 5.2.10  'ignore\_errors' => true,  'header' => "Authorization: Basic $auth")  );  $context = stream\_context\_create($data);  // Get the response from Bing.  $response = file\_get\_contents($requestUri, 0, $context); |

Finally, decode the JSON response, parse the results, and substitute the *{RESULTS}* placeholder.

|  |
| --- |
| **PHP.4 – Creating the final results to show to the user** |
| // Decode the response.  $jsonObj = json\_decode($response);  $resultStr = '';  // Parse each result according to its metadata type.  foreach($jsonObj->d->results as $value)  {  switch ($value->\_\_metadata->type)  {  case 'WebResult':  $resultStr .=  "<a href=\"{$value->Url}\">{$value->Title}</a><p>{$value->Description}</p>";  break;  case 'ImageResult':  $resultStr .=  "<h4>{$value->Title} ({$value->Width}x{$value->Height}) " .  "{$value->FileSize} bytes)</h4>" .  "<a href=\"{$value->MediaUrl}\">" .  "<img src=\"{$value->Thumbnail->MediaUrl}\"></a><br />";  break;  }  }  // Substitute the results placeholder. Ready to go.  $contents = str\_replace('{RESULTS}', $resultStr, $contents); |

After you parse the results and put them inside the *$contents* variable, just send everything back to the browser with the echo statement (shown in PHP.1). You’re done.

# Tutorial: Creating a simple .NET Framework application

Creating a .NET Framework C# application is pretty easy. Most of the work that you need to do is handled for you by the Microsoft .NET Framework C# Service Proxy Class Library. You can download the service proxy for the Bing Search API by clicking the following link: <https://datamarket.azure.com/dataset/explore/getproxy/5ba839f1-12ce-4cce-bf57-a49d98d29a44>.

**To create a simple .NET Framework application**

1. Create a new C# console application in Microsoft Visual Studio.
2. Add the service proxy container file (BingSearchContainer.cs) that you downloaded to the same directory as the new application and add the file to your project.
3. Add a reference to System.Data.Services.Client.
4. Paste the following code into the program.cs file:

|  |
| --- |
| **C# - Sample application to get image results** |
| using System;  using System.Net;  namespace Microsoft.Samples.BingSearch  {  class Program  {  static void Main(string[] args)  {  // This is the query - or you could get it from args.  string query = "Xbox Live";  // Create a Bing container.  string rootUri = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUri));  // Replace this value with your account key.  var accountKey = "YourAccountKey";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(accountKey, accountKey);  // Build the query.  var imageQuery = bingContainer.Image(query, null, null, null, null, null, null);  var imageResults = imageQuery.Execute();  foreach (var result in imageResults)  {  Console.WriteLine(result.Title);  }  }  }  } |

1. Replace the value of accountKey with your account key and you’re ready to run the application. Results are displayed on the console.

# Bing Search API parameter reference

The following table shows the various parameters that you can use when you make a request through the Bing Search API and the service operations that the parameter applies to. All parameters, except for *Query*, are optional. Parameter names are case-sensitive.

For information about the values that you can assign to each parameter, see [Schema Tabular Documentation for Bing Search API](http://go.microsoft.com/fwlink/?LinkID=252151)on the Bing Search API page of the Windows Azure Marketplace website.

| **Parameter** | **Description** | **Web** | **Image** | **Video** | **News** | **Spell** | **Related** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| $top | Specifies the number of results to return. The default is 50 for web, image, and video searches, 15 for news. |  |  |  |  |  |  |
| $skip | Specifies the offset requested for the starting point of returned results. The default is zero. |  |  |  |  |  |  |
| $format | Specifies the format of the OData response. Current options are Atom (for XML) or JSON. Default: Atom. |  |  |  |  |  |  |
| Query | Bing Search query. The query can contain any valid query text that the Bing Engine supports. |  |  |  |  |  |  |
| Market | If the parameter is not specified, the API tries to determine an applicable market by using the IP address of the request, cookies, and other factors. |  |  |  |  |  |  |
| Latitude | A north/south coordinate value. Valid input values are: –90 to 90. Default: (none) |  |  |  |  |  |  |
| Longitude | An east/west coordinate value. Valid input values  are: –180 to 180. Default: (none) |  |  |  |  |  |  |
| Adult | Used for filtering sexually explicit content. If you don’t specify this parameter, the API will use the default level for the specified market. |  |  |  |  |  |  |
| Options | Specifies general options. See next section for more detail. |  |  |  |  |  |  |
| WebSearch  Options | Specifies web-specific options. See the Bing Search API options section for more detail. |  |  |  |  |  |  |
| WebFileType | A file name extension that specifies the type of file to return in a result set. Default: (none). If you don’t specify an extension, Bing gets any document relevant to the query. |  |  |  |  |  |  |
| ImageFilters | An array of strings that filter the response returned by the API based on any combination of size, aspect, color, style, and face. Default: (none) |  |  |  |  |  |  |
| VideoFilters | An array of strings that filter the response returned by the API based on any combination of duration, aspect, and resolution. Default: (none) |  |  |  |  |  |  |
| VideoSortBy | Specifies the sort order of the video results. Default: relevance. |  |  |  |  |  |  |
| NewsCategory | Category of news for which to provide results. If no results match the input provided for this parameter, the API returns the default set of results, which is news that belongs to a mix of categories. This parameter is applicable only in the en-US market. |  |  |  |  |  |  |
| NewsLocationOverride | Overrides Bing location detection. Applicable only in the en-US market. |  |  |  |  |  |  |
| NewsSortBy | Specifies the sort order of the news results. Default: relevance. |  |  |  |  |  |  |

# Bing Search API option parameters

When you construct a query to the Bing Search API, you can also include the *Options* and *WebSearchOptions* parameters to modify the behavior of the search.

## Options

Specifies general options. You can use this parameter with all service operations. It can contain the following values:

* DisableLocationDetection – Prevents Bing from inferring location from the terms of a query.
* EnableHighlighting – Causes Bing to insert special characters in the results that identify the beginning and the end of a query term that appears in results. The characters are E000 and E001, respectively. Although you can use this option with any service operation, the special characters are inserted only for Web and News results.

To specify multiple options, concatenate the option names with a plus (+) character. For example:

https://api.datamarket.azure.com/Bing/Search/Web?Query=%27Xbox%27&Options=%27DisableLocationDetection%2BEnableHighlighting%27

Notice that the value of the *Options* parameter is surrounded by single quotation marks which have been URL encoded along with the rest of the string. All string parameter values must be handled this way.

## WebSearchOptions

Specifies web-specific options. You can use this parameter only with the Web service operation or when using a Composite service operation that contains a web request. It can contain the following values:

* DisableHostCollapsing – Prevents Bing from suppressing results from the same top-level URL for a request.
* DisableQueryAlterations – Prevents Bing from altering the query string. Such alteration may have been done to correct apparent spelling error in the original query string.

As with general options, you can specify multiple web search options by concatenating the option names with a plus (+) character. For example:

https://api.datamarket.azure.com/Bing/Search/Web?Query=%27Xbox%27&WebSearchOptions=%27DisableQueryAlterations%2BDisableHostCollapsing%27

# Code Samples

The following code samples produce requests for different service operations. These code samples are written in C# and demonstrate how to:

* Send a request to the Bing Search API using the different service operations.
* Receive and display the results.

The code samples are organized as follows:

* [Preparing Code Sample for Service Operation](#_Prepare_Code_Sample)
* [Executing a Web Service Operation](#_Executing_a_Web)
* [Executing an Image Service Operation](#_Executing_an_Image)
* [Executing a Video Service Operation](#_Executing_a_Video)
* [Executing a News Service Operation](#_Executing_a_News)
* [Executing a SpellingSuggestions Service Operation](#_Executing_a_SpellingSuggestions)
* [Executing a RelatedSearch Service Operation](#_Executing_a_RelatedSearch)
* [Executing a Composite Service Operation](#_Executing_a_Composite)

The code samples are followed by three example web applications:

* [Creating a jQuery/ AJAX web application](#_Creating_a_jQuery)
* [Creating a C++ App for Windows 8](#_Creating_a_C++)
* [Creating an Objective-C Console Application](#_Creating_an_Objective-C)

## 

## Prepare Code Sample for Service Operation

1. Create a C# console application.
2. Download the [.NET Framework C# Service Proxy Class Library](https://datamarket.azure.com/dataset/explore/getproxy/5ba839f1-12ce-4cce-bf57-a49d98d29a44).
3. Place the downloaded file (BingSearchContainer.cs) in the same directory as your application.
4. Add the BingSearchContainer.cs file to your project.
5. Add a reference to *System.Data.Services.Client*.

## Executing a Web Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the Web service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 10 results.  var webQuery =  bingContainer.Web(query, null, null, market, null, null, null, null);  webQuery = webQuery.AddQueryOption("$top", 10);  // Run the query and display the results.  var webResults = webQuery.Execute();  foreach (var result in webResults)  {  Console.WriteLine("{0}\n\t{1}", result.Title, result.Url);  }  }  } |

## Executing an Image Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the Image service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 10 results.  var imageQuery =  bingContainer.Image(query, null, market, null, null, null, null);  imageQuery = imageQuery.AddQueryOption("$top", 10);  // Run the query and display the results.  var imageResults = imageQuery.Execute();  foreach (var result in imageResults)  {  Console.WriteLine("{0}\n\t{1}", result.Title, result.MediaUrl);  }  }  } |

## Executing a Video Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the Video service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 10 results.  var videoQuery =  bingContainer.Video(query, null, market, null, null, null, null, null);  videoQuery = videoQuery.AddQueryOption("$top", 10);  // Run the query and display the results.  var videoResults = videoQuery.Execute();  foreach (var result in videoResults)  {  int runTime = (result.RunTime != null) ? (int)result.RunTime : 0;  int totalSecs = runTime / 1000;  int mins = totalSecs / 60;  int secs = totalSecs % 60;  Console.WriteLine("{0} ({1:d2}:{2:d2})\n\t{3}",  result.Title, mins, secs, result.MediaUrl);  }  }  } |

## Executing a News Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the News service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Get news for science and technology.  string newsCat = "rt\_ScienceAndTechnology";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 10 results.  var newsQuery =  bingContainer.News(query, null, market, null, null, null, null, newsCat, null);  newsQuery = newsQuery.AddQueryOption("$top", 10);  // Run the query and display the results.  var newsResults = newsQuery.Execute();  foreach (var result in newsResults)  {  Console.WriteLine("{0}-{1}\n\t{2}",  result.Source, result.Title, result.Description);  }  }  } |

## Executing a SpellingSuggestions Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the SpellingSuggestions service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the (misspelled) query expression.  string query = "Xbo Liive Gmes";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query.  var spellQuery =  bingContainer.SpellingSuggestions(query, null, market, null, null, null);  // Run the query and display the results.  var spellResults = spellQuery.Execute();  foreach (var result in spellResults)  {  Console.WriteLine(result.Value);  }  }  } |

## Executing a RelatedSearch Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the RelatedSearch service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 5 results.  var relatedQuery =  bingContainer.RelatedSearch(query, null, market, null, null, null);  relatedQuery = relatedQuery.AddQueryOption("$top", 5);  // Run the query and display the results.  var relatedResults = relatedQuery.Execute();  foreach (var result in relatedResults)  {  Console.WriteLine("{0}\n\t{1}", result.Title, result.BingUrl);  }  }  } |

## Executing a Composite Service Operation

|  |
| --- |
| **C#** |
| using System;  using System.Net;  // Bing Search API code sample that demonstrates the Composite service operation.  static class Program  {  // Replace this value with your account key.  private const string AccountKey = "YourAccountKey";  static void Main(string[] args)  {  try  {  MakeRequest();  }  catch (Exception ex)  {  string innerMessage =  (ex.InnerException != null) ?  ex.InnerException.Message : String.Empty;  Console.WriteLine("{0}\n{1}", ex.Message, innerMessage);  }  }  static void MakeRequest()  {  // This is the query expression.  string query = "Xbox Live Games";  // Create a Bing container.  string rootUrl = "https://api.datamarket.azure.com/Bing/Search";  var bingContainer = new Bing.BingSearchContainer(new Uri(rootUrl));  // The market to use.  string market = "en-us";  // The composite operations to use.  string operations = "web+image+video";  // Configure bingContainer to use your credentials.  bingContainer.Credentials = new NetworkCredential(AccountKey, AccountKey);  // Build the query, limiting to 5 results (per service operation).  var compositeQuery =  bingContainer.Composite(operations, query, null, null, market,  null, null, null, null, null,  null, null, null, null, null);  compositeQuery = compositeQuery.AddQueryOption("$top", 5);  // Run the query and display the results.  var compositeResults = compositeQuery.Execute();  // Object compositeResults is IEnumerable<ExpandableSearchResult>  // but there's only one ExpandableSearchResult object in the collection.  foreach(var cResult in compositeResults)  {  // Display web results.  foreach(var result in cResult.Web)  {  Console.WriteLine("{0}\n\t{1}", result.Title, result.Url);  }  // Display Image results.  foreach (var result in cResult.Image)  {  Console.WriteLine("{0}\n\t{1}", result.Title, result.MediaUrl);  }  // Display video results.  foreach (var result in cResult.Video)  {  int runTime = (result.RunTime != null) ? (int)result.RunTime : 0;  int totalSecs = runTime / 1000;  int mins = totalSecs / 60;  int secs = totalSecs % 60;  Console.WriteLine("{0} ({1:d2}:{2:d2})\n\t{3}",  result.Title, mins, secs, result.MediaUrl);  }  // This query used "web+image+video" - If you used other  // service operations, you could iterate through them also.  }  }  } |

## Creating a jQuery / AJAX web application

As a high-level overview, here are the pieces you’ll implement:

1. A simple HTML page with a form to let the user enter search text.
2. A simple style sheet.
3. A client-side JavaScript piece that leverages the jQuery library to issue a search request based on your user’s input, parses the results from Bing, and modifies the DOM to display the results to the user.
4. A server-side PHP proxy to receive the search request, relay the request to the Bing Search API, and return the results to the client JavaScript.

#### *Getting Started*

To get started, create a folder where you can place the files that belong to this application. For this tutorial, all files go in the same folder. Here’s a list the files you’ll have with the finished application.

|  |  |
| --- | --- |
| **File** | **Description** |
| bing.html | The webpage with a simple form to enter a search request. |
| bing.css | A supporting style sheet. |
| bing.js | Client-side JavaScript to handle sending the search request to the server and displaying the results. |
| jquery-1.7.2.js | jQuery library code. You can download this file from the [jQuery web site](http://jquery.com). Your downloaded version may be different. If so, you’ll need to modify the script reference in the HTML to access the version. If you prefer, you can access the jQuery library through an external site. |

#### *Creating the HTML Page*

The HTML page for your application contains references to a simple style sheet, the jQuery source file, and the application-specific JavaScript file. It also has a simple search form where you’ll be able to choose between a web and an image search. Here’s what the HTML looks like.

|  |
| --- |
| **Example 1 – bing.html** |
| <html>  <head>  <title>Bing Search Tester (Ajax)</title>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <link href="bing.css" rel="stylesheet" type="text/css" />  <!-- Load jQuery locally. Several sites also let you load this externally if you prefer -->  <script language="JavaScript" type="text/javascript" src="jquery-1.7.2.js"></script>  <script language="JavaScript" type="text/javascript" src="bing.js"></script>  </head>  <body>  <div id="wrapper">  <h1>Bing Search Tester (Ajax)</h1>  <form id="my\_form">  <label for="service\_op">Service Operation</label>  <input name="service\_op" type="radio" value="Web" CHECKED /> Web  <input name="service\_op" type="radio" value="Image" /> Image  <label for="query">Query</label>  <input id="query" name="query" type="text" size="60" /><br /><br />  <input id="bt\_search" name="bt\_search" type="button" value="Search" />  </form>  <!-- Results will be placed into the following container. -->  <div id="results"></div>  </div>  </body>  </html> |

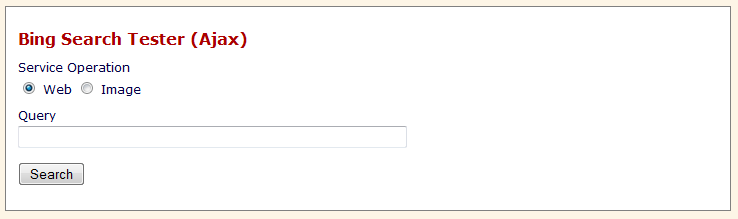
Fairly standard. One thing to note is that the page specifies UTF-8 encoding. You need this to correctly display the results from the Bing Search API. Save this web page as bing.html.

#### *Creating the Style Sheet*

The following is a style sheet you can use for the web page. Save the style sheet as bing.css.

|  |
| --- |
| **Example 2 – bing.css** |
| **body** {  background-color: #FDF5E6;  color: #000144;  font-family: verdana, arial, helvetica, sans-serif;  font-size: smaller;  margin: 0;  padding: 0;  }  **h1** {  font-size: 130%;  color: #AA0000;  }  **label** {  display: block;  margin-top: 12px;  margin-bottom: 4px;  }  #**wrapper** {  background-color: #FFFFFF;  border: 1px solid #808080;  font-size: 95%;  margin: 11px auto;  padding: 12px;  text-align: left;  width: 700px;  } |

When you load the web page in your browser, it should resemble the following image.



Creating the JavaScript

When you load the web page, the bing.js file that the web page references is loaded. You use this file to attach a click handler to the Search button. The click handler sends the request to the PHP proxy and manipulates the DOM to show the results.

The following code shows the contents of the supporting JavaScript. Save this in your application folder as bing.js.

|  |
| --- |
| **Example 3 – bing.js** |
| /\*! Bing Search Helper v1.0.0 - requires jQuery v1.7.2 \*/  $(function()  {  // Attaches a click handler to the button.  $('#bt\_search').click(function(e)  {  e.preventDefault();  // Clear the results div.  $('#results').empty();  var query = $('#query').val();  var serviceOp = $('input[name=service\_op]:checked', '#my\_form').val();  if (query)  search(query, serviceOp);  });  // Performs the search.  function search(query, serviceOp)  {  // Establish the data to pass to the proxy.  var data = { q: query, sop: serviceOp, market: 'en-us' };    // Calls the proxy, passing the query, service operation and market.  $.getJSON('bing\_proxy.php', data, function(obj)  {  if (obj.d !== undefined)  {  var items = obj.d.results;  for (var k = 0, len = items.length; k < len; k++)  {  var item = items[k];  switch (item.\_\_metadata.type)  {  case 'WebResult':  showWebResult(item);  break;  case 'ImageResult':  showImageResult(item);  break;  }  }  }  });  }  // Shows one item of Web result.  function showWebResult(item)  {  var p = document.createElement('p');  var a = document.createElement('a');  a.href = item.Url;  $(a).append(item.Title);  $(p).append(item.Description);  // Append the anchor tag and paragraph with the description to the results div.  $('#results').append(a, p);  }    // Shows one item of Image result.  function showImageResult(item)  {  var p = document.createElement('p');  var a = document.createElement('a');  a.href = item.MediaUrl;  // Create an image element and set its source to the thumbnail.  var i = document.createElement('img');  i.src = item.Thumbnail.MediaUrl;  // Make the object that the user clicks the thumbnail image.  $(a).append(i);  $(p).append(item.Title);  // Append the anchor tag and paragraph with the title to the results div.  $('#results').append(a, p);  }  }); |

#### *Creating the PHP Proxy*

The last piece that you’ll need is the PHP proxy. This server-side component of your application accepts the parameters passed to it by the AJAX component (invoked by the $.getJSON function), relays the request to the Bing Search API and finally, returns the JSON to the JavaScript for the rest of the processing.

The PHP code here is similar to the example in *Bing Search API Quick Start Guide*. Main differences are:

* With the proxy, you get the arguments for the query through a GET operation instead of a POST.
* You don’t have to decode the JSON from the Bing Search API. You just echo the JSON directly back to the caller; the jQuery library takes care of the decoding.

The following shows the PHP code for the proxy. Replace the value of $acctKey with your account key and save the code in your application folder as bing\_proxy.php

|  |
| --- |
| **Example 4 – bing\_proxy.php** |
| <?php  /\*\*\*\*  \* PHP proxy for using the Bing Search API with AJAX  \*/  $acctKey = 'YourAccountKey';  $rootUri = 'https://api.datamarket.azure.com/Bing/Search';  // Get the query. Default to 'sushi'.  $query = ($\_GET['q']) ? $\_GET['q'] : 'sushi';  // Get the service operation. Default to Web.  $serviceOp = ($\_GET['sop']) ? $\_GET['sop'] : 'Web';  // Get the market. Default to en-us.  $market = ($\_GET['market']) ? $\_GET['market'] : 'en-us';  // Encode the query along with the single quotation marks that must surround it.  $query = urlencode("'$query'");  // Encode the market along with the single quotation marks that must surround it.  $market = urlencode("'$market'");  // Construct the full URL for the query.  $requestUri = "$rootUri/$serviceOp?\$format=json&Query=$query&Market=$market";  // Encode the credentials and create the stream context.  $auth = base64\_encode("$acctKey:$acctKey");  $data = array(  'http' => array(  'request\_fulluri' => true,  // ignore\_errors can help debug – remove for production. This option added in PHP 5.2.10  'ignore\_errors' => true,  'header' => "Authorization: Basic $auth")  );  $context = stream\_context\_create($data);  // Get the response from Bing.  $response = file\_get\_contents($requestUri, 0, $context);  // Send the response back to the browser.  echo $response;  ?> |

#### *Debugging Tip*

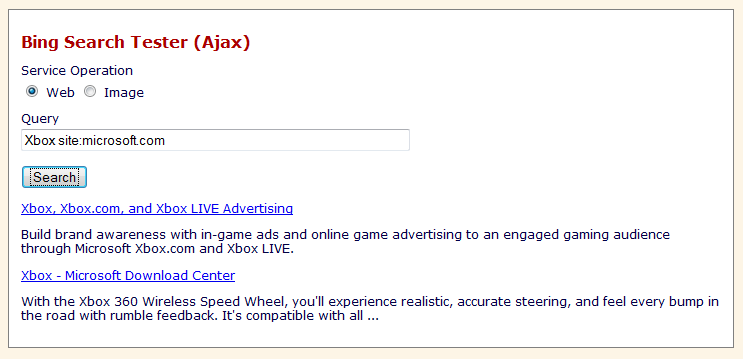
Debugging an AJAX application can sometimes be tricky, especially if there’s a problem with the server-side piece that’s keeping you from getting a proper response. One thing you can do is access the server-side PHP directly from your browser. For example, you can try entering the following in the address bar of your browser.

http://example.com/bing\_proxy.php?q=Xbox&sop=Image

If there’s a problem with the script (for example, syntax error), you’ll be able to figure it out pretty easily. If everything’s working correctly, you’ll see the JSON in your browser.

#### *Running Your Application*

Once you have all the pieces put together, you can run your application. You should see results similar to the following when doing a Web search.



Or similar to the following when doing an Image search. 

## Creating a C++ App for Windows 8

This tutorial shows you how to create a C++ app for Windows 8 that gets and displays web results from the Bing Search API.

#### Prerequisites

* A computer running the Windows 8 operating system.
* Microsoft Visual Studio 2012 or Visual Studio 2012 Express.
* Some working knowledge of C++.
* Familiarity with Visual Studio.

#### Overview

Here’s what the application we’ll create will do:

* Retrieve an RSS feed of Microsoft Knowledge Base articles and show it in a list view.
* When you select one of the KB articles, create a Bing query to find other webpages that mention the article.
* When you select one of the related webpages, display it in a web view.

To do this, we’ll create the following pieces:

* **A main page**.

For this tutorial, we’ll use the blank application template and create the XAML markup by hand. We’ll also just use one page for our application to focus on the coding logic we need to handle the network requests.

* **Code to retrieve the information from the RSS feed**.

We can use classes in the [Windows::Web::Syndication](http://msdn.microsoft.com/en-us/library/windows/apps/windows.web.syndication.syndicationclient) namespace to do this.

* **Classes to hold the RSS feed data**, which we’ll bind to the list view.
* **Classes to hold web results from the Bing Search API**.
* **Classes that use the** [**IXMLHTTPRequest2**](http://msdn.microsoft.com/en-us/library/hh831151.aspx) **and** [**IXMLHTTPRequest2Callback**](http://msdn.microsoft.com/en-us/library/hh831152.aspx) **interfaces to give us access to basic HTTP request operations**.

Although C++ does provide some higher level access to managed code, you can’t use the HttpClient class as you can with C#.

* **A method to parse the JSON results from Bing and populate our web result data structures with the parsed values**.

#### Getting Started

To get started, open Visual Studio 2012 and create a new Visual C++ Windows app by using the blank application template. Name the project BingSearch.

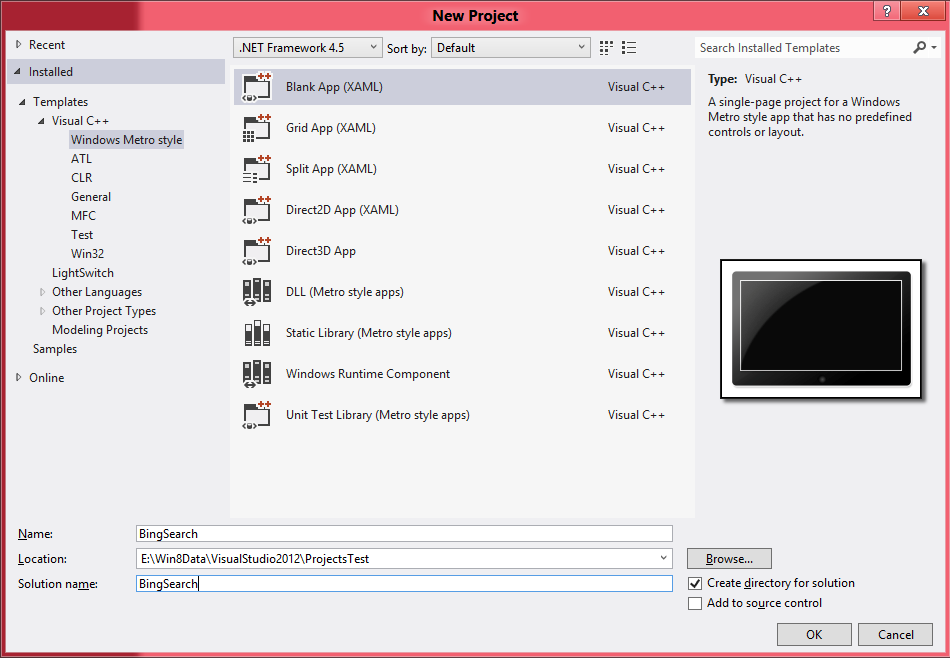


Figure 1 - Creating a new project

This sets up your project and creates several files.

In Solution Explorer, you should see the following list of files:

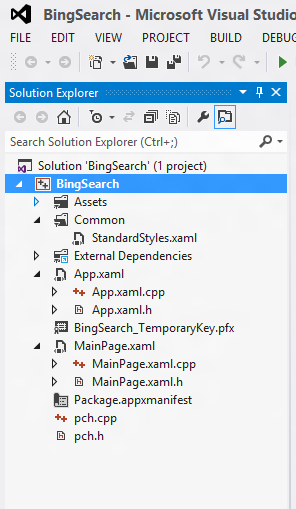


Figure 2 – Files created by Visual Studio

#### Files You Edit

Let's look first at the project files you can edit.

|  |  |
| --- | --- |
| **File name** | **Description** |
| App.xaml  MainPage.xaml | The XAML markup files that represent the App object and the default page for the UI. You can modify these files by using the Visual Studio XAML Designer, Microsoft Expression Blend, or another XAML designer tool. For this tutorial, all XAML modifications will be in MainPage.xaml. |
| App.xaml.h  App.xaml.cpp  MainPage.xaml.h  MainPage.xaml.cpp | The user-editable header and implementation code-behind files for the App and MainPage classes. These classes correspond to the XAML trees in App.xaml and MainPage.xaml, respectively. The MainPage.xaml.h and MainPage.xaml.cpp files are where you add event handlers and other custom program logic that's related to this page. Member variables in the App class are in scope throughout your app. Variables in the Page class are in scope only in that page. For this tutorial, we will not make any changes to App.xaml.h or App.xaml.cpp. |
| StandardStyles.xaml | Contains predefined item templates, styles, and other elements that define the look and feel of an app for Windows 8. Don't modify the styles and templates in-place. However, you can base custom styles on these (by using the BasedOn attribute), or add new styles. |
| Package.appxmanifest | Contains metadata that describes your app, for example: display name, description, logos, and capabilities. When you click this item, it opens in the App Manifest Designer. |
| pch.h  pch.cpp | Typical C++ precompiled header files. We’ll add several #include directives and a linker directive to the pch.h file. |

There are other files that you can edit, such as image files, but for our purposes, we’ll be concentrating on the files listed in the above table plus adding our own files when we create our custom classes.

#### Specifying App Capabilities

An app for Windows 8 runs in a security container that has limited access to the file system, network resources, and hardware. When you create a new app for Windows, you specify which capabilities your app needs: internet, microphone, web cam, and so forth. For our app, we need only internet capability. This is the default when you create a new application.

#### To view or modify the capabilities of your app

1. In Solution Explorer, open Package.appxmanifest.

The file opens in the App Manifest Designer.

1. Click the **Capabilities** tab.
2. The **Internet (Client)** check box is already selected. If we needed some other capability, we’d have to select it manually in the App Manifest Designer.

#### Creating XAML Layout for the App

The following shows the default XAML created by Visual Studio for our main page. If you create applications using Windows Presentation Foundation (WPF), this will look familiar. We have a Page class from which our page derives and a bare Grid layout control.

|  |
| --- |
| **Example 1 – MainPage.xaml** |
| <Page  x:Class="BingSearch.MainPage"  IsTabStop="false"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:local="using:BingSearch"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  mc:Ignorable="d">  <!-- Main grid -->  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">  </Grid>  </Page> |

We’ll be adding rows, columns, and other nested Grid objects to the XAML to create the layout shown in the following diagram.

|  |  |
| --- | --- |
| RSS Feed Title | |
| RSS Feed Results  ListView element | Status |
| Bing Search Results  ListView element |
| Return Navigation |
| Web View  WebView element |

Figure 3 – Application layout diagram

The ListView we use for the Bing search results and the WebView element are both in the same Grid column. However, later we’ll add coding logic to hide one while the other is displayed and vice-versa.

First we need to add some row and column definitions to our main Grid element.

|  |
| --- |
| **Example 2 – MainPage.xaml** |
| <Page  x:Class="BingSearch.MainPage"  IsTabStop="false"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:local="using:BingSearch"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  mc:Ignorable="d">  <!-- Main grid -->  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">  <Grid.RowDefinitions>  <RowDefinition Height="auto"/>  <RowDefinition Height="\*"/>  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="2\*"/>  </Grid.ColumnDefinitions>  </Grid>  </Page> |

This partitions our main Grid into two rows and two columns. For the left side of the layout, we’ll add a TextBlock element to display the title of the RSS feed and a ListView element to show the items from the feed. Copy the following XAML and paste it into MainPage.xaml just below the column and row definitions for the main Grid element.

|  |
| --- |
| **Example 3 – MainPage.xaml (left side of layout)** |
| <!-- Start left column -->  <!-- Text block to display the title of the RSS feed -->  <TextBlock x:Name="RssTitle"  Style="{StaticResource HeaderTextStyle}"  Grid.ColumnSpan="2" Margin="16"/>  <!-- ListView to hold the items from the RSS feed -->  <ListView Grid.Row="1" Margin="10"  ItemsSource="{Binding Path=Items}">  <ListView.ItemTemplate>  <DataTemplate>  <StackPanel>  <TextBlock Text="{Binding Path=Title}"  Style="{StaticResource TitleTextStyle}" />  <TextBlock Text="{Binding Path=Summary}"  Style="{StaticResource CaptionTextStyle}" />  <TextBlock Text="{Binding Path=KBID}"  Style="{StaticResource CaptionTextStyle}" />  </StackPanel>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  <!-- End left column --> |

#### Data binding

Notice that the ListView element uses data binding markup to establish a relationship between the UI and its underlying data source. When you establish a binding and the data source changes, the UI elements that are bound to the data source reflect the changes automatically. With data-binding, all or almost all of the code is written in the XAML file rather than in the code-behind.

The significance of the Path parameter specified in the binding statement depends on the data context of the element that declares the binding. For instance, our ListView element specifies the name “Items” as its binding path. At run time, the value of DataContextObject->Items will be used to populate our ListView element. If the data context object doesn’t have an Items property, no error occurs. This means you could run the app at this point and there wouldn’t be any errors. Of course, there wouldn’t be any data displayed either.

You establish a data context for an element by assigning an object to its DataContext property. We’ll do that later in our code. Child elements inherit the data context of their parent element or you can override the inheritance by assigning a different object to the child’s DataContext property. We’ll do that later in our code as well.

Right side of the app layout

Next we’ll add a nested Grid element to hold the elements that we’ll need on the right side of our layout. Copy the following XAML and paste it into MainPage.xaml just below the comment we inserted earlier that says “End left column”.

|  |
| --- |
| **Example 4 – MainPage.xaml (grid for right side of layout)** |
| <!-- Start right column -->  <!-- Nested grid to hold the search results -->  <Grid Grid.Row="1" Grid.Column="1">  <Grid.RowDefinitions>  <RowDefinition Height="auto" />  <RowDefinition Height="\*" />  </Grid.RowDefinitions>  <!-- This will display a simple status message -->  <TextBlock x:Name="BingStatus"  Style="{StaticResource TitleTextStyle}"  Foreground="Red" TextAlignment="Center" Margin="20"/>  <!-- ListView to hold the Bing search results -->  <ListView x:Name="BingListView" Grid.Row="1"  ItemsSource="{Binding Path=Items}">  <ListView.ItemTemplate>  <DataTemplate>  <StackPanel>  <TextBlock Text="{Binding Path=Title}"  Style="{StaticResource TitleTextStyle}" />  <TextBlock Text="{Binding Path=Description}"  Style="{StaticResource CaptionTextStyle}" />  <TextBlock Text="{Binding Path=DisplayUrl}"  Style="{StaticResource CaptionTextStyle}" />  </StackPanel>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  <!-- Grid to hold the WebView and the button to collapse back to the list -->  <Grid x:Name="BrowserGrid" Grid.Row="1" Visibility="Collapsed">  <Grid.RowDefinitions>  <RowDefinition Height="auto"/>  <RowDefinition Height="\*"/>  </Grid.RowDefinitions>  <!-- Button to perform the collapse -->  <Button Style="{StaticResource BackButtonStyle}" />  <WebView x:Name="WebContentView" Grid.Row="1" />  </Grid>  </Grid>  <!-- End right column --> |

#### Adding Element Event Handlers

Now that we have our XAML for the app layout, we’ll add some event handlers to certain elements. We need to handle the following element events:

* The Loaded event of the main page. This is where we’ll get the RSS data to populate the ListView element.
* The SelectionChanged event of the ListView element that holds the items from the RSS feed. When the selection changes, we’ll perform the Bing search to find articles related to the selected item.
* The SelectionChanged event of the ListView element that holds the search results from Bing. When the selection changes, we’ll hide the Bing search results, display the WebView element, and navigate it to the selected page.
* The Click event of the button that collapses the Grid element that holds the WebView element. With this, you’ll collapse the WebView element and show the Bing search results.

#### To add the event handlers for the MainPage class

1. In MainPage.xaml:
   1. Add the attribute Loaded="MainPage\_Loaded" to the opening Page element

|  |
| --- |
| **Example 5 – MainPage.xaml (adding Loaded event)** |
| <Page  x:Class="BingSearch.MainPage"  IsTabStop="false"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:local="using:BingSearch"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  mc:Ignorable="d"  Loaded="MainPage\_Loaded"><!-- Event handler attribute added --> |

* 1. Add the attribute SelectionChanged="FeedListView\_SelectionChanged” to the ListView element that holds the items from the RSS feed.

|  |
| --- |
| **Example 6 – MainPage.xaml.h (adding the first SelectionChanged event)** |
| <!-- ListView to hold the items from the RSS feed -->  <ListView Grid.Row="1" Margin="10"  ItemsSource="{Binding Path=Items}"  SelectionChanged="FeedListView\_SelectionChanged"><!-- Added --> |

* 1. Add attribute SelectionChanged="WebListView\_SelectionChanged" to the ListView element that holds the Bing search results.

|  |
| --- |
| **Example 7 – MainPage.xaml (adding the second SelectionChanged event)** |
| <!-- ListView to hold the Bing search results -->  <ListView x:Name="BingListView" Grid.Row="1"  ItemsSource="{Binding Path=Items}"  SelectionChanged="BingListView\_SelectionChanged"><!-- Added --> |

* 1. Add the attribute Click="HideBrowser\_Click" to the Button element.

|  |
| --- |
| **Example 8 – MainPage.xaml (adding the Click event)** |
| <!-- Button to perform the collapse -->  <Button Style="{StaticResource BackButtonStyle}"  Click="HideBrowser\_Click" /><!-- Added --> |

1. Add the method signatures for the event handlers to MainPage.xaml.h.

|  |
| --- |
| **Example 9 – MainPage.xaml.h** |
| //  // MainPage.xaml.h  // Declaration of the MainPage class.  //  #pragma once  #include "MainPage.g.h"  namespace BingSearch  {  public ref class MainPage sealed  {  private:  // Event handler signatures.  void MainPage\_Loaded(  Platform::Object^ sender,  Windows::UI::Xaml::RoutedEventArgs^ e);  void FeedListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e);  void BingListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e);  void HideBrowser\_Click(  Platform::Object^ sender,  Windows::UI::Xaml::RoutedEventArgs^ e);  public:  // Constructor. Added by Visual Studio template.  MainPage();  protected:  // Added by Visual Studio template.  virtual void OnNavigatedTo(  Windows::UI::Xaml::Navigation::NavigationEventArgs^ e) override;  };  } |

1. Add the method implementation bodies for the event handlers to MainPage.xaml.cpp.

|  |
| --- |
| **Example 10 – MainPage.xaml.cpp (event handler implementation bodies)** |
| /\*  Invoked when the page is loaded.  \*/  void MainPage::MainPage\_Loaded(  Platform::Object^ sender,  Windows::UI::Xaml::RoutedEventArgs^ e)  {  (void) sender, e; // Unused parameters.  // TODO: Load RSS feed.  }  /\*  Invoked when the selection changes on the RSS feed list.  \*/  void MainPage::FeedListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e)  {  // TODO:: Perform Bing Search on selected item.  }  /\*  Invoked when the selection changes on the Bing list.  \*/  void MainPage::BingListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e)  {  // TODO: Display the selected webpage.  }  /\*  Invoked when the user clicks the button to hide the WebView.  \*/  void MainPage::HideBrowser\_Click(  Platform::Object^ sender,  Windows::UI::Xaml::RoutedEventArgs^ e)  {  (void) sender, e; // Unused parameters.  BrowserGrid->Visibility = Windows::UI::Xaml::Visibility::Collapsed;  BingListView->Visibility = Windows::UI::Xaml::Visibility::Visible;  } |

So far, only the event handler for the Button::Click event is fully implemented. It simply hides the Grid element that holds the WebView and displays the ListView that holds the Bing search results. The other three event handlers require objects and methods that we haven’t implemented yet. As we proceed, we’ll add the necessary code into the method bodies of the remaining handlers.

At this point, you should be able to press F7 to successfully build your project and F5 to run it, although the app doesn’t yet display anything on the screen.

#### Getting the RSS Feed Data

Before we get the RSS feed data, we’ll create a simple class header to define an object to hold information about each feed item. For simplicity, we’ll also create a class in the same header file that contains a collection of individual feed items.

#### To create the FeedData.h file

1. On the **Project** menu, click **Add New Item**.
2. In the **New Item** dialog box, select **Visual C++**, and then select **Header File (.h)**.
3. Name the file **FeedData.h**.
4. Click **Add**.
5. Paste the following code into FeedData.h.

|  |
| --- |
| **Example 11 – FeedData.h** |
| #pragma once  #include "pch.h"  namespace BingSearch  {  // In C++, to be bindable, a class must be defined  // within a namespace and have BindableAttribute.  [Windows::UI::Xaml::Data::BindableAttribute]  public ref class FeedItem sealed  {  public:  FeedItem(void){}  ~FeedItem(void){}  property Platform::String^ Title;  property Platform::String^ KBID;  property Platform::String^ Summary;  property Windows::Foundation::Uri^ Link;  };  [Windows::UI::Xaml::Data::BindableAttribute]  public ref class FeedData sealed  {  public:  FeedData(void)  {  items = ref new Platform::Collections::Vector<FeedItem^>();  }  ~FeedData(void){}  property Windows::Foundation::Collections::IVector<FeedItem^>^ Items  {  Windows::Foundation::Collections::IVector<FeedItem^>^ get()  {  return items;  }  }  private:  Platform::Collections::Vector<FeedItem^>^ items;  };  } |

Note that the property names of a FeedItem object match the names that we used earlier in the XAML to declare our binding to the UI elements. Once we populate our collection of FeedItem objects, and complete the binding tasks, each corresponding property will appear in the UI.

Before we proceed, let’s take a look at the pch.h file. This file is included in other files within the project (such as MainPage.xaml.cpp) and is used to ensure that needed declarations are available throughout the app. When you open pch.h, you’ll see some standard include directives added by Visual Studio. Add an include directive for ppltasks.h. This header file provides declarations you can use to perform asynchronous and parallel processing tasks, which we’ll need soon. We’ll be adding other include directives and a linker directive as we continue.

|  |
| --- |
| **Example 12 – pch.h** |
| //  // pch.h  // Header for standard system include files.  //  #pragma once  // Include directives added by Visual Studio.  #include <collection.h>  #include "App.xaml.h"  // Our additional includes.  #include <ppltasks.h> |

#### Populating feed data

Now that we have our data structures in place, let’s implement the necessary code to fetch the RSS feed data and populate our FeedData object.

#### To populate the FeedData object from the RSS feed

1. In MainPage.xaml.h, add an include directive for FeedData.h. You should have include directives at the top of MainPage.xaml.h as shown in the following example.

|  |
| --- |
| **Example 13 – MainPage.xaml.h (include directives)** |
| //  // MainPage.xaml.h  // Declaration of the MainPage class.  //  #pragma once  // Includes added by Visual Studio.  #include "MainPage.g.h"  // Our additional include directives.  #include "FeedData.h" |

1. Also in MainPage.xaml.h, add the following declarations for the FeedData object and supporting method signatures below the event handler signatures that we added earlier.

|  |
| --- |
| **Example 14 – MainPage.xaml.h (FeedData and supporting methods)** |
| private:  // Event handler signatures.  // ...  // Feed data object and supporting methods.  FeedData^ feedData;  void GetFeedData(Platform::String^ feedUriString);  Platform::String^ GetKBID(Windows::Foundation::Uri^ uri); |

1. In the MainPage constructor (file MainPage.xaml.cpp), instantiate the FeedData object and assign it to the DataContext property.

|  |
| --- |
| **Example 15 – MainPage.xaml.cpp (constructor)** |
| /\*  Constructor.  \*/  MainPage::MainPage()  {  InitializeComponent();  this->feedData = ref new FeedData();  this->DataContext = this->feedData;  } |

1. Near the top of MainPage.xaml.cpp, add three additional namespaces as shown in the following example.

|  |
| --- |
| **Example 16 – MainPage.xaml.cpp (namespaces)** |
| // Namespaces added by Visual Studio.  using namespace BingSearch;  using namespace Platform;  using namespace Windows::Foundation;  using namespace Windows::Foundation::Collections;  using namespace Windows::UI::Xaml;  using namespace Windows::UI::Xaml::Controls;  using namespace Windows::UI::Xaml::Controls::Primitives;  using namespace Windows::UI::Xaml::Data;  using namespace Windows::UI::Xaml::Input;  using namespace Windows::UI::Xaml::Media;  using namespace Windows::UI::Xaml::Navigation;  // Our additional namespaces.  using namespace Windows::Web::Syndication;  using namespace concurrency;  using namespace std; |

1. Copy the following two methods that you declared in the header file of MainPage into the implementation file, MainPage.xaml.cpp.

|  |
| --- |
| **Example 17 – MainPage.xaml.cpp (methods to get the RSS feed data)** |
| /\*  Gets the feed data specified by feedUriStr.  \*/  void MainPage::GetFeedData(Platform::String^ feedUriStr)  {  // Create the SyndicationClient and the target URI.  SyndicationClient^ client = ref new SyndicationClient();  Uri^ feedUri = ref new Uri(feedUriStr);  // Create the async operation. feedOp is an  // IAsyncOperationWithProgress<SyndicationFeed^, RetrievalProgress>^  auto feedOp = client->RetrieveFeedAsync(feedUri);  // Create the task object and pass it the async operation.  // SyndicationFeed^ is the type of the return value  // that the feedOp operation will eventually produce.  concurrency::create\_task(feedOp)    // Create a continuation that will run when the first task completes.  // The continuation takes the return value of the first operation,  // and then defines its own asynchronous operation by using a lambda.  .then([this] (SyndicationFeed^ feed) -> void  {  // Assign the title of the feed.  RssTitle->Text = feed->Title->Text;  // Retrieve the individual items from the feed.  auto feedItems = feed->Items;  // Iterate over the items.  for(unsigned k = 0; k < feedItems->Size; k++)  {  auto item = feedItems->GetAt(k);  FeedItem^ feedItem = ref new FeedItem();  feedItem->Title = item->Title->Text;  feedItem->Summary = (item->Summary != nullptr) ?  item->Summary->Text : "(no summary)";  feedItem->Link = item->Links->GetAt(0)->Uri;  feedItem->KBID = GetKBID(feedItem->Link);  feedData->Items->Append(feedItem);  }  })  // The last continuation serves as an error handler. The  // call to get() will surface any exceptions that were raised  // at any point in the task chain.  .then([] (concurrency::task<void> t)  {  try  {  t.get();  }  catch(Platform::Exception^ e)  {  // TODO handle error. For example purposes  // we just output error to console.  OutputDebugString(String::Concat("Error: ", e->Message)->Data());  }  });  }  /\*  Helper method to extract the KB article number.  \*/  Platform::String^ MainPage::GetKBID(Uri^ uri)  {  Platform::String^ retVal = "KB 00000";  if (uri != nullptr)  {  std::wstring str(uri->RawUri->Data());  std::wstring::size\_type idx1, idx2;  idx1 = str.find(L"/kb/", 0);  if (idx1 != std::string::npos)  {  idx2 = str.find(L"/", idx1 + 4);  if (idx2 != std::string::npos)  {  std::wstring str2 = str.substr (idx1 + 4 , idx2 - idx1 - 4);  retVal = Platform::String::Concat(  "KB ", ref new Platform::String(str2.c\_str()));  }  }  }  return retVal;  } |

1. Finally, modify MainPage.xaml.cpp to call the GetFeedData method from the MainPage::Loaded event handler.

|  |
| --- |
| **Example 18 – MainPage.xaml.cpp (Loaded event handler)** |
| /\*  Invoked when the page is loaded.  \*/  void MainPage::MainPage\_Loaded(  Platform::Object^ sender,  Windows::UI::Xaml::RoutedEventArgs^ e)  {  (void) sender, e; // Unused parameters.  GetFeedData("http://support.microsoft.com/common/rss.aspx?rssid=14019");  } |

When you build and run the app, the display should resemble the following figure.



Figure 4 – Running the app

The Knowledge Base articles appear in the ListView control on the left side of the display. Note that the article titles also appear in the ListView element on the right side of the display. This is because we set the data context for the entire XAML tree in constructor of MainPage.

|  |
| --- |
| **Example 19 – MainPage.xaml.cpp (setting data context)** |
| this->DataContext = feedData; |

Recall that elements inherit data context from their parent element. In this case, the ListView element on the right side of the display also has a binding to a Title property. This will be used to display the titles of the Bing search results. Because we haven’t yet overridden the data context, for now we see the title of the Knowledge Base articles instead. Later, we’ll override the data context for the second ListView so that the titles will be those we receive from the Bing Search API.

#### Preparing to Access the Bing Search API

Now that we’re able to retrieve and display the RSS feed items, let’s take a look at how we’ll access the Bing Search API.

First, we’ll create a simple class header to define an object to hold a single Bing web result. This is very similar to the steps we used to create the FeedData.h file. Once again, for simplicity, we’ll also create a class in the same header file that contains a collection of individual web result items.

#### To create the WebResult.h file

1. On the **Project** menu, click **Add New Item**.
2. In the **New Item** dialog box, select **Visual C++**, and then select **Header File (.h)**.
3. Name the file **WebResult.h**.
4. Click **Add**.
5. Paste the following code into WebResult.h.

|  |
| --- |
| **Example 20 – WebResult.h** |
| #pragma once  #include "pch.h"  namespace BingSearch  {  // In C++, to be bindable, a class must be defined  // within a namespace and have BindableAttribute.  [Windows::UI::Xaml::Data::BindableAttribute]  public ref class WebResult sealed  {  public:  WebResult(void){}  ~WebResult(void){}  property Platform::String^ ID;  property Platform::String^ Title;  property Platform::String^ Description;  property Platform::String^ DisplayUrl;  property Windows::Foundation::Uri^ Url;  };  [Windows::UI::Xaml::Data::BindableAttribute]  public ref class WebResultCollection sealed  {  public:  WebResultCollection(void)  {  items = ref new Platform::Collections::Vector<WebResult^>();  }  ~WebResultCollection(void){}  property Windows::Foundation::Collections::IVector<WebResult^>^ Items  {  Windows::Foundation::Collections::IVector<WebResult^>^ get()  {  return items;  }  }  private:  Platform::Collections::Vector<WebResult^>^ items;  };  } |

Note once again that the property names of our WebResult object match the names that we used earlier in the XAML to declare our binding to the UI elements. Once we populate our collection of WebResult objects, and complete the binding tasks, each corresponding property will appear in the UI.

Next, we’ll declare and implement an instance of a WebResultCollection object and create two strings to hold information we’ll need for the Bing Search API.

#### To implement the web results object and other Bing variables

1. Add an include directive for WebResult.h to the include directives of MainPage.xaml.h. as shown in the following example.

|  |
| --- |
| **Example 21 – MainPage.xaml.h (include directives)** |
| //  // MainPage.xaml.h  // Declaration of the MainPage class.  //  #pragma once  // Include directives added by Visual Studio.  #include "MainPage.g.h"  // Our additional include directives.  #include "FeedData.h"  #include "WebResult.h" |

1. Also in MainPage.xaml.h, add declarations for the WebResultCollection object and two String objects to hold the Bing Search API service root and your account key. Add these below the FeedData objects and methods we added earlier.

|  |
| --- |
| **Example 22 – MainPage.xaml.h (WebResultCollection and other Bing vars)** |
| private:  // Event handler signatures.  // ...  // Feed data object and supporting methods.  // ...  // Web results collection and Bing information.  WebResultCollection^ webResults;  Platform::String^ serviceRoot;  Platform::String^ accountKey; |

In the MainPage constructor (file MainPage.xaml.cpp), instantiate the WebResultCollection object and assign it to the DataContext property of our Bing ListView element. Specify the service root for the Bing Search API and your account key.

|  |
| --- |
| **Example 23 – MainPage.xaml.cpp (constructor)** |
| /\*  Constructor.  \*/  MainPage::MainPage()  {  InitializeComponent();  this->feedData = ref new FeedData();  this->DataContext = this->feedData;  this->webResults = ref new WebResultCollection();  BingListView->DataContext = webResults;  // The service root for Bing.  this->serviceRoot = "https://api.datamarket.azure.com/Bing/Search";  // Replace this with your account key.  this->accountKey = "YourAccountKey";  } |

At this point, our app should build without errors or warnings. When you run the app, notice that the Knowledge Base titles no longer appear on the right side of the display. This is because we assigned the BingListView->DataContext property, thereby overriding the value it had formerly inherited from its parent element. Since the data context we assigned hasn’t yet been populated, the right side of the display is empty.

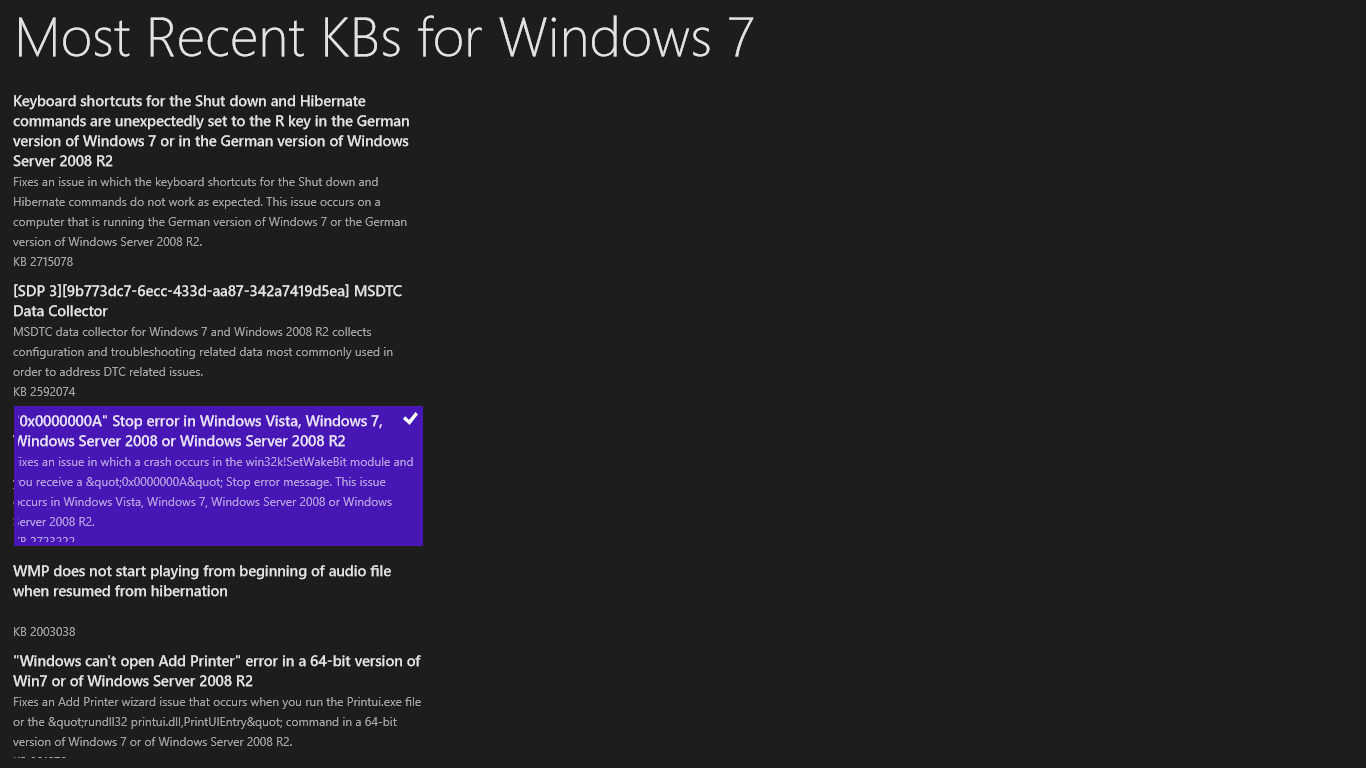


Figure 5 – Running the app

#### Creating the HttpRequest and HttpRequestCallback classes

Next, we need to create two custom classes, HttpRequest and HttpRequestCallback, that implement the [IXMLHTTPRequest2](http://msdn.microsoft.com/en-us/library/hh831151.aspx) and [IXMLHTTPRequest2Callback](http://msdn.microsoft.com/en-us/library/hh831152.aspx) interfaces, respectively. With these classes, we can create a request to the Bing Search API and receive the response.

For simplicity, we’ll create the files for the HttpRequest class using the Visual Studio class wizard, and implement HttpRequestCallback in the same file as HttpRequest. This approach works because while HttpRequestCallback is used in the process of making a request, your other code doesn’t refer to it directly.

#### To create the HttpRequest and HttpRequestCallback classes

1. On the **Project** menu, click **Add Class**. The class wizard appears. Make sure **C++ Class** is selected and then click **Add**.
2. On the next screen, type **HttpRequest** in the **Class name** box, and then click **Finish**. Visual Studio fills in the header and implementation file names automatically.
3. Paste the following code into HttpRequest.h, replacing the code that was generated by the wizard. We don’t need a constructor or a destructor; we’ll only have a single static method.

|  |
| --- |
| **Example 24 – HttpRequest.h** |
| #pragma once  /\*  Utility class for performing asynchronous HTTP requests.  \*/  class HttpRequest  {  public:  /\*  Starts a download of the specified URI.  The returned task produces a tuple of the status code of the operation  and the HTTP response.  \*/  static concurrency::task<std::tuple<HRESULT, std::wstring>> StartDownload(  const std::wstring& method,  const std::wstring& uri,  const std::wstring& user,  const std::wstring& password,  concurrency::cancellation\_token cancellationToken);  }; |

1. Open HttpRequest.cpp and remove the constructor and destructor implementations that were added by the wizard. Add three namespace statements just below the include directives that were added by the class wizard.

|  |
| --- |
| **Example 25 – HttpRequest.cpp (include directives and namespace statements)** |
| // Include directives added by Visual Studio.  #include "pch.h"  #include "HttpRequest.h"  // Our additional namespaces.  using namespace concurrency;  using namespace std;  using namespace Microsoft::WRL; |

1. Implement HttpRequestCallback by pasting the following code into HttpRequest.cpp below the namespace statements you added in the previous step.

|  |
| --- |
| **Example 26 – HttpRequest.cpp (implementation of HttpRequestCallback)** |
| /\*  Implementation of IXMLHTTPRequest2Callback.  \*/  class HttpRequestCallback  : public RuntimeClass<RuntimeClassFlags<ClassicCom>,  IXMLHTTPRequest2Callback, FtmBase>  {  public:  HttpRequestCallback(IXMLHTTPRequest2\* request,  cancellation\_token cancellationToken) :  m\_request(request),  m\_cancellationToken(cancellationToken),  m\_aborted(false)  {  // Register a callback function that cancels the HTTP operation when  // the cancellation token is canceled.  if (m\_cancellationToken != cancellation\_token::none())  {  m\_registrationToken = m\_cancellationToken.register\_callback([this]()  {  // Set the m\_aborted flag so that OnError does not  // change the status code to E\_ABORT.  m\_aborted = true;  if (m\_request != nullptr)  {  m\_request->Abort();  }  m\_completionEvent.set(  make\_tuple<HRESULT, wstring>(S\_OK, wstring()));  });  }  }  // Called when the HTTP request is being redirected to a new URL.  IFACEMETHODIMP OnRedirect(IXMLHTTPRequest2\*, PCWSTR)  {  return S\_OK;  }  // Called when HTTP headers have been received and processed.  IFACEMETHODIMP OnHeadersAvailable(IXMLHTTPRequest2\*, DWORD, PCWSTR)  {  return S\_OK;  }  // Called when a portion of the entity body has been received.  IFACEMETHODIMP OnDataAvailable(IXMLHTTPRequest2\*, ISequentialStream\*)  {  return S\_OK;  }  // Called when the entire entity response has been received.  IFACEMETHODIMP OnResponseReceived(IXMLHTTPRequest2\*,  ISequentialStream\* pResponseStream)  {  // Convert the response to Unicode wstring.  HRESULT hr;  // Holds the response as a Unicode string.  wstringstream ss;  while (true)  {  // Read the response.  ULONG cb;  char buffer[4096];  hr = pResponseStream->Read(buffer, sizeof(buffer), &cb);  if (FAILED(hr) || (cb == 0))  {  break; // Error or no more data to process, exit loop.  }  else  {  // First determine the size required to store the Unicode string.  int const sizeRequired =  MultiByteToWideChar(CP\_UTF8, 0, buffer, cb, nullptr, 0);  wchar\_t\* wstr = new(std::nothrow) wchar\_t[sizeRequired + 1];  hr = wstr ? S\_OK : E\_OUTOFMEMORY;  if (SUCCEEDED(hr))  {  // Convert the string.  MultiByteToWideChar(  CP\_UTF8, 0, buffer, cb, wstr, sizeRequired);  wstr[sizeRequired] = L'\0'; // Terminate the string.  ss << wstr; // Write the string to the stream.  delete[] wstr;  }  }  }  m\_completionEvent.set(make\_tuple<HRESULT, wstring>(  SUCCEEDED(hr) ? S\_OK : hr, // dont' return S\_FALSE  SUCCEEDED(hr) ? ss.str() : wstring()));  return hr;  }  // Called when an error occurs during the HTTP request.  IFACEMETHODIMP OnError(IXMLHTTPRequest2\*, HRESULT hrError)  {  // Set the completion event if the operation was not canceled.  if (!m\_aborted)  {  m\_completionEvent.set(  make\_tuple<HRESULT, wstring>(move(hrError), wstring()));  }  return S\_OK;  }  // Retrieves the completion event for the HTTP operation.  task\_completion\_event<tuple<HRESULT, wstring>> const& GetCompletionEvent()  {  return m\_completionEvent;  }  private:  ~HttpRequestCallback()  {  // Unregister the callback.  if (m\_cancellationToken != cancellation\_token::none())  {  m\_cancellationToken.deregister\_callback(m\_registrationToken);  }  }  // When set, flags that the operation was canceled.  bool m\_aborted;  // Signals that the download operation was canceled.  cancellation\_token m\_cancellationToken;  // Used to unregister the cancellation token callback.  cancellation\_token\_registration m\_registrationToken;  // The IXMLHTTPRequest2 that processes the HTTP request.  ComPtr<IXMLHTTPRequest2> m\_request;  // Task completion event that is set when the  // download operation completes.  task\_completion\_event<tuple<HRESULT, wstring>> m\_completionEvent;  }; |

1. Implement the StartDownload method of HttpRequest by pasting the following code into HttpRequest.cpp below the implementation of HttpRequestCallback you created in the previous step.

|  |
| --- |
| **Example 27 – HttpRequest.cpp (implementation of HttpRequest)** |
| /\*  Implementation of class HttpRequest  \*/  /\*  Starts a download of the specified URI.  \*/  task<tuple<HRESULT, wstring>> HttpRequest::StartDownload(  const wstring& method, const wstring& uri,  const wstring& user, const wstring& password,  cancellation\_token cancellationToken)  {  // Create an IXMLHTTPRequest2 object.  ComPtr<IXMLHTTPRequest2> xhr;  HRESULT hr = CoCreateInstance(CLSID\_XmlHttpRequest, nullptr,  CLSCTX\_INPROC,IID\_PPV\_ARGS(&xhr));  if (FAILED(hr)) throw Platform::Exception::CreateException(hr);  // Create callback.  auto callback = Make<HttpRequestCallback>(xhr.Get(), cancellationToken);  if (!callback) throw Platform::Exception::CreateException(E\_OUTOFMEMORY);  // Open and send the request.  // You must pass both user and password.  // The value of user doesn't matter but it can't be nullptr.  hr = xhr->Open(  method.c\_str(),  uri.c\_str(),  callback.Get(),  user.c\_str(),  password.c\_str(),  nullptr, nullptr);  if (FAILED(hr)) throw Platform::Exception::CreateException(hr);  hr = xhr->Send(nullptr, 0);  if (FAILED(hr)) throw Platform::Exception::CreateException(hr);  // Return a task that completes when the HTTP operation completes.  // We pass the callback to the continuation because the lifetime of the  // callback must exceed the operation to ensure that cancellation  // works correctly.  return create\_task(callback->GetCompletionEvent()).then(  [callback](task<tuple<HRESULT, wstring>> previousTask)  {  return previousTask;  });  } |

1. Add an include directive for HttpRequest.h to MainPage.xaml.h.

|  |
| --- |
| **Example 28 – MainPage.xaml.h (include directives)** |
| //  // MainPage.xaml.h  // Declaration of the MainPage class.  //  #pragma once  // Include directives added by Visual Studio.  #include "MainPage.g.h"  // Our additional include directives.  #include "FeedData.h"  #include "WebResult.h"  #include "HttpRequest.h" |

Add include directives and a linker directive to pch.h. The following shows the complete pch.h file.

|  |
| --- |
| **Example 29 – pch.h (complete)** |
| //  // pch.h  // Header for standard system include files.  //  #pragma once  // Include directives added by Visual Studio.  #include <collection.h>  #include "App.xaml.h"  // Our additional include directives.  #include <ppltasks.h>  #include <string>  #include <sstream>  #include <wrl.h>  #include <msxml6.h>  // Our linker directive.  #pragma comment(lib, "msxml6") |

With the implementations of these classes, we’re almost done with the app. We’re ready to create our query into the Bing Search API. At this point, you might want to make sure that your app builds successfully. You can run it too, although it won’t yet do anything different.

#### Performing the Bing Search

Next, we’ll implement two methods in the MainPage class to handle fetching the search results from the Bing Search API and parsing them.

#### To implement the Bing search functionality

1. Add the following two method declarations into MainPage.xaml.h below the web results collection and Bing information we added earlier.

|  |
| --- |
| **Example 30 – MainPage.xaml.h (methods to get and parse Bing search results)** |
| private:  // Event handler signatures.  // ...  // Feed data object and supporting methods  // ...  // Web results collection and Bing information.  // ...  // Methods to get and parse Bing search results.  void SearchFor(Platform::String^ text);  void ParseResults(Platform::String^ resultStr); |

1. In MainPage.xaml.cpp, add the method body for the SearchFor method.

|  |
| --- |
| **Example 31 – MainPage.xaml.cpp (SearchFor method)** |
| /\*  Performs the Bing Search  \*/  void MainPage::SearchFor(Platform::String^ text)  {  String^ url = serviceRoot + "/Web?Query=%27" + text + "%27&$format=json";    HttpRequest::StartDownload(  L"GET",  url->Data(),  accountKey->Data(), accountKey->Data(),  concurrency::cancellation\_token::none())  // Create a continuation that shows the results on the UI.  // The UI must be updated on the STA thread.  // Therefore, schedule the continuation to run on the current context.  .then([this](task<tuple<HRESULT, wstring>> previousTask)  {  HRESULT hr = get<0>(previousTask.get());  if (FAILED(hr))  {  BingStatus->Text = "The operation failed";  // TODO: Handle the error further.  }  else  {  auto text = ref new String(get<1>(previousTask.get()).c\_str());  ParseResults(text);  BingStatus->Text = "Bing Search Results";  BingListView->Visibility = Windows::UI::Xaml::Visibility::Visible;  BrowserGrid->Visibility = Windows::UI::Xaml::Visibility::Collapsed;  }  }, task\_continuation\_context::use\_current());  } |

1. Also in MainPage.xaml.cpp, add the method body for the ParseResults method.

|  |
| --- |
| **Example 32 – MainPage.xaml.cpp (ParseResults method)** |
| /\*  Helper method to parse the JSON results from Bing.  \*/  void MainPage::ParseResults(Platform::String^ resultStr)  {  auto json = Windows::Data::Json::JsonObject::Parse(resultStr);  auto results = json->GetNamedObject("d")->GetNamedArray("results");  for (unsigned k = 0; k < results->Size; k++)  {  auto r = ref new WebResult();  auto obj = results->GetObjectAt(k);  r->ID = obj->GetNamedString("ID");  r->Title = obj->GetNamedString("Title");  r->Description = obj->GetNamedString("Description");  r->DisplayUrl = obj->GetNamedString("DisplayUrl");  r->Title = obj->GetNamedString("Title");  r->Url = ref new Windows::Foundation::Uri(obj->GetNamedString("Url"));  this->webResults->Items->Append(r);  }  } |

1. Finally, modify the FeedListView\_SelectionChanged event handler we implemented earlier as a stub method to get the currently selected feed item from the list and use it to begin the Bing search.

|  |
| --- |
| **Example 33 – MainPage.xaml.cpp (FeedListView\_SelectionChanged)** |
| /\*  Invoked when the selection changes on the RSS feed list.  \*/  void MainPage::FeedListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e)  {  (void) e; // Unused parameter.  ListView^ view = safe\_cast<ListView^>(sender);  FeedItem^ feedItem = safe\_cast<FeedItem^>(view->SelectedItem);  if (feedItem != nullptr)  {  webResults->Items->Clear();  BingStatus->Text = "Searching Bing";  SearchFor(feedItem->KBID);  }  } |

Now when you run the app and select a feed item, you should see the Bing search results on the right side of the display.



Figure 6 – Running the app

#### Displaying the Webpage of a Search Result

When you select one of the search results, we want to navigate to the webpage that the search result points to. We have everything in place, we just need to wire up the last event handler.

Modify the BingListView\_SelectionChanged event handler we implemented earlier as a stub method to get the currently selected search result item from the list and use it to navigate our WebView element to the selected webpage.

|  |
| --- |
| **Example 34 – MainPage.xaml.cpp (BingListView\_SelectionChanged)** |
| /\*  Invoked when the selection changes on the Bing list.  \*/  void MainPage::BingListView\_SelectionChanged(  Platform::Object^ sender,  Windows::UI::Xaml::Controls::SelectionChangedEventArgs^ e)  {  (void) sender, e; // Unused parameters.  WebResult^ webResult = safe\_cast<WebResult^>(BingListView->SelectedItem);  if (webResult != nullptr)  {  BingListView->Visibility = Windows::UI::Xaml::Visibility::Collapsed;  BrowserGrid->Visibility = Windows::UI::Xaml::Visibility::Visible;  WebContentView->Navigate(webResult->Url);  }  } |

Now when you run the app, you can view the webpage of a search result by selecting it. To hide the WebView element, tap the back button or select another item from the RSS feed list. Your app is completed.



Figure 7 - Running the completed app

## Creating an Objective-C Console Application

This tutorial shows you how to create an Objective-C console application to retrieve search results in XML and display them. This tutorial uses the native NSURLConnection class to create a connection to the Bing Search service.

#### Overview

As a high-level overview, here are the pieces you’ll implement:

1. A main program entry point.
2. A custom interface and implementation that creates an asynchronous request to the Bing Search API and displays the results.
3. A category for the native NSData type to add a method for Base64 encoding. You need Base64 encoding for your account key in the HTTP Authorization header.

#### Getting started

To get started, create a folder where you can place the files that belong to this application. For this tutorial, all files go in the same folder. Here’s a list of the files you’ll have with the finished application.

|  |  |
| --- | --- |
| **File** | **Description** |
| Main.m | The main program file. |
| Bing.h | Header definition for the Bing class. |
| Bing.m | Implementation of the Bing class. |
| NSData\_Base64.h | Header definition file for the additional category of NSData. |
| NSData\_Base64.m | Implementation of the Base64 encoder for NSData. |

#### Creating the Main Program Entry Point

Main.m is pretty straightforward. With it, you set up an auto-release pool, create a Bing object and send it a *search* message. Since the search is performed asynchronously, Main.m also keeps a run loop going while it waits for the Bing object to complete. In a standard GUI-based application, you wouldn’t have to do this but here you’ll need it or your application will exit before the asynchronous operation has a chance to finish.

|  |
| --- |
| **Example 1 – Main.m** |
| #include <Foundation/Foundation.h>  #import "Bing.h"  int main (void)  {  NSAutoreleasePool \*pool = [[NSAutoreleasePool alloc] init];  Bing \*bing = [[[Bing alloc] init] autorelease];  [bing search];  // This prevents the app from exiting before the async operation is complete.  while (bing.busy)  {  [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode  beforeDate:[NSDate distantFuture]];  }  NSLog(@"Done with main");  [pool drain];  return 0;  } |

#### Creating the Bing Class

#### Interface

The Bing class interface is sparse; it declares instance variables and exposes three members. Save the following code as Bing.h.

|  |
| --- |
| **Example 2 – Bing.h** |
| #include <Foundation/Foundation.h>  @interface Bing : NSObject  {  @private  NSString \*rootUri;  NSString \*accountKey;  NSMutableData \*responseData;  BOOL busy;  BOOL isTitle;  }  @property (readonly) BOOL busy;  - (id) init;  - (void) search;  @end |

#### Implementation

Bing.m provides the implementation for the Bing class. It uses an interface extension to provide two helper methods that are used internally to the class. The following code is a partial listing of the Bing class. The code shows the Bing interface extension and its two public methods init and search.

|  |
| --- |
| **Example 3 – Bing.m (partial)** |
| #import "Bing.h"  #import "NSData\_Base64.h"  @interface Bing ()  - (void)asyncLoad:(NSString \*)fullUri;  - (NSMutableURLRequest \*)getRequest:(NSString \*)fullUri;  @end  @implementation Bing  @synthesize busy;    /\*\*\*  \* Initializes the instance.  \*\*\*/  - (id)init  {  if (self = [super init])  {  self->busy = NO;  // Replace the following with your account key.  self->accountKey = @"YourAccountKey";  self->rootUri = @"https://api.datamarket.azure.com/Bing/Search/";  }  return self;  }  /\*\*\*  \* Main entry point to perform the search.  \*\*\*/  - (void)search  {  NSString \*query = @"'San Francisco Baseball'";  NSString \*format = @"atom";  NSString \*market = @"'en-us'";  NSInteger top = 10;  NSMutableString \*fullUri = [NSMutableString stringWithCapacity:256];  [fullUri appendString:rootUri];  [fullUri appendFormat:@"Web?$format=%@", format];  [fullUri appendFormat:@"&Query=%@",  [query stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];  [fullUri appendFormat:@"&Market=%@",  [market stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];  [fullUri appendFormat:@"&$top=%d", top];  [self asyncLoad:fullUri];  }  @end |

The init method is pretty standard, initializing the instance variables for your account key and the service root URI. The search method creates the full URI for the query and passes it to asyncLoad. The asyncLoad method performs the network operations for the search.

Save the code in Example 3 as Bing.m. You’ll be adding to this file throughout the rest of this section of the tutorial.

#### Starting the asynchronous request

The following code shows the asyncLoad method, which you use to create the NSURLConnection object and begin the asynchronous request to the Bing Search API. Helper method getRequest creates and returns an NSMutableURLRequest object that has been initialized with the full request URI and the HTTP Authorization header. Note that getRequest uses a non-native method to Base64-encode your account key. This method is implemented as part of a category for the native NSData type and is discussed later in this tutorial.

|  |
| --- |
| **Example 4 – Bing.m (partial – implementation section)** |
| /\*\*\*  \* Performs an asynchronous load.  \*\*\*/  - (void)asyncLoad:(NSString \*)fullUri  {  self->busy = YES;  NSLog(@"Begin asynchronous request");  NSURLConnection \*connection = [[NSURLConnection alloc] initWithRequest:  [self getRequest:fullUri] delegate:self];  if (connection)  {  NSLog(@"Connection established");  self->responseData = [[NSMutableData data] retain];  }  else  {  NSLog(@"Connection failed");  self->busy = NO;  }  }  /\*\*\*  Gets the request object, fully initialized with headers.  \*\*\*/  -(NSMutableURLRequest \*)getRequest:(NSString \*)fullUri  {  NSMutableURLRequest \*request = [[NSMutableURLRequest alloc] init];  [request setURL:[NSURL URLWithString:fullUri]];  [request setHTTPMethod:@"GET"];  NSString \*authStr = [NSString stringWithFormat:@"%@:%@", accountKey, accountKey];  NSData \*authData = [authStr dataUsingEncoding:NSUTF8StringEncoding];  NSString \*authValue = [NSString stringWithFormat:@"Basic %@",  [authData base64Encoding]];  [request setValue:authValue forHTTPHeaderField:@"Authorization"];  return request;  } |

Add the code in Example 4 to the implementation section of bing.m.

#### Asynchronous request delegates

The initializer for the NSURLConnection object specifies keyword self as the object that contains the asynchronous delegates. Therefore, you can implement the asynchronous delegates as part of the Bing class implementation. There are four delegates that you can implement to handle the various phases of the asynchronous request.

|  |
| --- |
| **Example 5 – Bing.m (partial – implementation section)** |
| /\*\*\*  \* Delegate to handle receiving a response.  \* This delegate could be called more than once.  \*\*\*/  - (void)connection:(NSURLConnection \*)connection didReceiveResponse:  (NSHTTPURLResponse \*)response  {  NSLog(@"Response Code: %d", [response statusCode]);  [self->responseData setLength:0];  }  /\*\*\*  \* Delegate to handle receiving a chunk of data.  \*\*\*/  - (void)connection:(NSURLConnection \*)connection didReceiveData:(NSData \*)data  {  NSLog(@"Did receive data - received %d bytes", [data length]);  [self->responseData appendData:data];  }  /\*\*\*  \* Delegate to handle successful completeion of the asyncronous request.  \* Creates an XML parser, parses the response, releases resources,  \* and sets the busy flag to NO.  \*\*\*/  - (void)connectionDidFinishLoading:(NSURLConnection \*)connection  {  NSLog(@"Finished loading: Received %d bytes of data",[self->responseData length]);  NSXMLParser \*parser = [[NSXMLParser alloc] initWithData: self->responseData];  [parser setDelegate: self];  [parser parse];  // release the connection, and the data object  [connection release];  [self->responseData release];  self->busy = NO;  }  /\*\*\*  \* Delegate to handle failure in the asyncronous request.  \* Releases resources, emits a message, and sets the busy flag to NO.  \*\*\*/  - (void)connection:(NSURLConnection \*)connection didFailWithError:(NSError \*)error  {  [connection release];  [self->responseData release];  NSLog(@"Connection failed Error - %@", [error localizedDescription]);  self->busy = NO;  } |

With the asynchronous delegates in Example 5, you can handle the following tasks:

* Response received – Because of a possible redirect, the didReceiveData delegate can be called more than once during the lifetime of the asynchronous request. When the delegate is invoked, you reset the length of the response data to zero, discarding any previous data.
* Data received – Data may arrive in several chunks. As data is received, you append it to any previous data by using the didReceiveData delegate.
* Finished loading – The connectionDidFinishLoading delegate is invoked when the data transfer is complete. Here you create an NSXMLParser object to parse the data.
* Error – Handles errors in the asynchronous request.

Add the code in Example 5 to the implementation section of Bing.m.

#### XML parser delegates

NSXMLParser implements event-driven XML parsing. For this tutorial, you’ll implement two delegates to respond to XML parsing events. Add these to the implementation section of bing.m.

|  |
| --- |
| **Example 6 – Bing.m (partial – implementation section)** |
| /\*\*\*  \* Delegate to handle starting an XML element parse.  \* If the element is "d:Title", set the flag.  \*\*\*/  - (void)parser:(NSXMLParser \*)parser didStartElement:(NSString \*)elementName  namespaceURI:(NSString \*)namespaceURI  qualifiedName:(NSString \*)qName  attributes:(NSDictionary \*)attributeDict  {  self->isTitle = ([elementName isEqualToString:@"d:Title"]);  }  /\*\*\*  \* Delegate to handle finding characters during an XML element parse.  \* If the flag is set, output the title.  \*\*\*/  - (void)parser:(NSXMLParser \*)parser foundCharacters:(NSString \*)string  {  if (self->isTitle) NSLog(@"Title: %@", string);  self->isTitle = NO;  } |

#### Adding a Category to the NSData Class

The final step in creating your application is to add a category to the NSData class that provides a method for Base64 encoding. You’ll need Base64 encoding for your account key in the HTTP Authorization header.

#### Interface

Create the interface for the new category. The interface exposes one method. Save the following code as NSData\_Base64.h.

|  |
| --- |
| **Example 7 – NSData\_Base64.h** |
| #include <Foundation/Foundation.h>  /\*\*\*  \* A category to add Base64 encoding to the NSData type.  \*\*\*/  @interface NSData (Base64)  - (NSString \*)base64Encoding;  @end |

#### Implementation

The following code shows the implementation of the base64Encoding method. Save this as NSData\_Base64.m.

|  |
| --- |
| **Example 8 – NSData\_Base64.m** |
| #import "NSData\_Base64.h"  static const char encodingTable[] =  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";  @implementation NSData (Base64)  /\*\*\*  \* Returns an NSString that is Base64 encoded from the data in the current instance.  \*\*\*/  - (NSString \*)base64Encoding  {  if ([self length] == 0) return @"";  char \*characters = malloc((([self length] + 2) / 3) \* 4);  if (characters == NULL) return nil;  NSUInteger length = 0;  NSUInteger i = 0;  while (i < [self length])  {  char buffer[3] = {0,0,0};  short bufferLength = 0;  while (bufferLength < 3 && i < [self length])  buffer[bufferLength++] = ((char \*)[self bytes])[i++];  characters[length++] = encodingTable[(buffer[0] & 0xFC) >> 2];  characters[length++] = encodingTable[((buffer[0] & 0x03) << 4) | ((buffer[1] & 0xF0) >> 4)];  if (bufferLength > 1)  characters[length++] = encodingTable[((buffer[1] & 0x0F) << 2) | ((buffer[2] & 0xC0) >> 6)];  else characters[length++] = '=';  if (bufferLength > 2)  characters[length++] = encodingTable[buffer[2] & 0x3F];  else characters[length++] = '=';  }  return [[[NSString alloc] initWithBytesNoCopy:characters length:length  encoding:NSASCIIStringEncoding freeWhenDone:YES] autorelease];  }  @end |

#### Compiling and Running Your Application

To compile your application, use the gcc compiler. The following is the gcc command to compile the application:

gcc Main.m Bing.m NSData\_Base64.m -I /GNUstep/GNUstep/System/Library/Headers

-L /GNUstep/GNUstep/System/Library/Libraries -lobjc -lgnustep-base

-fconstant-string-class=NSConstantString -o bing

Because of text wrapping, the gcc command above spans multiple lines. However, when you invoke the command, it will be a single, long command on one line. You might need to modify the paths that point to the header and library directories.

The compiler produces a Bing executable. When you run your application, you should see output similar to the following:

Begin asynchronous request

Connection established

Response Code: 200

Did receive data - received 2920 bytes

Did receive data - received 6597 bytes

Finished loading: Received 9517 bytes of data

Title: The Official Site of the San Francisco Giants

Title: San Francisco nabs 2013 World Baseball Classic final; Phoenix and ...

Title: University of San Francisco Baseball - USF Dons

Title: San Francisco Giants | MLB Baseball Schedule, Stats, News ...

Title: San Francisco Giants - Giants Baseball Clubhouse - ESPN

Title: San Francisco Giants and other baseball teams get their Facebook ...

Title: San Francisco Giants - Wikipedia, the free encyclopedia

Title: San Francisco Giants vs. Colorado Rockies - Recap ...

Title: San Jose Officials Confident On Ballpark Plans ...

Done with main

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.