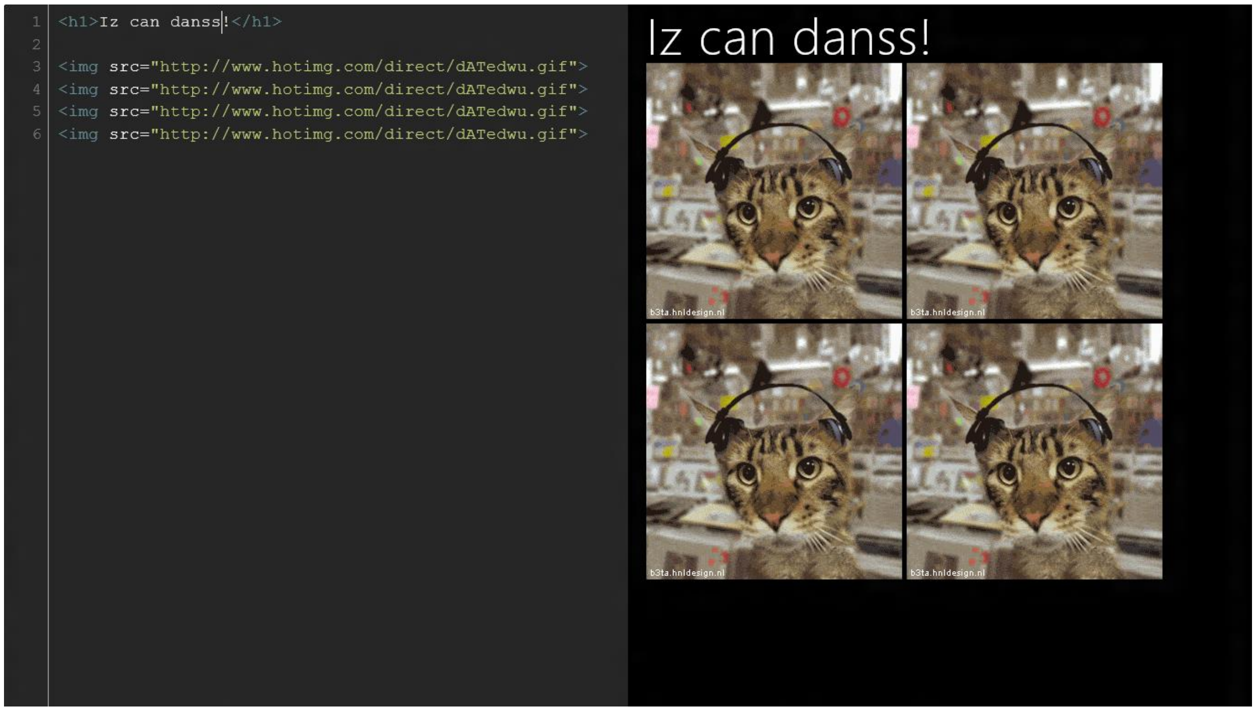
Using Windows Runtime and SDK to Build Metro style Apps (DEV350)

We will create a simple HTML editor in this document. Along the way, I will introduce you to a number of principles that underlie WinRT:

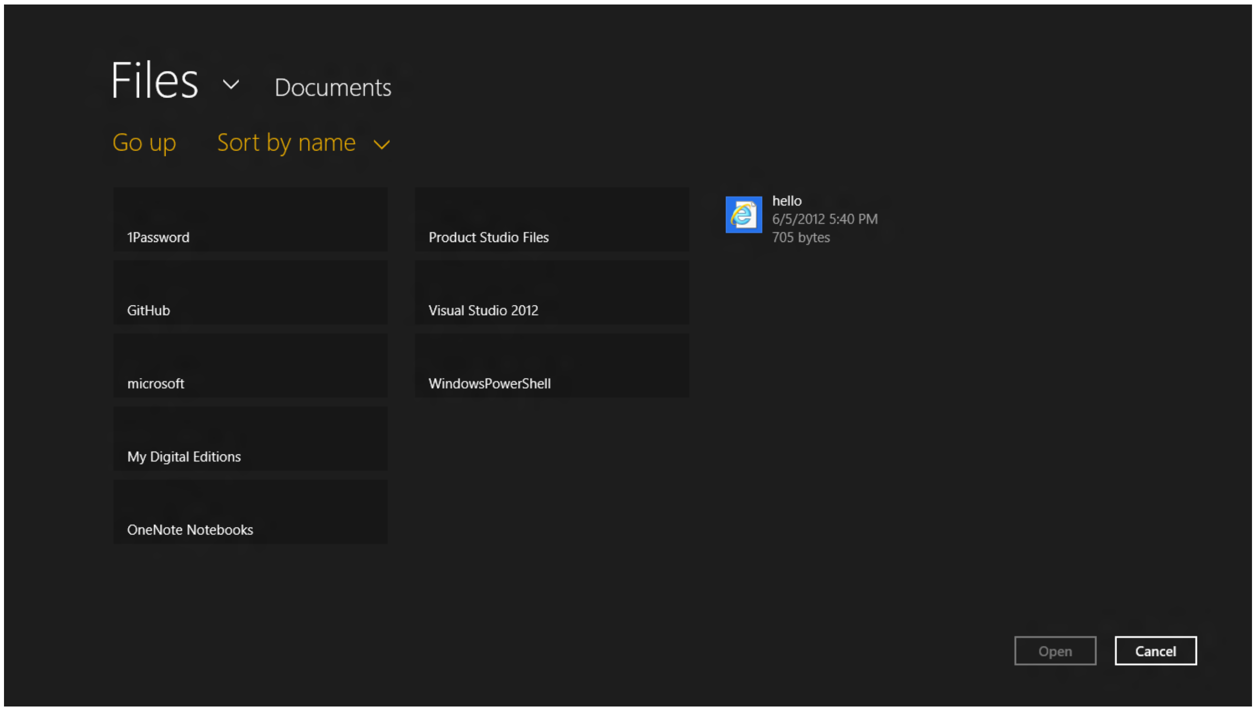
- Easily Create Apps
- Rich Apps that use all of Windows
- High Quality Apps

This document describes in detail all of the steps that I did in my Tech Ed talk. You can use this document to follow along after the talk.



```
<h1>Iz can danss!</h1>

<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
```

Before we begin, let's take a tour of the app that we will build in this talk. When we run it, you will see that it is a split screen HTML editor. The HTML that you will edit is on the left hand side. Notice how the HTML is syntax colored. This happens using a 3rd party library called CodeMirror. On the right hand side we render the HTML whenever the user presses CTRL-ENTER. Taken together, this app **uses what you know** (HTML/CSS/JS) and **uses what you have**, CodeMirror, and packages it inside of a Metro style app.



Our app is integrated with the Metro style user experience and the user's local files. Let's open a file from the user's file system. Notice how we can bring up a Metro style AppBar by swiping up from the bottom of the screen, or by pressing Windows-Z for folks who don't yet have a touch screen. When we click on the Open button, you'll see how we have what we call a File Open Picker dialog that lets the user browse for files in their file system. Both of these are examples of how we can **light up your app on the OS**. Users want apps to behave consistently across the OS, and the AppBar is one example of this. Users also want to use files that they already have on their computer, and the File Picker is a great way to enable this in your app.

Notice how this dialog is full screen; remember that Metro style is all about a touch-first experience, so the touch targets are intentionally large to make it easier for your customers to press them using their fingers. Once you select the file, notice how we can load it up directly in the HTML editor. If we hit CTRL-ENTER you can see that the file now renders.

Not only can our app open regular HTML files, it can also open .zip files via the .NET ZipArchive API. For simplicity, I've omitted a UI for picking files out of a .zip archive, instead choosing to open the first file that we find in the .zip archive and assuming that it is a HTML file. This introduces another key principle: **use the right language/runtime for the job**. Since we're building an HTML editor, it makes sense that we use JS/HTML/CSS as our rendering runtime. However, since there is a tested and supported API for manipulating .zip files in the .NET libraries, it makes sense to use C# to grab the data.

Our app also uses the Search contract in Windows 8 to search for HTML pages on Wikipedia and provide as-you-type-it search suggestions. As it turns out, it is incredibly easy to take code from the Search Contract Sample the Windows Developer Center for Metro style apps and copy-and-paste a key piece of code into your app to get it to work. This is the principle of **use what others know** in action.

All apps have bugs. It's just a matter of whether you discover them or your users discover them. If we use the File Picker in this app and click on the cancel button, the app will crash (in this case by design). In this talk we'll see how the platform can **help you when bad things happen** by seeing how Visual Studio can be used to do both live and post-mortem debugging of your app.

Windows 8 enables a broad and diverse ecosystem of devices. Some of those devices will run on CPUs with the ARM architecture. You should test your apps on those devices as well to ensure that you have a great user experience on machines that are not as powerful as your development machine. We'll see how you can use platform features to **help you test your app** and to diagnose problems if your app crashes on those devices.

Finally the Store lets you distribute your app to hundreds of millions of users. The Windows Metro style SDK helps you rapidly onboard your app to the store by letting you run the same set of automated tests that the Store runs on your app when you submit it. This **helps you ship your app** faster since running tests ahead of time will let you catch and fix problems that could delay getting your app out to your customers.

Let's start by building our app using what we know.

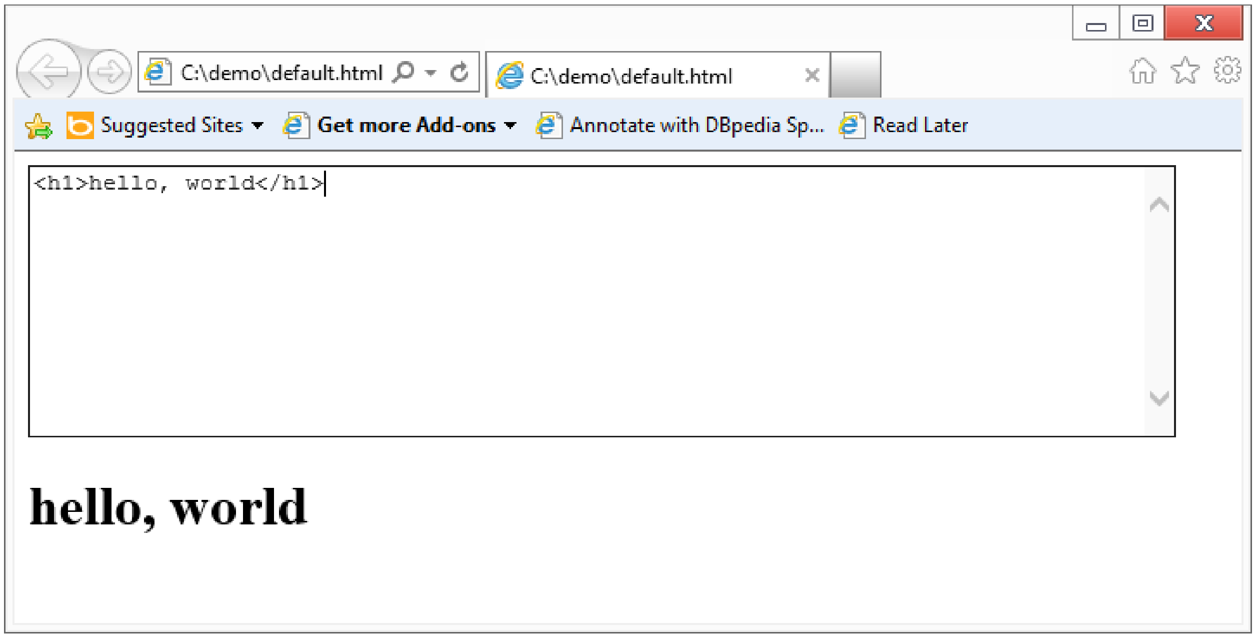
An HTML page as an "app"

Let's begin by creating an HTML page that implements a simple HTML editor. We'll see how you can use what you know already (HTML/JS) and what you have already (CodeMirror, a library for a syntax coloring text editor) to rapidly assemble the core of an application.

1. Create file called default.html that contains:

```
<html>
  <body>
    <textarea id="editor" cols="80" rows="10">&lt;h1&gt;hello,
world&lt;/h1&gt;</textarea>
    <div id="output"></div>
    <script>
        window.onload = function () {
            editor.onkeyup = function (args) {
                if (args.ctrlKey && args.key == "Enter") {
                    output.innerHTML = editor.innerText;
                }
            };
        };
    </script>
  </body>
</html>
```

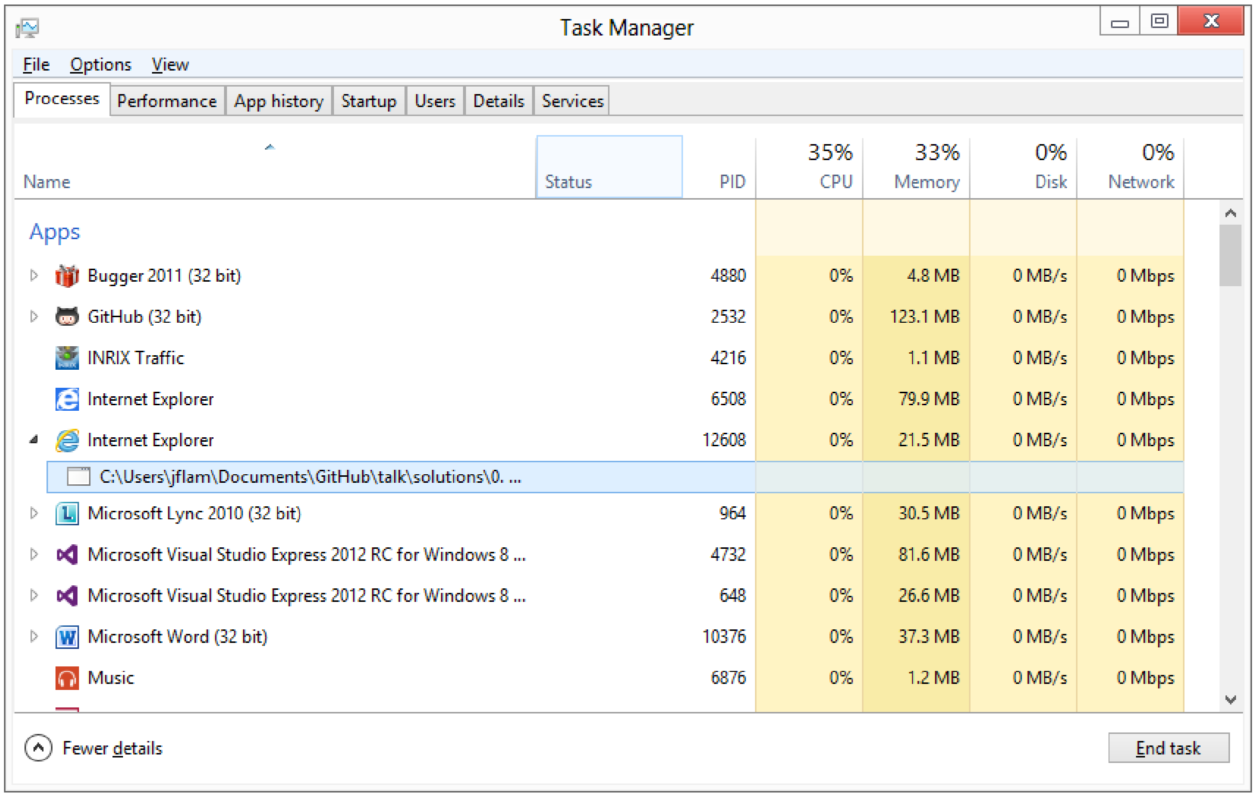
2. Note that we have an event handler bound to the CTRL-ENTER key that will take the text that the user entered in the <textarea> element and inject it into the **output** <div> element as HTML.
3. Open the file in Internet Explorer via "start editor.html". This is what you should see:



```
<h1>hello, world</h1>
```

hello, world

4. Note that it starts up in **iexplore.exe**. You can see it via task manager:


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Fewer details
End task

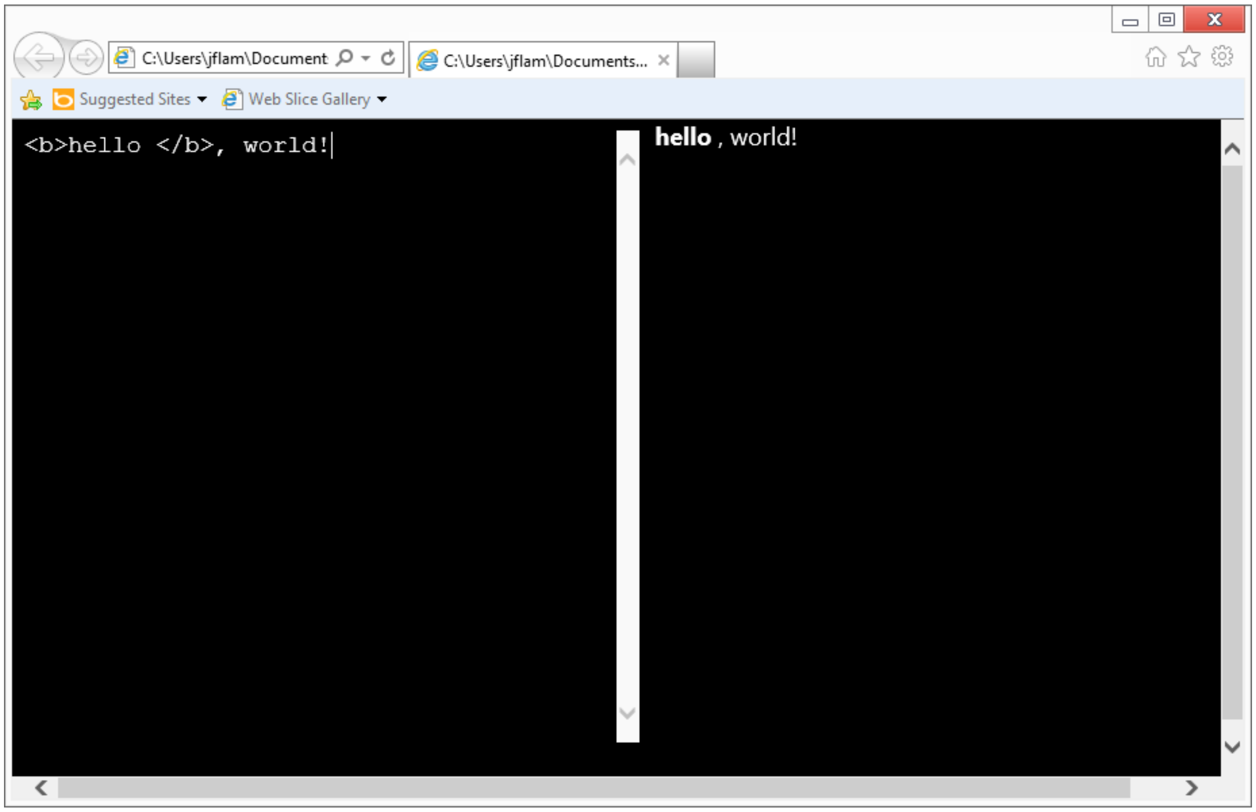
Name	Status	PID	35% CPU	33% Memory	0% Disk	0% Network
Apps						
Bugger 2011 (32 bit)		4880	0%	4.8 MB	0 MB/s	0 Mbps
GitHub (32 bit)		2532	0%	123.1 MB	0 MB/s	0 Mbps
INRIX Traffic		4216	0%	1.1 MB	0 MB/s	0 Mbps
Internet Explorer		6508	0%	79.9 MB	0 MB/s	0 Mbps
Internet Explorer		12608	0%	21.5 MB	0 MB/s	0 Mbps
C:\Users\jflam\Documents\GitHub\talk\solutions\0. ...						
Microsoft Lync 2010 (32 bit)		964	0%	30.5 MB	0 MB/s	0 Mbps
Microsoft Visual Studio Express 2012 RC for Windows 8 ...		4732	0%	81.6 MB	0 MB/s	0 Mbps
Microsoft Visual Studio Express 2012 RC for Windows 8 ...		648	0%	26.6 MB	0 MB/s	0 Mbps
Microsoft Word (32 bit)		10376	0%	37.3 MB	0 MB/s	0 Mbps
Music		6876	0%	1.2 MB	0 MB/s	0 Mbps

5. A small bit of CSS goes a long way; insert this <head> element into your code:

```
<head>
  <style>
```

```
      body {
        font-family: "Segoe UI";
        font-size: 18px;
        background-color: black;
        color: white;
      }

      #editor {
        height: 90vh;
        width: 50vw;
        border: 0px;
        position: absolute;
        background-color: black;
        color: white;
        font-size: 18px;
      }

      #output {
        left: 50vw;
        width: 50vw;
        top: 0px;
        position: absolute;
        margin-left: 20px;
      }
    </style>
  </head>
```

6. This is what you'll see:



```
<b>hello </b>, world!
```

hello , world!

7. Now let's go beyond <textarea>; let's search the web to find a library that will do syntax coloring for a browser-based editor. If you search for "browser editor javascript" you'll find CodeMirror.
8. The library contains a number of files. Let's integrate a few key files into your app. If you're following along at home, copy these files into a directory called **js**:
    - codemirror.js
    - css.js
    - htmlmixed.js
    - javascript.js
    - xml.js
9. Copy these files into a directory called **css**:
    - codemirror.css
    - lesser-dark.css
10. Edit your default.html file and add some references to the JS files:

```
<!-- CodeMirror JS References -->
<script src="js/codemirror.js"></script>
<script src="js/htmlmixed.js"></script>
<script src="js/css.js"></script>
<script src="js/xml.js"></script>
<script src="js/javascript.js"></script>
```

11. Add some references to the CSS files:

```
<!-- CSS References -->
<link rel="stylesheet" href="css/codemirror.css" />
<link rel="stylesheet" href="css/lesser-dark.css" />
```

12. Add some <style>:

```
<style>
  body {
      color: white;
      background-color: black;
      font-family: "Segoe UI";
      font-size: 18px;
  }

  .CodeMirror {
      font-size: 18px;
      width: 50`%;
  }

  .CodeMirror-scroll {
      height: 100vh;
  }

  #output {
      position: absolute;
      top: 0px;
      width: 50vw;
      left: 50vw;
      height: 100vh;
      margin-left: 20px;
  }
</style>
```

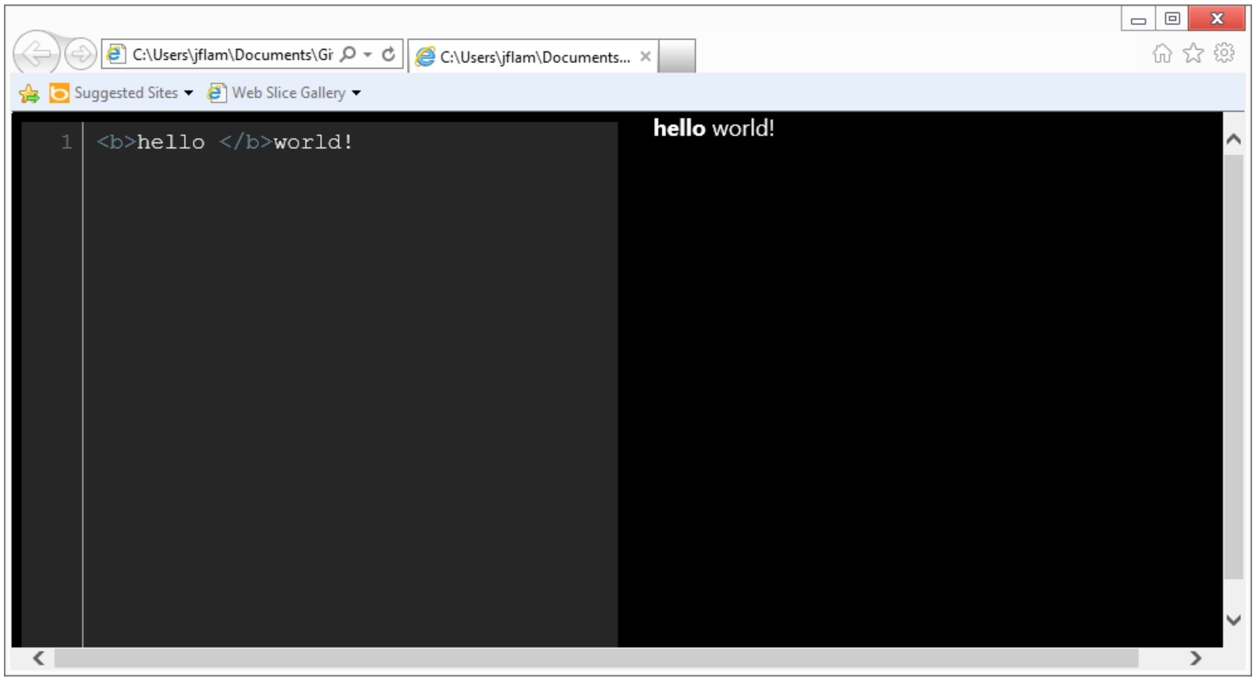
13. Change your <textarea> to a <div>:

```
<div id="editor">&lt;h1&gt;hello, world&lt;/h1&gt;</div>
```

14. Replace your existing <script>:

```
<script>
  var editor = CodeMirror.fromTextArea(document.getElementById("editor"), {
    lineNumbers: true,
    indentUnit: 2,
    theme: "lesser-dark",
    keyMap: "html_editor"
  });

  CodeMirror.keyMap.html_editor = {
    'Ctrl-Enter': function (cm) {
      var html = editor.getValue();
      output.innerHTML = html;
    },
    fallthrough: ["default"]
  };
</script>
```

15. When you run this new version of the app, you'll see that you have syntax coloring and a full-featured text editor:



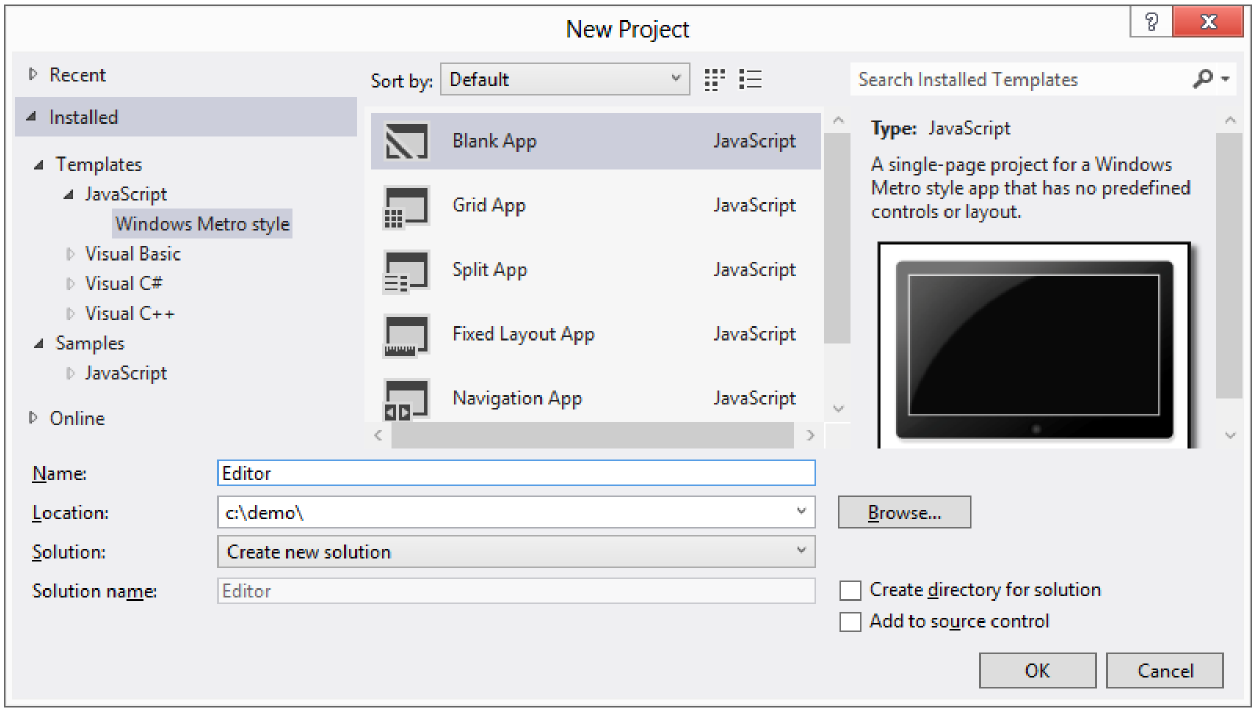
```
<b>hello </b>world!
```

hello world!

Turning our HTML page into a Metro style app

Now that you've seen how to create a simple web page "as an app", let's turn it into a full-fledged Metro style app that you can deploy to the app store and get paid (or get recognition, or drive recognition of your brand … it really is up to you!).

1. Startup Visual Studio and select File … New … Project. Name your project **Editor**.



New Project

Sort by: Default

Search Installed Templates

Recent
Installed
Templates
JavaScript
Windows Metro style
Visual Basic
Visual C#
Visual C++
Samples
JavaScript
Online

Blank App	JavaScript
Grid App	JavaScript
Split App	JavaScript
Fixed Layout App	JavaScript
Navigation App	JavaScript

Type: JavaScript

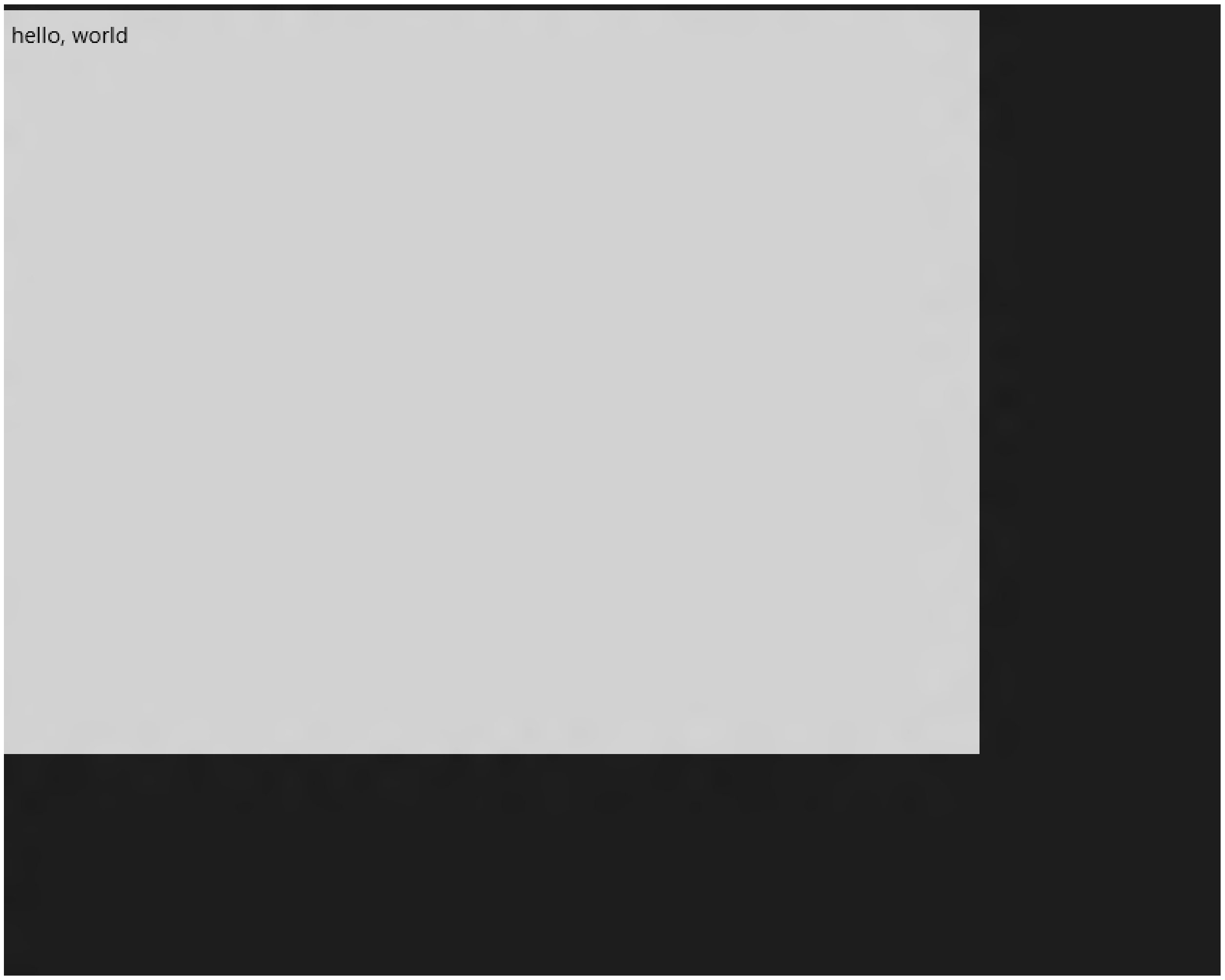
A single-page project for a Windows Metro style app that has no predefined controls or layout.

Name:	Editor
Location:	c:\demo\
Solution:	Create new solution
Solution name:	Editor

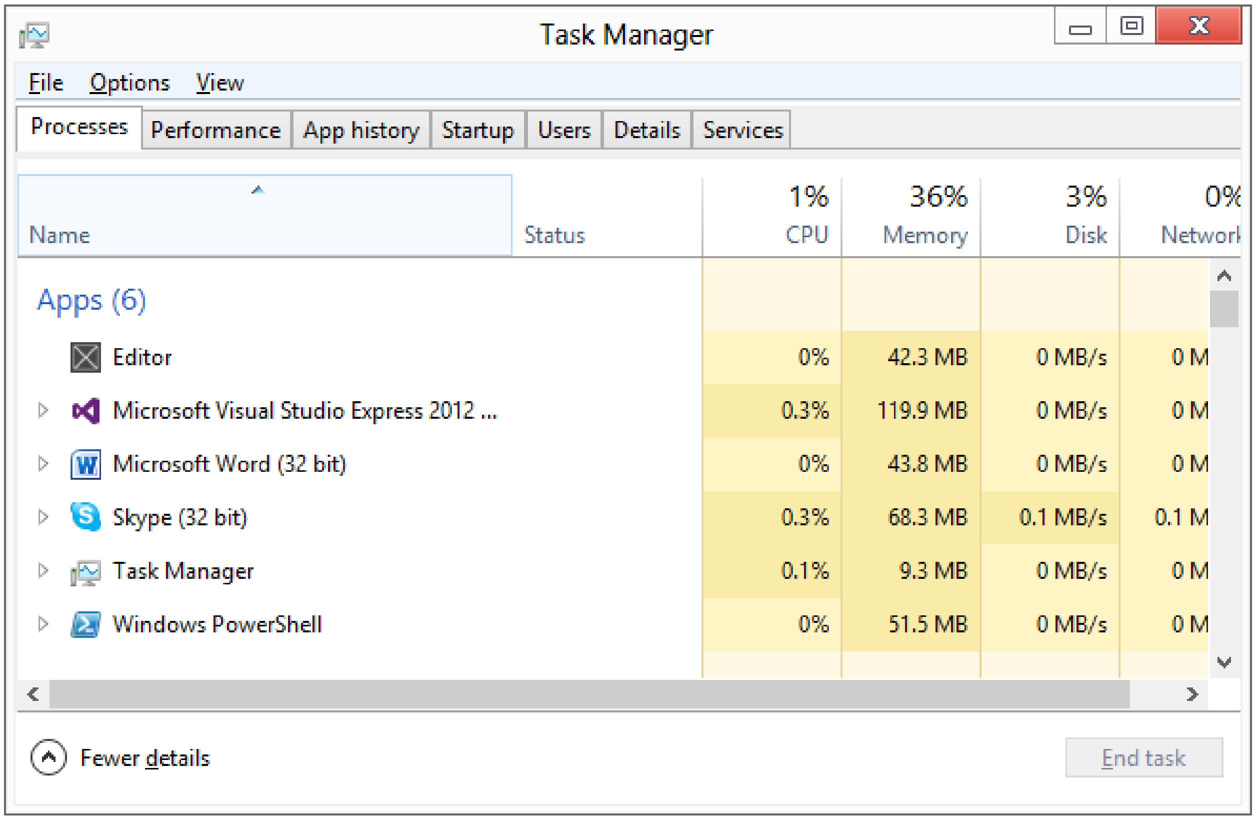
2. Navigate to **default.html** in the solution explorer, and replace the <p>Content goes here</p> HTML with:

```
<textarea cols="80" rows="25">hello, world</textarea>
```

3. If you run the app, you should see a full screen Metro style app that does much like what you saw before:

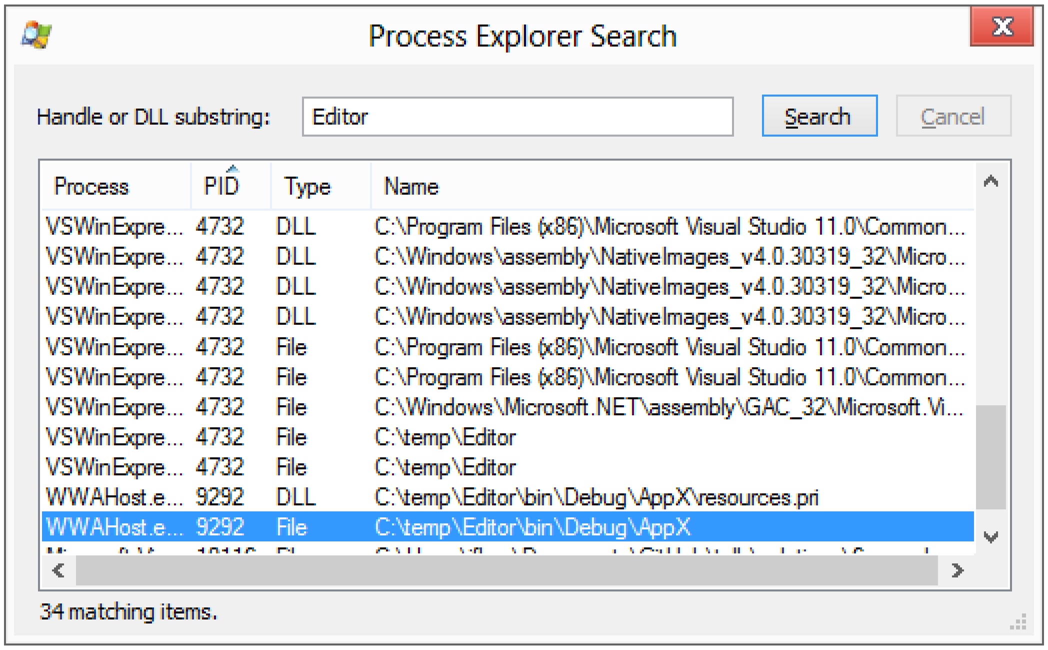

hello, world

4. However, if we look under task manager, you'll see a different view of the world than you saw before. You'll see that Editor is now a top-level entry under Apps vs. being a category under Internet Explorer.


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Fewer details
End task

Name	Status	1% CPU	36% Memory	3% Disk	0% Network
Apps (6)					
Editor		0%	42.3 MB	0 MB/s	0 M
Microsoft Visual Studio Express 2012 ...		0.3%	119.9 MB	0 MB/s	0 M
Microsoft Word (32 bit)		0%	43.8 MB	0 MB/s	0 M
Skype (32 bit)		0.3%	68.3 MB	0.1 MB/s	0.1 M
Task Manager		0.1%	9.3 MB	0 MB/s	0 M
Windows PowerShell		0%	51.5 MB	0 MB/s	0 M

5. Let's get some more information by running procexp (Process Explorer). Start procexp from the command line and hit CTRL-F to search for our app. Search for a string called "Editor". On my machine, it found a process called WWAHost that references my Editor directory. On my machine, its PID is 9292:

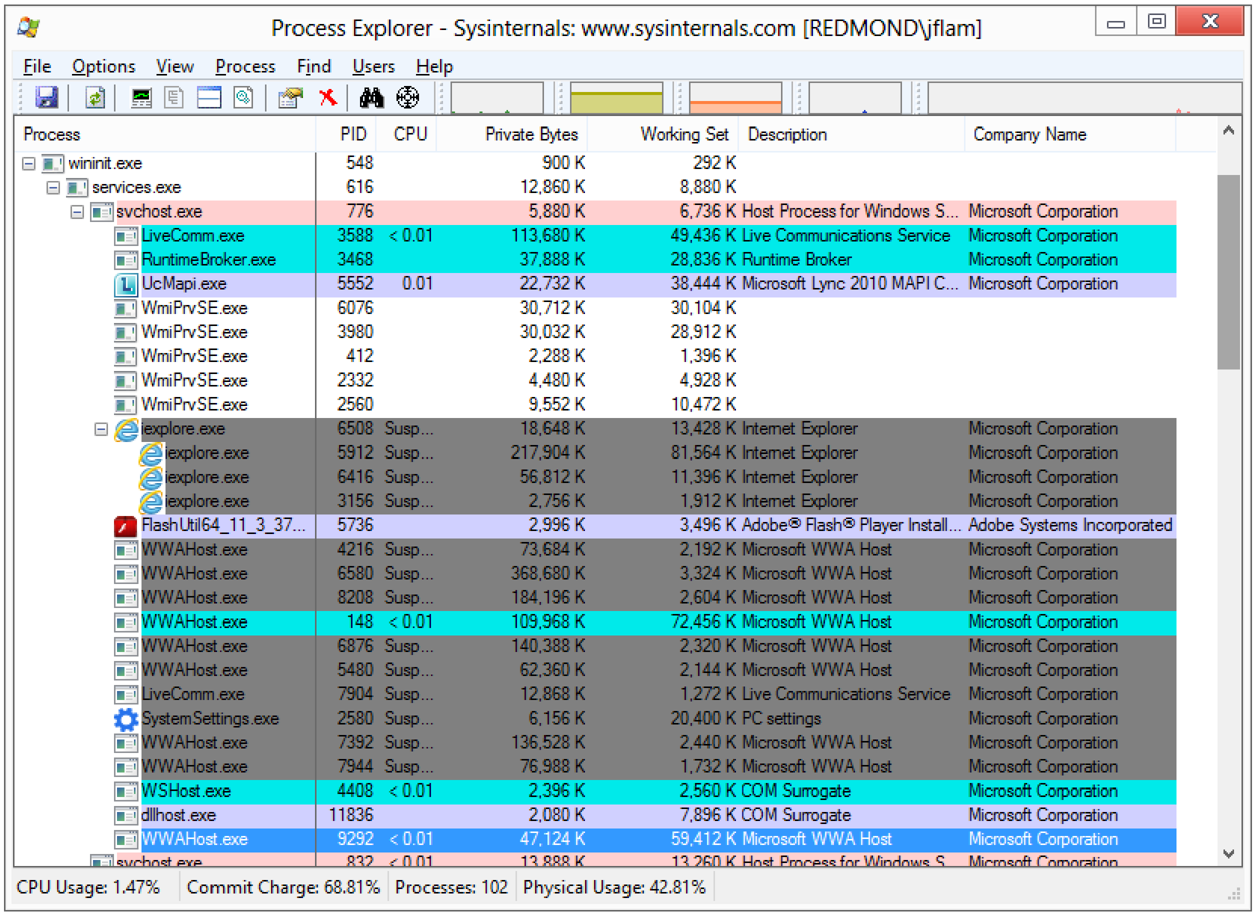


Handle or DLL substring: Editor

Process	PID	Type	Name
VSWinExpre...	4732	DLL	C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common...
VSWinExpre...	4732	DLL	C:\Windows\assembly\NativeImages_v4.0.30319_32\Micro...
VSWinExpre...	4732	DLL	C:\Windows\assembly\NativeImages_v4.0.30319_32\Micro...
VSWinExpre...	4732	DLL	C:\Windows\assembly\NativeImages_v4.0.30319_32\Micro...
VSWinExpre...	4732	File	C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common...
VSWinExpre...	4732	File	C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common...
VSWinExpre...	4732	File	C:\Windows\Microsoft.NET\assembly\GAC_32\Microsoft.Vi...
VSWinExpre...	4732	File	C:\temp\Editor
VSWinExpre...	4732	File	C:\temp\Editor
WWAHost.e...	9292	DLL	C:\temp\Editor\bin\Debug\AppX\resources.pri
WWAHost.e...	9292	File	C:\temp\Editor\bin\Debug\AppX

34 matching items.

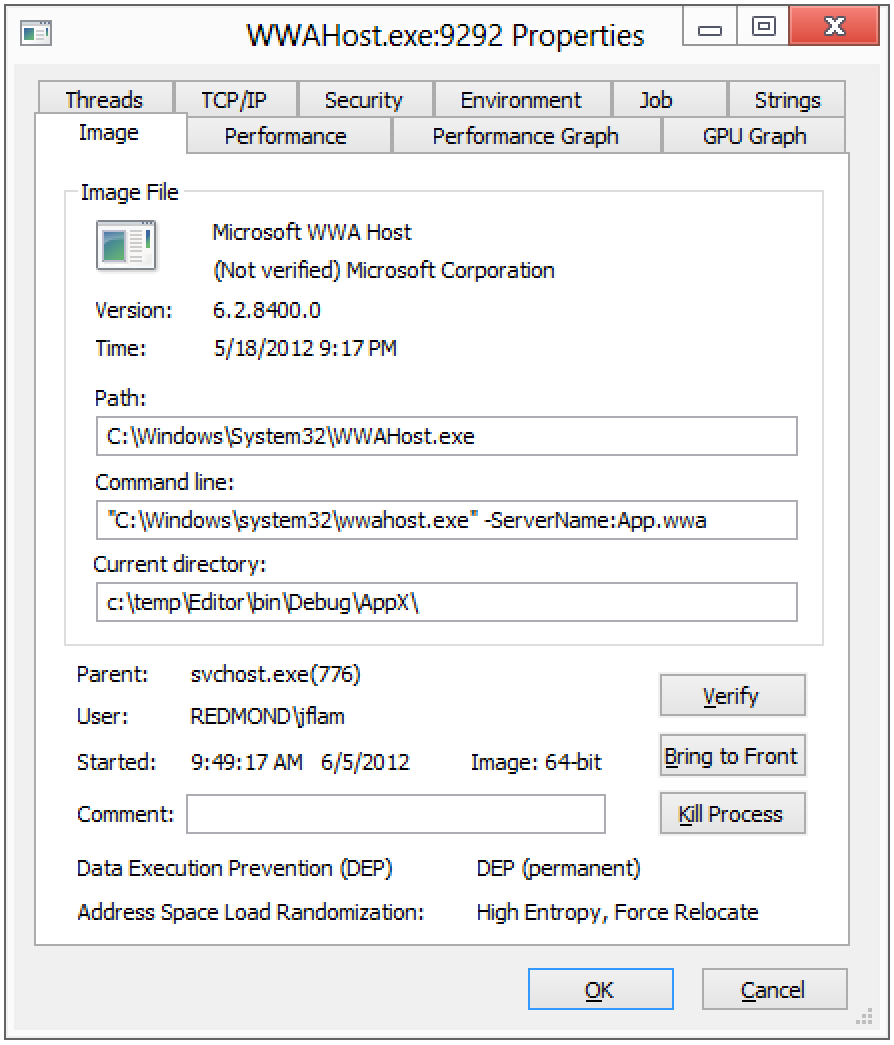
In the rest of procexp we can see additional interesting details, including the relationship of its parent process tree:


Process Explorer - Sysinternals: www.sysinternals.com [REDMOND\jflam]
File Options View Process Find Users Help

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name
wininit.exe	548		900 K	292 K		
services.exe	616		12,860 K	8,880 K		
svchost.exe	776		5,880 K	6,736 K	Host Process for Windows S...	Microsoft Corporation
LiveComm.exe	3588	< 0.01	113,680 K	49,436 K	Live Communications Service	Microsoft Corporation
RuntimeBroker.exe	3468		37,888 K	28,836 K	Runtime Broker	Microsoft Corporation
UcMapi.exe	5552	0.01	22,732 K	38,444 K	Microsoft Lync 2010 MAPI C...	Microsoft Corporation
WmiPrvSE.exe	6076		30,712 K	30,104 K		
WmiPrvSE.exe	3980		30,032 K	28,912 K		
WmiPrvSE.exe	412		2,288 K	1,396 K		
WmiPrvSE.exe	2332		4,480 K	4,928 K		
WmiPrvSE.exe	2560		9,552 K	10,472 K		
iexplore.exe	6508	Susp...	18,648 K	13,428 K	Internet Explorer	Microsoft Corporation
iexplore.exe	5912	Susp...	217,904 K	81,564 K	Internet Explorer	Microsoft Corporation
iexplore.exe	6416	Susp...	56,812 K	11,396 K	Internet Explorer	Microsoft Corporation
iexplore.exe	3156	Susp...	2,756 K	1,912 K	Internet Explorer	Microsoft Corporation
FlashUtil64_11_3_37...	5736		2,996 K	3,496 K	Adobe® Flash® Player Install...	Adobe Systems Incorporated
WWAHost.exe	4216	Susp...	73,684 K	2,192 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	6580	Susp...	368,680 K	3,324 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	8208	Susp...	184,196 K	2,604 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	148	< 0.01	109,968 K	72,456 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	6876	Susp...	140,388 K	2,320 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	5480	Susp...	62,360 K	2,144 K	Microsoft WWA Host	Microsoft Corporation
LiveComm.exe	7904	Susp...	12,868 K	1,272 K	Live Communications Service	Microsoft Corporation
SystemSettings.exe	2580	Susp...	6,156 K	20,400 K	PC settings	Microsoft Corporation
WWAHost.exe	7392	Susp...	136,528 K	2,440 K	Microsoft WWA Host	Microsoft Corporation
WWAHost.exe	7944	Susp...	76,988 K	1,732 K	Microsoft WWA Host	Microsoft Corporation
WSHost.exe	4408	< 0.01	2,396 K	2,560 K	COM Surrogate	Microsoft Corporation
dllhost.exe	11836		2,080 K	7,896 K	COM Surrogate	Microsoft Corporation
WWAHost.exe	9292	< 0.01	47,124 K	59,412 K	Microsoft WWA Host	Microsoft Corporation
svchost.exe	832	< 0.01	13,888 K	13,260 K	Host Process for Windows S	Microsoft Corporation

CPU Usage: 1.47% Commit Charge: 68.81% Processes: 102 Physical Usage: 42.81%

If we right click on WWAHost.exe and click Properties … we'll see additional information about the process, including the command line used to start it and its current directory:



Microsoft WWA Host
(Not verified) Microsoft Corporation

Version: 6.2.8400.0

Time: 5/18/2012 9:17 PM

Path:

```
C:\Windows\System32\WWAHost.exe
```

Command line:

```
"C:\Windows\system32\wwahost.exe" -ServerName:App.wwa
```

Current directory:

```
c:\temp\Editor\bin\Debug\AppX\
```

Parent: svchost.exe(776)

User: REDMOND\jflam

Started: 9:49:17 AM 6/5/2012 Image: 64-bit

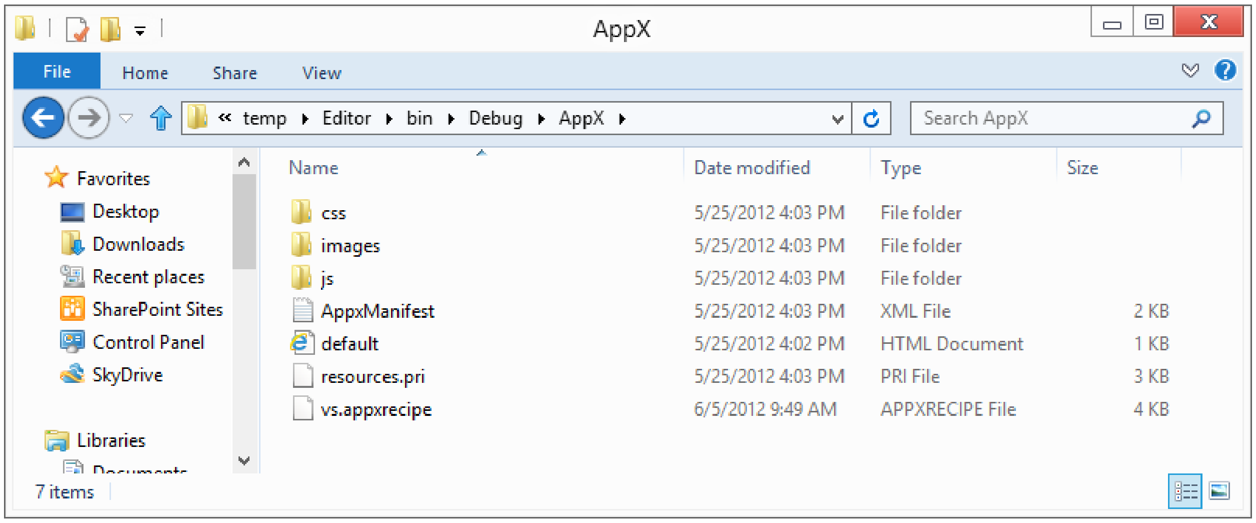
Comment:

Data Execution Prevention (DEP) DEP (permanent)

Address Space Load Randomization: High Entropy, Force Relocate

There are many more details here, but the key points are that we are launching your "application" now using a different host application: **WWAHost.exe** vs. **iexplore.exe**.

6. Another important distinction is that your app is running in a "current directory" – c:\temp\Editor\bin\Debug\AppX\ on my machine. If you look inside this directory you'll see a copy of all of the project files from your app:


AppX

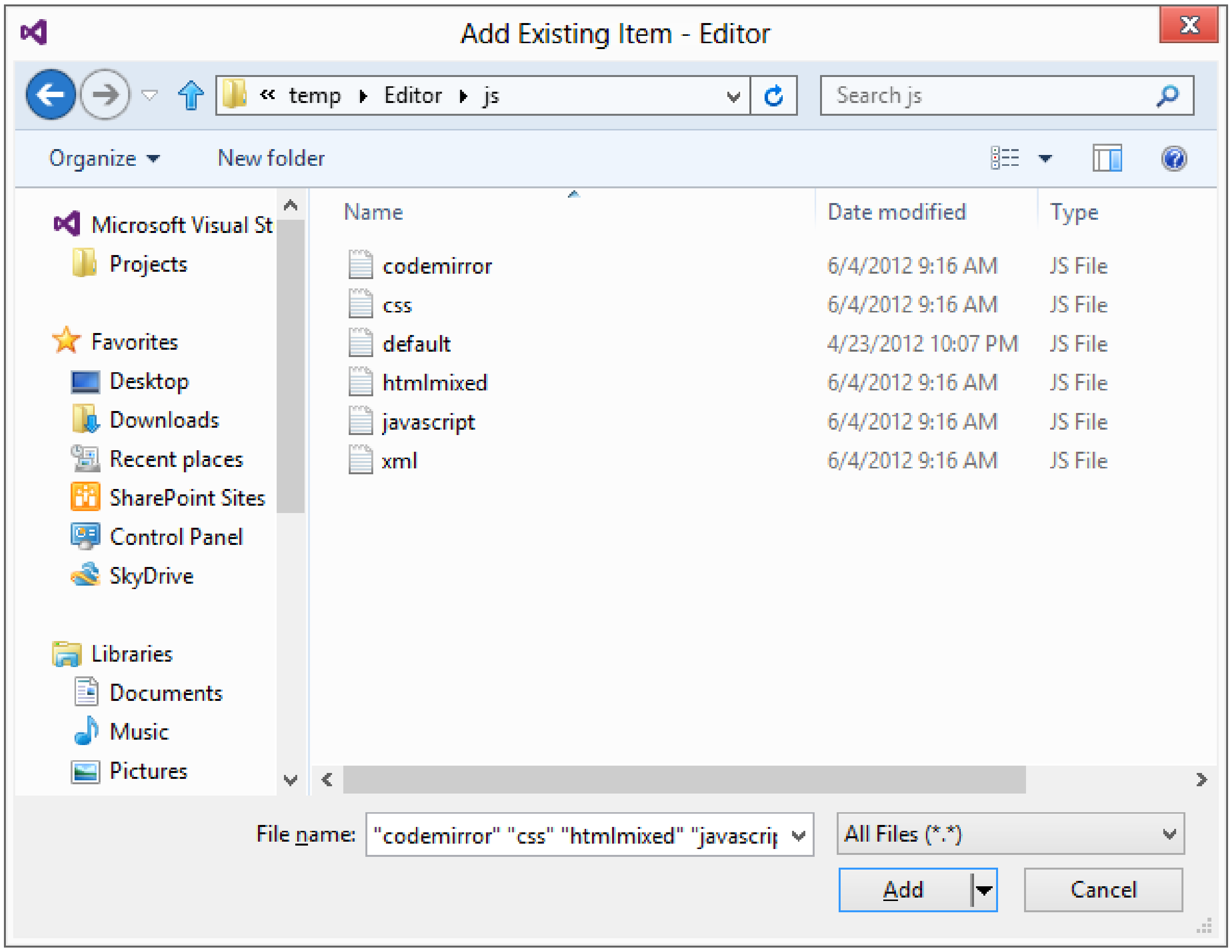
File Home Share View

« temp ▸ Editor ▸ bin ▸ Debug ▸ AppX ▸

Name	Date modified	Type	Size
css	5/25/2012 4:03 PM	File folder	
images	5/25/2012 4:03 PM	File folder	
js	5/25/2012 4:03 PM	File folder	
AppxManifest	5/25/2012 4:03 PM	XML File	2 KB
default	5/25/2012 4:02 PM	HTML Document	1 KB
resources.pri	5/25/2012 4:03 PM	PRI File	3 KB
vs.appxrecipe	6/5/2012 9:49 AM	APPXRECIPE File	4 KB

7 items

7. Because we are running under a debugger, you'll see that VS will simply point Wwahost.exe at all of the files in your project to run it. However, if you package up your application for deployment or submission to the store, you'll see that everything will wind up inside of an **.appx** file, which is really just a **.zip** file.
8. An important note here is that all of the files **must be in your project** if you want VS to copy them to this directory. Let's add files from our existing app to this directory.
9. Copy the **codemirror.js, htmlmixed.js, javascript.js, css.js, and xml.js** files from the directory you used to create the app from the previous step to the **js** directory of your new project. Also the **codemirror.css, lesser-dark.css** files from that directory to the **css** directory in your new project.
10. Add them to your project by right-clicking on the **js** directory in solution explorer and selecting Add Existing Item ... Repeat this action for the files from the **css** directory.



Name	Date modified	Type
codemirror	6/4/2012 9:16 AM	JS File
css	6/4/2012 9:16 AM	JS File
default	4/23/2012 10:07 PM	JS File
htmlmixed	6/4/2012 9:16 AM	JS File
javascript	6/4/2012 9:16 AM	JS File
xml	6/4/2012 9:16 AM	JS File

File name: "codemirror" "css" "htmlmixed" "javascrip
All Files (*.*)

11. Now add some references to those files in **default.html**:

```
 <!-- Codemirror references -->
<script src="js/codemirror.js"></script>
<script src="js/htmlmixed.js"></script>
<script src="js/css.js"></script>
<script src="js/xml.js"></script>
<script src="js/javascript.js"></script>

<link rel="stylesheet" href="css/codemirror.css" />
<link rel="stylesheet" href="css/lesser-dark.css" />
```

12. Now, change the <textarea> element into a <div> element that has the id of **editor**:

```
<div id="editor">hello, world</div>
```

13. Add some code to initialize the editor in **default.html**:

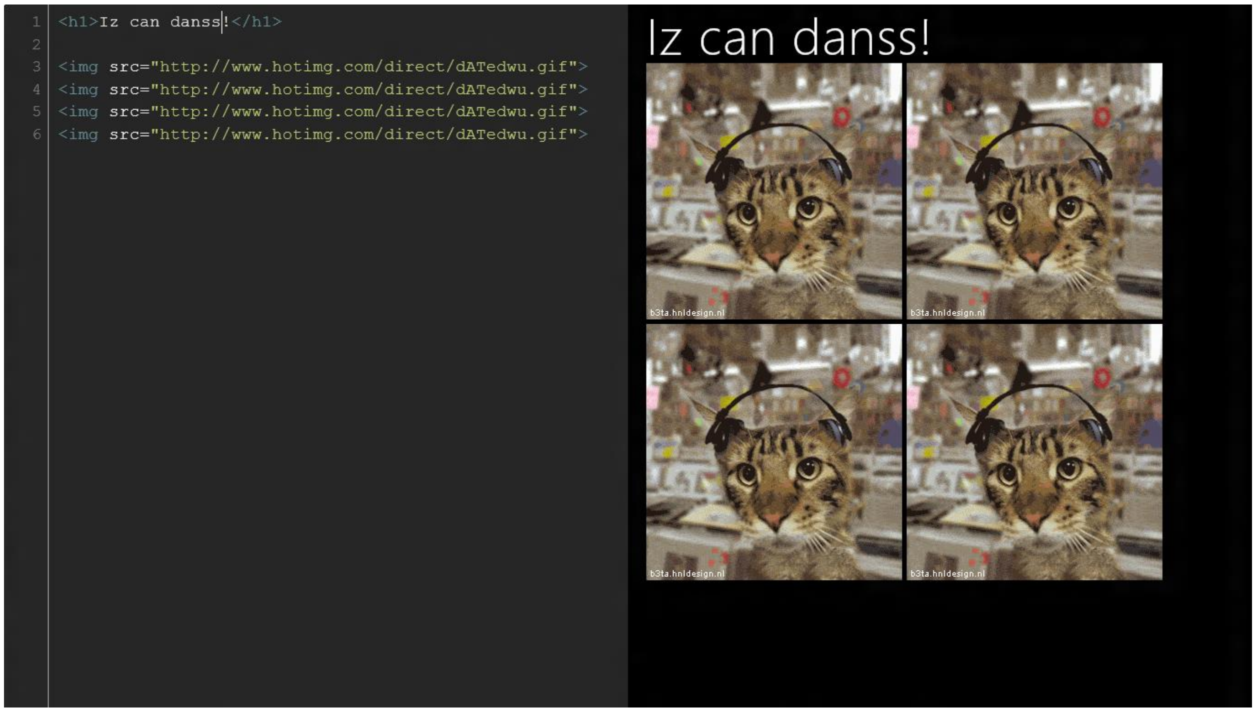
```
<script>
    WinJS.Application.editor =
CodeMirror.fromTextArea(document.getElementById("editor"), {
        lineNumbers: true,
            indentUnit: 2,
            theme: "lesser-dark",
            keyMap: "html_editor"
        });

        CodeMirror.keyMap.html_editor = {
            'Ctrl-Enter': function (cm) {
                var html = WinJS.Application.editor.getValue();
                output.innerHTML = window.toStaticHTML(html);
            },
            fallthrough: ["default"]
        };
    </script>
```

This is the first time that we've seen a reference to "WinJS" anywhere. In this case, you can see that we are creating a CodeMirror editor object and assigning it to the **editor** attribute of an object called **WinJS.Application**. Later in the event handler for the Ctrl-Enter keystroke, we retrieve this object. You should store attributes that have application-wide scope in the **WinJS.Application** object.

14. Now add some code to **default.css** to ensure that the editor occupies the left half of the app:

```
<style>
body {
    color: white;
    background-color: black;
    font-family: "Segoe UI";
    font-size: 18px;
}

.CodeMirror {
    font-size: 18px;
    width: 50%;
}

.CodeMirror-scroll {
    height: 100vh;
}

#output {
    position: absolute;
    top: 0px;
    width: 50vw;
    left: 50vw;
    height: 100vh;
    margin-left: 20px;
}
</style>
```

15. When you run your app now, you should see:



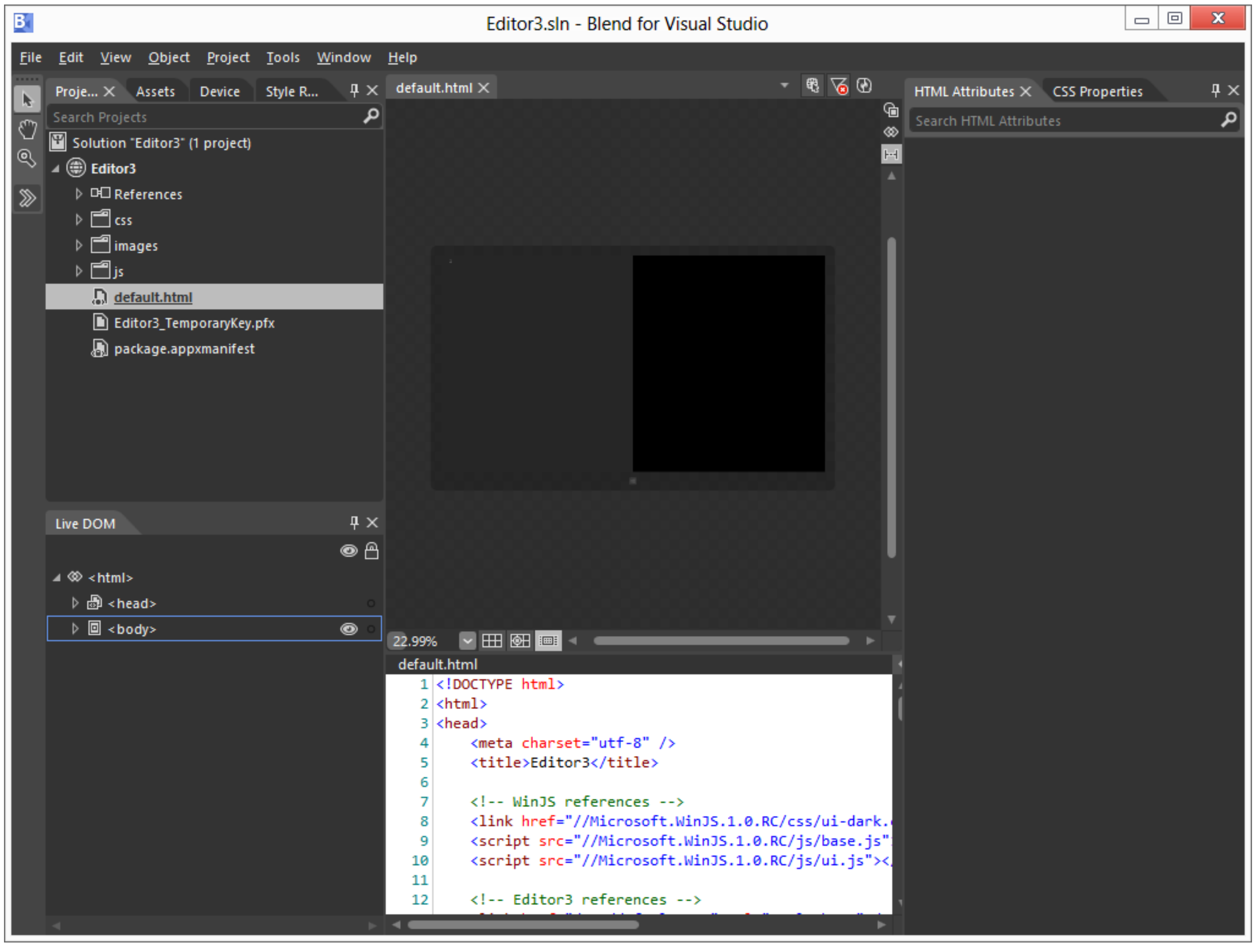
```
<h1>Iz can danss!</h1>

<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
<img src="http://www.hotimg.com/direct/dATedwu.gif">
```

Lighting up our app on Windows

Up until now, we've implemented a local HTML editor that lives in a self-contained package. Now let's do some work to integrate it into both the look and feel of the Windows 8 Metro style UI as well as adding support for interacting with files on the user's computer.

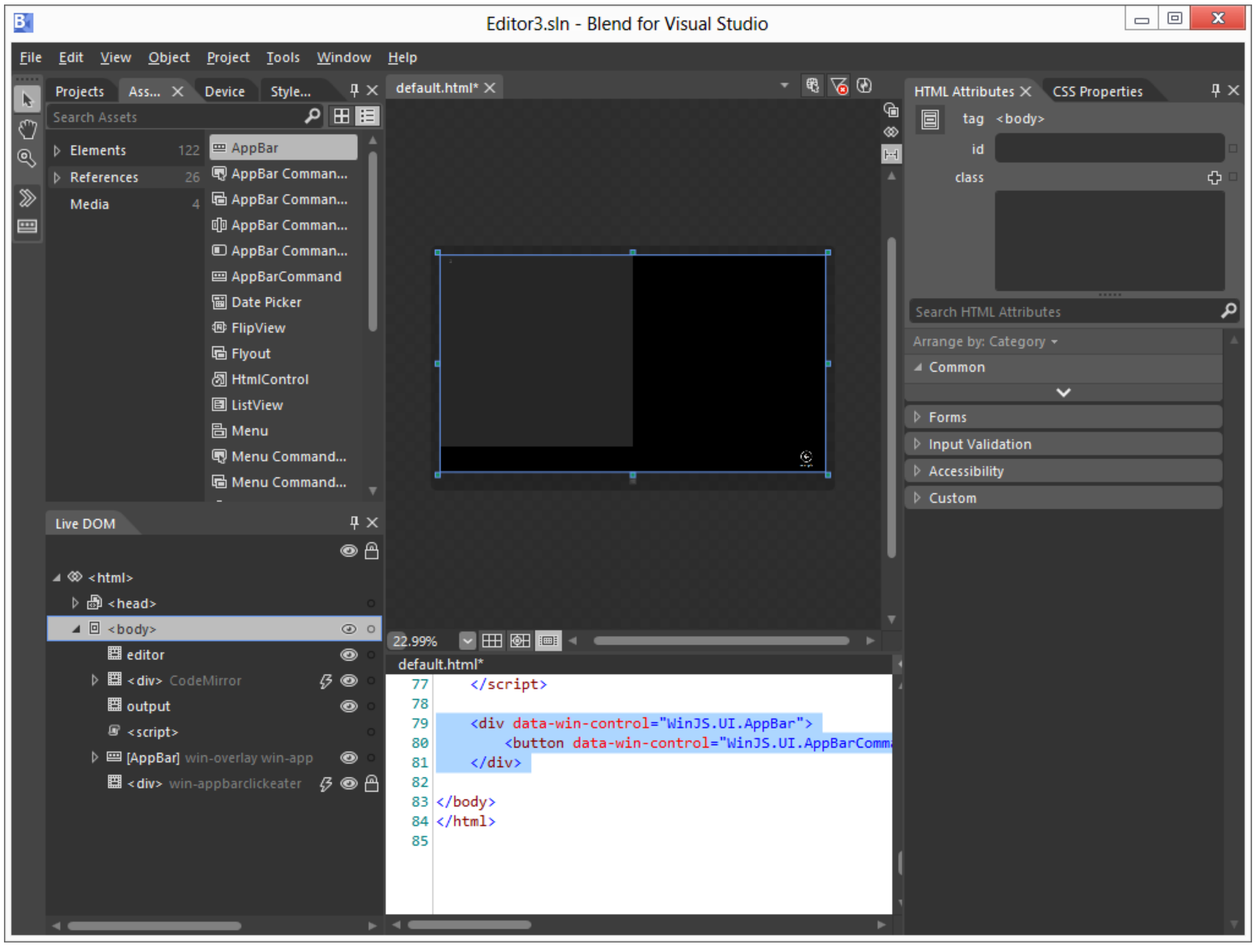
In the first part, let's add an App Bar with a couple of buttons using Blend.

1. Right click on **default.html** in VS and select Open in Blend:


Editor3.sln - Blend for Visual Studio
File Edit View Object Project Tools Window Help
Proje... Assets Device Style R...
Search Projects
Solution "Editor3" (1 project)
Editor3
References
css
images
js
default.html
Editor3_TemporaryKey.pfx
package.appxmanifest
Live DOM
<html>
<head>
<body>
default.html
22.99%
HTML Attributes CSS Properties
Search HTML Attributes

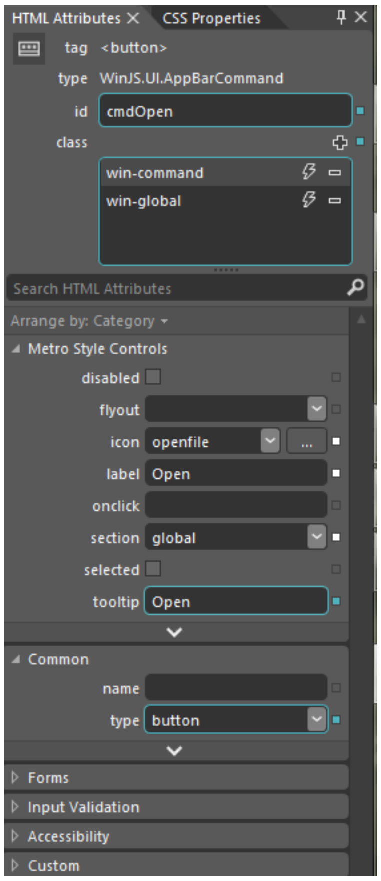
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Editor3</title>

    <!-- WinJS references -->
    <link href="//Microsoft.WinJS.1.0.RC/css/ui-dark.
    <script src="//Microsoft.WinJS.1.0.RC/js/base.js"
    <script src="//Microsoft.WinJS.1.0.RC/js/ui.js"><

    <!-- Editor3 references -->
```

2. Now let's insert an AppBar by clicking on Assets and dragging and dropping an AppBar onto the surface:



```
    </script>

    <div data-win-control="WinJS.UI.AppBar">
        <button data-win-control="WinJS.UI.AppBarComm
    </div>

</body>
</html>
```

3. Now let’s tweak the attributes of the Example button on the AppBar. Let’s zoom in on the HTML Attributes of the sample button, and change the **icon** attribute to **openfile** and the **label** attribute to Open, and the **id** attribute to **cmdOpen**:


HTML Attributes | CSS Properties

tag <button>
type WinJS.UI.AppBarCommand
id cmdOpen
class
win-command
win-global

Search HTML Attributes
Arrange by: Category

Metro Style Controls
disabled
flyout
icon openfile
label Open
onclick
section global
selected
tooltip Open

Common
name
type button

Forms
Input Validation
Accessibility
Custom

4. Now let's bind the event handler to the **click** event handler of the AppBar Open button:

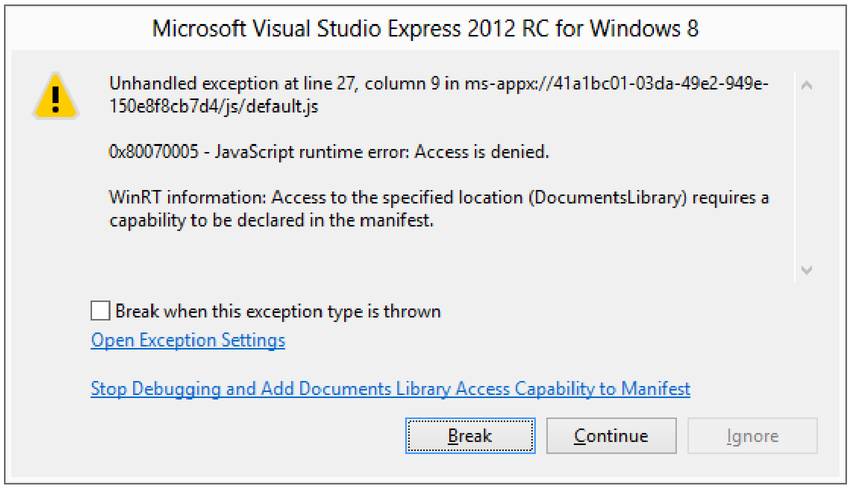
```
document.getElementById('cmdOpen').addEventListener('click', app.open_file);
```

5. Let's try to programmatically open a file on the user's computer. Add this code:

```
 app.open_file = function (args) {
     // This won't work
     var file =
Windows.Storage.KnownFolders.documentsLibrary.getFileAsync("hello.html")
     var text = Windows.Storage.FileIO.readTextAsync(file);
 };
```

Note that this code is referencing APIs in the **Windows.Storage** namespace. You should experiment with IntelliSense to examine other APIs in this namespace.

6. When we run it this is what you'll see:


Microsoft Visual Studio Express 2012 RC for Windows 8

Unhandled exception at line 27, column 9 in ms-appx://41a1bc01-03da-49e2-949e-150e8f8cb7d4/js/default.js

0x80070005 - JavaScript runtime error: Access is denied.

WinRT information: Access to the specified location (DocumentsLibrary) requires a capability to be declared in the manifest.

Break when this exception type is thrown
Open Exception Settings
Stop Debugging and Add Documents Library Access Capability to Manifest
Break
Continue
Ignore

7. Note that we are not manifesting the Documents Library Access capability which enables programmatic access to files in the user's Documents directory which map the file type associations declared in the manifest. We won't be doing this for this app, but for apps that need this kind of programmatic access (imagine a music player that enumerates all of the **.mp3** files in your music collection) they will need to do this explicitly in the manifest.
8. Instead, let's write some code to open a file picker that lets the user choose the file to open.

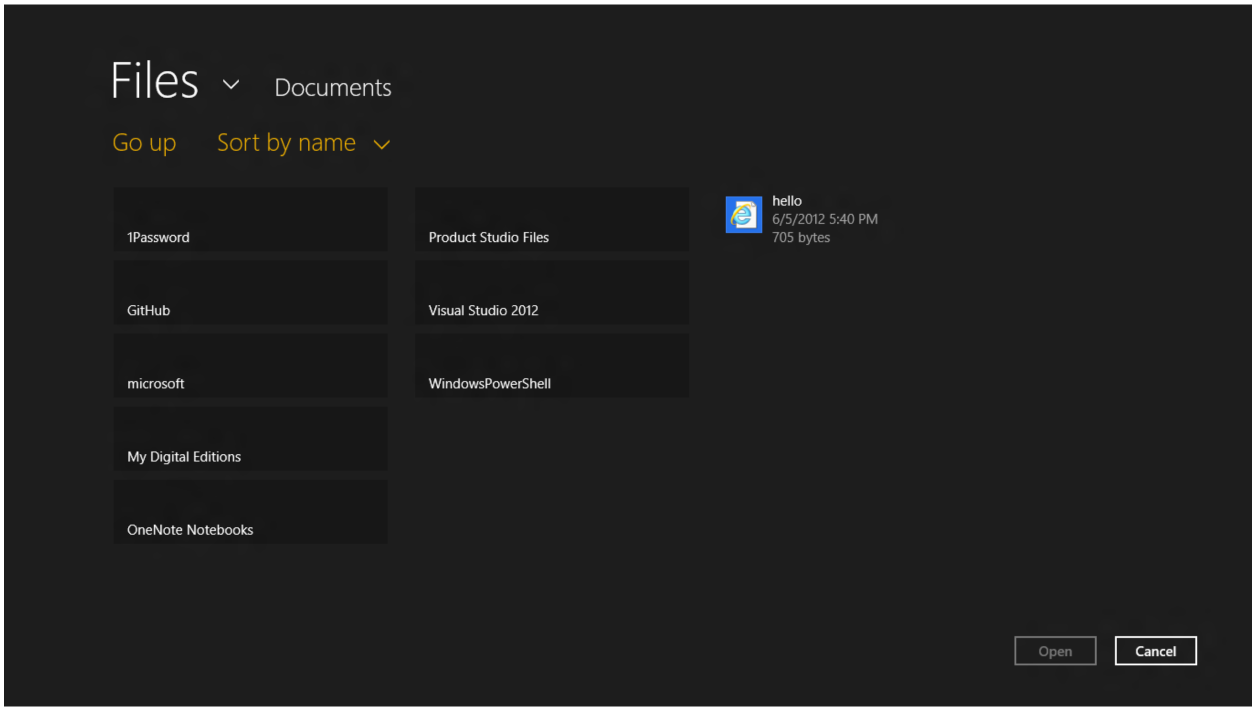
```
app.open_file = function (args) {
    var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
    openPicker.viewMode = Windows.Storage.Pickers.PickerViewMode.list;
    openPicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.documentsLibrary;
    openPicker.fileTypeFilter.replaceAll([".htm", ".zip"]);

    openPicker.pickSingleFileAsync().done(function (file) {
        FileIO.readTextAsync(file).done(function (html) {
            var clean_html = window.toStaticHTML(result);
            app.editor.setValue(clean_html);
        });
    });
};
```

This snippet creates a FileOpenPicker object, configures it to browse for documents that end in a ".htm" extension in the user's documents directory. What's interesting in this code snippet is how you tell the FileOpenPicker to go and pick some files. Notice that we that call the **pickSingleFileAsync** method on the object. This is our first exposure to the asynchronous programming model in WinRT. To help users create fast and fluid applications, we deliberately made virtually all of the APIs in WinRT async. This means that the call to **pickSingleFileAsync** returns immediately. But what does it return?

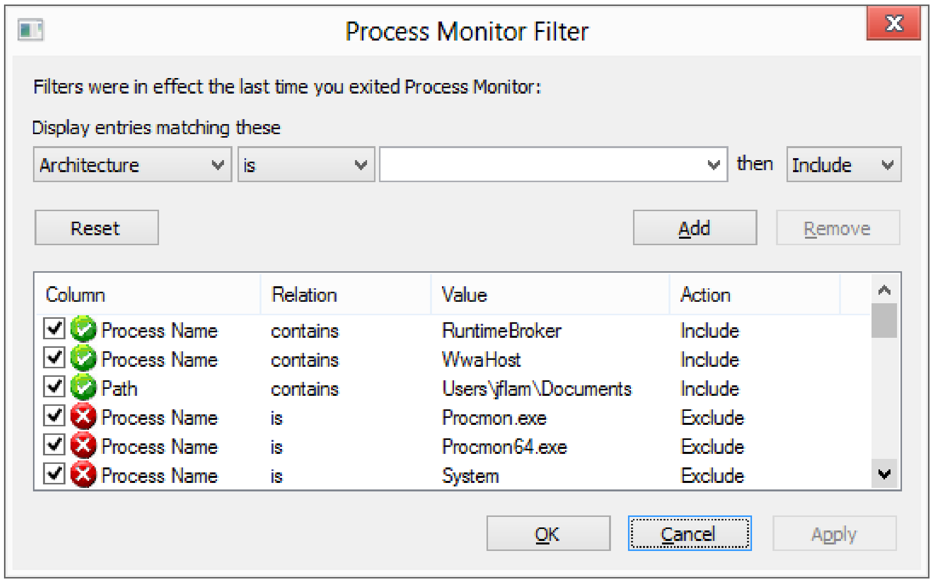
It returns a WinJS **Promise** object. The **done** method of the Promise object accepts (at a minimum) a completion function that is called once the user has picked a file. This function accepts a **file** parameter that can be used in subsequent code to manipulate the file that the user selected.

Notice that we can call another asynchronous function – the **readTextAsync** method on the **FileIO** object using the **file** object. The code here follows exactly the same pattern as the earlier call to **pickSingleFileAsync** – we pass a completion function to the Promise returned by the call to **readTextAsync**. In WinRT we have very strong naming conventions. You can expect that any method whose name ends in Async will return a WinJS Promise object.

9. When you now run the application, and click on the Open command bar button, you'll see the File picker experience:



10. Now let's see if we can understand what caused that dialog box to pop up in VS when we tried to access the file programmatically. Let's use procmon to take a look at what's manipulating the files that we are interested in. We can setup some filters in **procmon** to filter out events of interest. Note that we're looking for activity in either the RuntimeBroker.exe process or in a WwaHost.exe process where the Path contains my Documents directory.


Process Monitor Filter

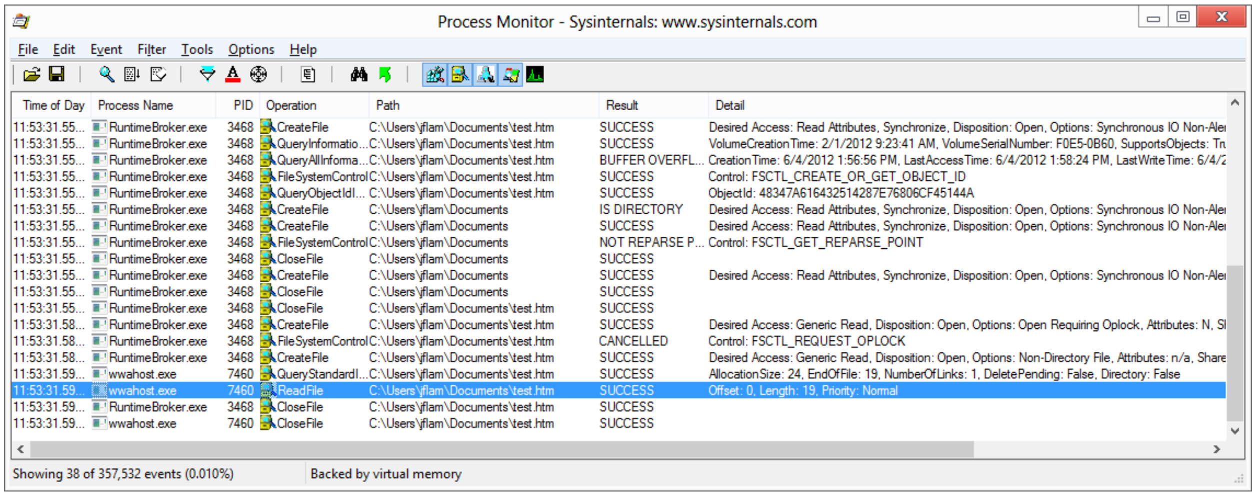
Filters were in effect the last time you exited Process Monitor:

Display entries matching these

Architecture | is | | then Include

Column	Relation	Value	Action
✓ Process Name	contains	RuntimeBroker	Include
✓ Process Name	contains	WwaHost	Include
✓ Path	contains	Users\jflam\Documents	Include
✓ Process Name	is	Procmon.exe	Exclude
✓ Process Name	is	Procmon64.exe	Exclude
✓ Process Name	is	System	Exclude

11. This is what **procmon** shows when we look for all access to files under my Documents directory.


Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Path	Result	Detail
11:53:31.55...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents\test.htm	SUCCESS	Desired Access: Read Attributes, Synchronize, Disposition: Open, Options: Synchronous IO Non-Ale
11:53:31.55...	RuntimeBroker.exe	3468	QueryInformatio...	C:\Users\jflam\Documents\test.htm	SUCCESS	VolumeCreationTime: 2/1/2012 9:23:41 AM, VolumeSerialNumber: F0E5-0B60, SupportsObjects: Tr
11:53:31.55...	RuntimeBroker.exe	3468	QueryAllInforma...	C:\Users\jflam\Documents\test.htm	BUFFER OVERFL...	CreationTime: 6/4/2012 1:56:56 PM, LastAccessTime: 6/4/2012 1:58:24 PM, LastWriteTime: 6/4/2
11:53:31.55...	RuntimeBroker.exe	3468	FileSystemControl	C:\Users\jflam\Documents\test.htm	SUCCESS	Control: FSCTL_CREATE_OR_GET_OBJECT_ID
11:53:31.55...	RuntimeBroker.exe	3468	QueryObjectIdI...	C:\Users\jflam\Documents\test.htm	SUCCESS	ObjectId: 48347A616432514287E76806CF45144A
11:53:31.55...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents	IS DIRECTORY	Desired Access: Read Attributes, Synchronize, Disposition: Open, Options: Synchronous IO Non-Ale
11:53:31.55...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents	SUCCESS	Desired Access: Read Attributes, Synchronize, Disposition: Open, Options: Synchronous IO Non-Ale
11:53:31.55...	RuntimeBroker.exe	3468	FileSystemControl	C:\Users\jflam\Documents	NOT REPARSE P...	Control: FSCTL_GET_REPARSE_POINT
11:53:31.55...	RuntimeBroker.exe	3468	CloseFile	C:\Users\jflam\Documents	SUCCESS	
11:53:31.55...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents	SUCCESS	Desired Access: Read Attributes, Synchronize, Disposition: Open, Options: Synchronous IO Non-Ale
11:53:31.55...	RuntimeBroker.exe	3468	CloseFile	C:\Users\jflam\Documents	SUCCESS	
11:53:31.55...	RuntimeBroker.exe	3468	CloseFile	C:\Users\jflam\Documents\test.htm	SUCCESS	
11:53:31.58...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents\test.htm	SUCCESS	Desired Access: Generic Read, Disposition: Open, Options: Open Requiring Oplock, Attributes: N, Sl
11:53:31.58...	RuntimeBroker.exe	3468	FileSystemControl	C:\Users\jflam\Documents\test.htm	CANCELLED	Control: FSCTL_REQUEST_OPLOCK
11:53:31.58...	RuntimeBroker.exe	3468	CreateFile	C:\Users\jflam\Documents\test.htm	SUCCESS	Desired Access: Generic Read, Disposition: Open, Options: Non-Directory File, Attributes: n/a, Share
11:53:31.59...	wwahost.exe	7460	QueryStandardI...	C:\Users\jflam\Documents\test.htm	SUCCESS	AllocationSize: 24, EndOfFile: 19, NumberOfLinks: 1, DeletePending: False, Directory: False
11:53:31.59...	wwahost.exe	7460	ReadFile	C:\Users\jflam\Documents\test.htm	SUCCESS	Offset: 0, Length: 19, Priority: Normal
11:53:31.59...	RuntimeBroker.exe	3468	CloseFile	C:\Users\jflam\Documents\test.htm	SUCCESS	
11:53:31.59...	wwahost.exe	7460	CloseFile	C:\Users\jflam\Documents\test.htm	SUCCESS	

Showing 38 of 357,532 events (0.010%) | Backed by virtual memory

12. Note that procmon shows us that there is a special process, **RuntimeBroker.exe** that first opens the file, and now we see that **WWAHost.exe** now reads the file, but without opening it. RuntimeBroker is responsible for opening the file and then marshaling the file handle back to WwaHost.exe who can now manipulate it.

13. Let's complete our scenario by adding the ability to save our files. Let's add a save button. Insert this HTML in the App Bar section of **default.html**:

```
<button data-win-control="WinJS.UI.AppBarCommand" data-win-options="{icon:'save',
id:'cmdSave', label:'Save', section:'global', type:'button'}"></button>
```

14. We will bind the save button to another handler. Put this after the other **addEventListener** call in **app.onactivated**:

```
document.getElementById('cmdSave').addEventListener('click', app.save_file);
```

15. Now let’s add the **save_file** function, making sure that we use the FileSavePicker to open the file and the futureAccessList APIs to cache it so that we can do CTRL-S correctly.

```
app.save_file = function (args) {
    var text = app.editor.getValue();
    if (app.file_token == null) {
        var savePicker = new Windows.Storage.Pickers.FileSavePicker();
        savePicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.documentsLibrary;
        savePicker.defaultFileExtension = ".htm";
        savePicker.suggestedFileName = "my_html";
        savePicker.fileTypeChoices.insert("HTML", [".htm"]);

        savePicker.pickSaveFileAsync().done(function (file) {
            if (file) {
                // Squirrel away a token for future access
                app.file_token =
Windows.Storage.AccessCache.StorageApplicationPermissions.futureAccessList.add(fil
e);
                Windows.Storage.FileIO.writeTextAsync(file, text);
            }
        });
    } else {

Windows.Storage.AccessCache.StorageApplicationPermissions.futureAccessList.getFile
Async(app.file_token).done(function (file) {
            if (file) {
                Windows.Storage.FileIO.writeTextAsync(file, text);
            }
        });
    }
};
```

16. Now let’s add code to our **open_file** API above to ensure that we squirrel away a token for future access when we open the file as well. Do this before we call **readTextAsync**:

```
       // Squirrel away a token for future access
       app.file_token =
Windows.Storage.AccessCache.StorageApplicationPermissions.futureAccessList.add(file);
```

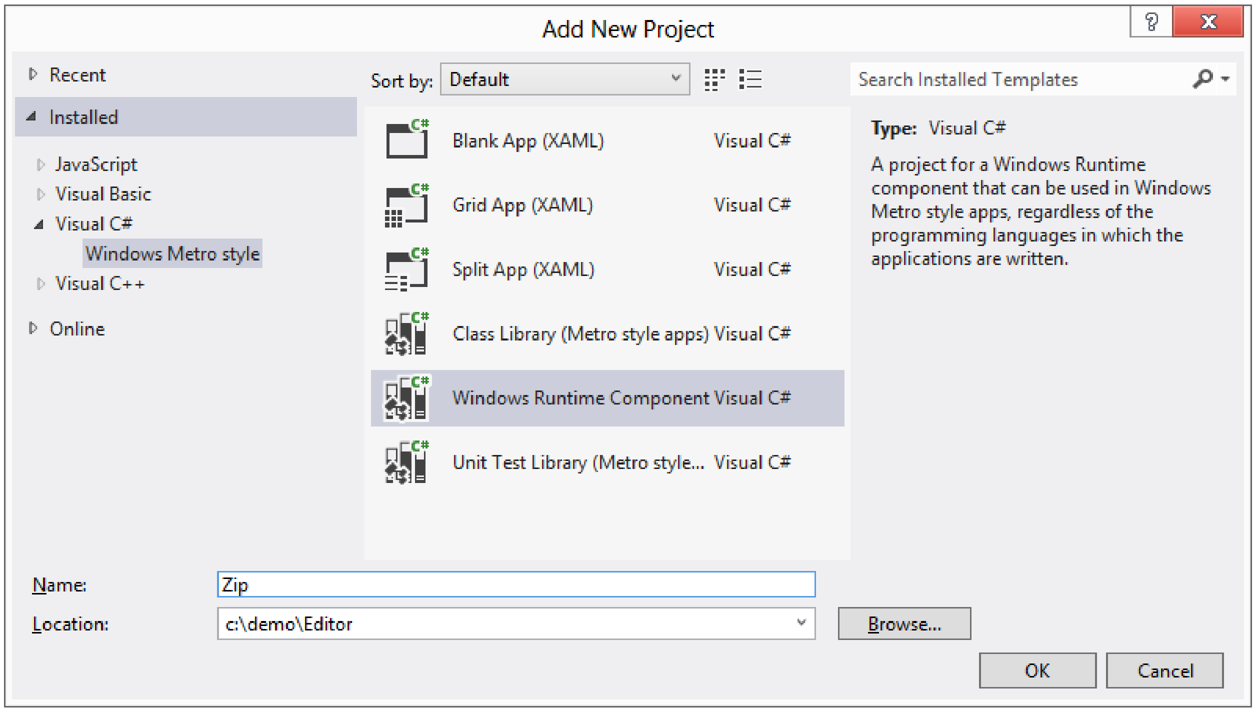
17. We now have a fully functional app that supports loading and saving to the file system. Note that we don’t need to add any additional capabilities to the app – programmatic access to the file system is not allowed.

Mixing and matching code and libraries written with different languages

Let’s add an unusual, but not unreasonable capability to our app. Let’s open a zipped file that contains some HTML that we would like to edit. For simplicity we will assume that the file contains a single HTML file and decompress and open.

If we look around, we’ll find a few JS libraries that can manipulate **.zip** files. However, if we look a bit harder we’ll see that there was a new ZipArchive class that was added in .NET 4.5. While that ZipArchive class is not a WinRT component, we'll see that it is really easy to create a .NET WinRT component that calls ZipArchive on behalf of our app. Let's do this now.

1. Add a New Project to the solution by right-clicking on the Solution node in Solution Explorer and selecting Add New Project from the popup menu. It must be a C# Windows Runtime Component called **Zip**:


Add New Project
Recent
Installed
JavaScript
Visual Basic
Visual C#
Windows Metro style
Visual C++
Online
Sort by: Default
Search Installed Templates
Blank App (XAML) Visual C#
Grid App (XAML) Visual C#
Split App (XAML) Visual C#
Class Library (Metro style apps) Visual C#
Windows Runtime Component Visual C#
Unit Test Library (Metro style... Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for a Windows Runtime component that can be used in Windows Metro style apps, regardless of the programming languages in which the applications are written.

Name: Zip

Location: c:\demo\Editor

2. Replace the existing code in **Class1.cs** with the following and rename the file to **Zip.cs**:

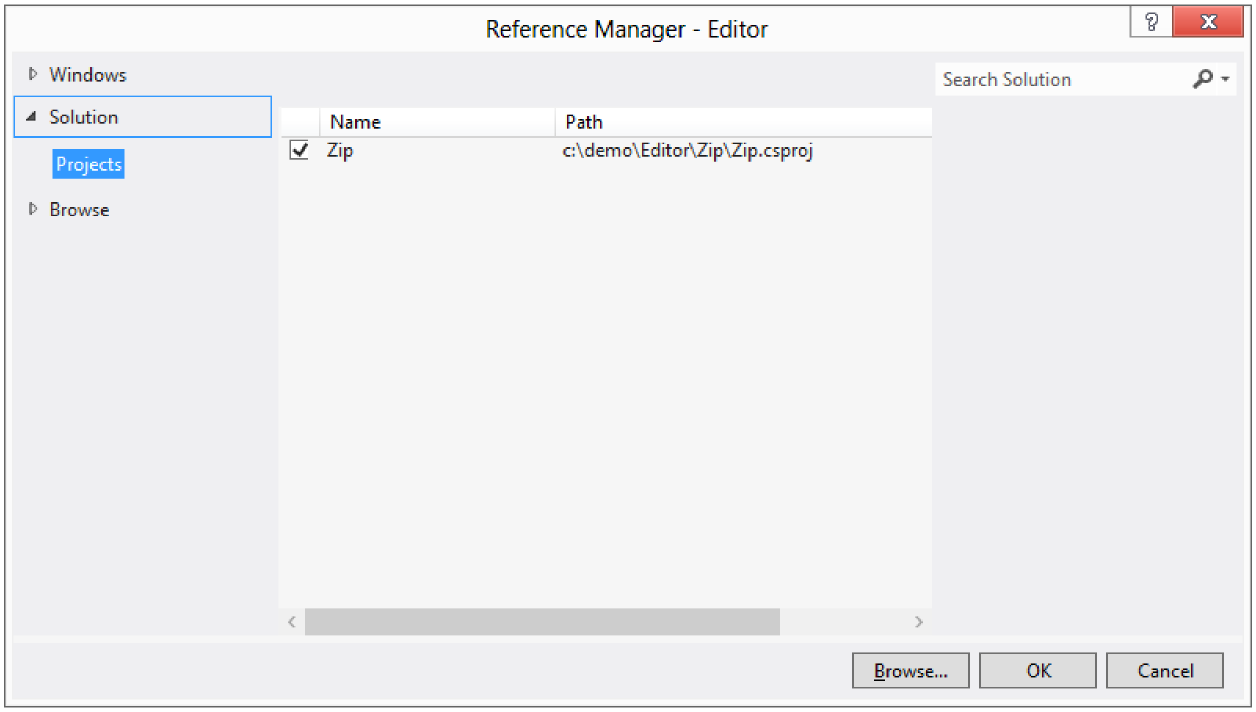
```
using System;
using System.IO;
using System.IO.Compression;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Threading.Tasks;
using Windows.Foundation;
using Windows.Storage;

namespace Zip
{
    public sealed class Zip
    {
        // Inefficient copy
        private static string CopyStreamToString(Stream stream)
        {
            using (var reader = new StreamReader(stream))
            {
                return reader.ReadToEnd();
            }
        }
        private async static Task<MemoryStream> CopyToMemoryStream(StorageFile
file) {
            var memoryStream = new MemoryStream();
            using (var stream = await file.OpenStreamForReadAsync())
            {
                stream.CopyTo(memoryStream);
                memoryStream.Seek(0, SeekOrigin.Begin);
            }
            return memoryStream;
        }

        // Private awaitable async function
        private static async Task<string> ReadFirstFileInternal(StorageFile file)
        {
            using (var memoryStream = await CopyToMemoryStream(file))
            {
                using (var archive = new ZipArchive(memoryStream))
                {
                    // Return the first opened file
                    if (archive.Entries.Count > 0)
                    {
                        return CopyStreamToString(archive.Entries[0].Open());
                    }
                    else
                    {
                        return String.Empty; // default case where there aren't
any files
                    }
                }
            }
        }

        // Public exposed function that maps an IAsyncOperation<string> to
Task<string>
        public static IAsyncOperation<string> ReadFirstFile(StorageFile file)
        {
            return AsyncInfo.Run(token => Zip.ReadFirstFileInternal(file));
        }
    }
}
```

Let's focus on the **ReadFirstFileInternal** function. If you have any familiarity with the asynchronous programming features recently introduced in C#, you'll immediately recognize the use of both the **async** and the **await** keywords in this code sample. Since this method does a potentially lengthy operation, we want to make it asynchronous to conform to the fast and fluid principle of Metro style apps. Notice that we took some shortcuts in this function – we only open the first file in the .zip archive and we assume that it is a HTML file. A robust implementation with a UI to select the right file from the .zip archive is an exercise left to the reader ☺

Notice that the **ReadFirstFileInternal** method is private. It is called from the **ReadFirstFile** function, which is marked as a public method. By default, all public methods in a .NET WinMD assembly are callable using the WinRT calling convention. One additional thing to point out is that the WinRT calling convention mandates that asynchronous operations return an interface type of **IAsyncOperation<T>**. .NET asynchronous operations return a **Task<T>**. The **ReadFirstFile** method uses the **.AsAsyncOperation** extension method from the **System.Runtime.InteropServices.WindowsRuntime** namespace to adapt the .NET asynchronous calling convention to the WinRT asynchronous calling convention.

One additional point: if we look in the file system, we'll see that we actually compile the file into a Zip.winmd file (see c:\demo\Zip\bin\debug).

3. Add a reference from **Editor** to **ZipHelper**:

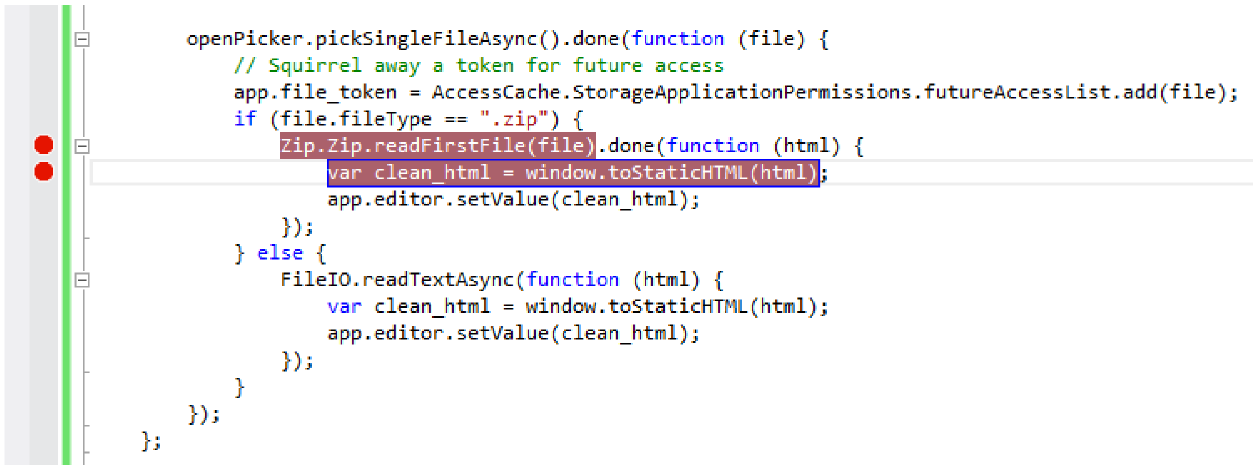


	Name	Path
✓	Zip	c:\demo\Editor\Zip\Zip.csproj

4. Replace the call to `Windows.Storage.FileIO.readTextAsync(file)` in **app.open_file** with Zip.Zip.open(file) to open a zip file:

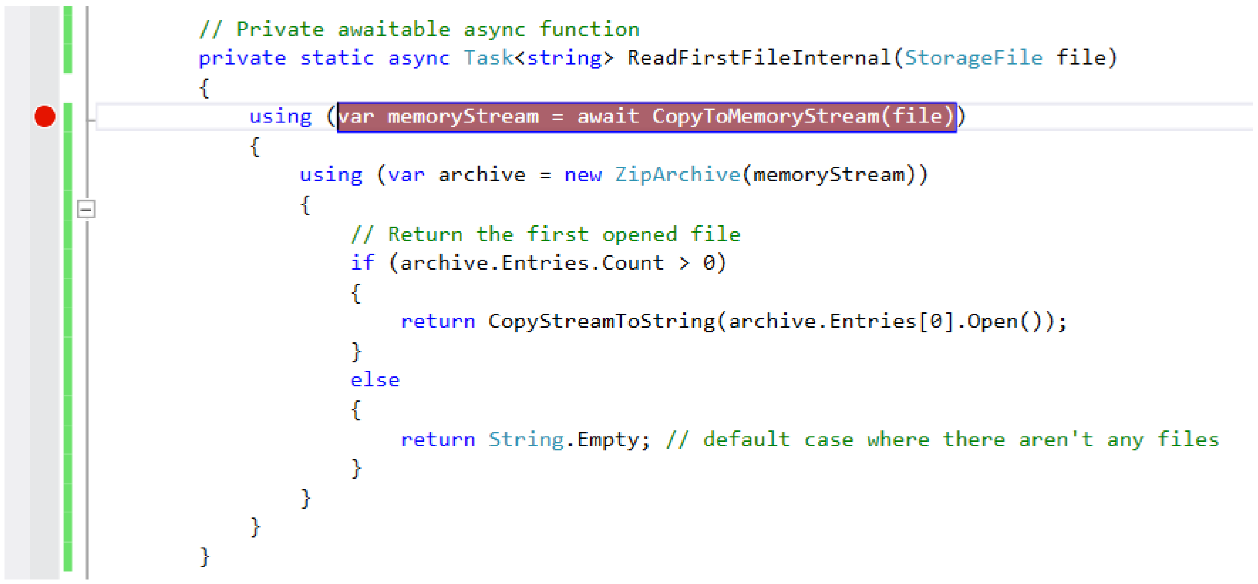
```
if (file.fileType == ".zip") {
    Zip.Zip.readFirstFile(file).done(function (html) {
        var clean_html = window.toStaticHTML(html);
        app.editor.setValue(clean_html);
    });
} else {
    Windows.Storage.FileIO.readTextAsync(function (html) {
        var clean_html = window.toStaticHTML(html);
        app.editor.setValue(clean_html);
    });
}
```

5. To see the WinRT calling convention, we can attach two debuggers to the process. Set two breakpoints, one on the call to Zip.Zip.readFirstFile() and one on the line inside of the callback function:



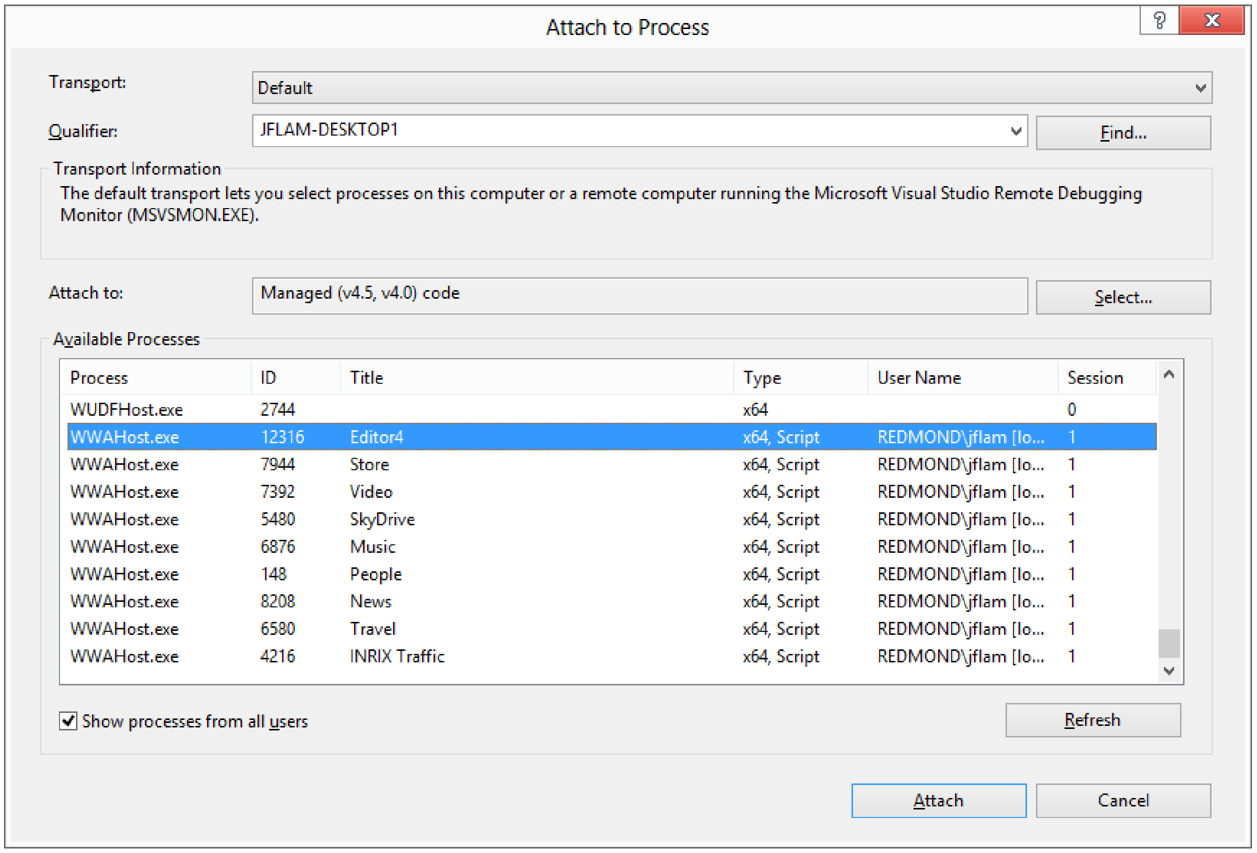
```
        openPicker.pickSingleFileAsync().done(function (file) {
            // Squirrel away a token for future access
            app.file_token = AccessCache.StorageApplicationPermissions.futureAccessList.add(file);
            if (file.fileType == ".zip") {
                Zip.Zip.readFirstFile(file).done(function (html) {
                    var clean_html = window.toStaticHTML(html);
                    app.editor.setValue(clean_html);
                });
            } else {
                FileIO.readTextAsync(function (html) {
                    var clean_html = window.toStaticHTML(html);
                    app.editor.setValue(clean_html);
                });
            }
        });
    };
```

6. Open a new instance of Visual Studio and open your Editor.sln project in it. Open the Zip.cs file and set a breakpoint on the first line of code in the readFirstFileInternal function:



```
        // Private awaitable async function
        private static async Task<string> ReadFirstFileInternal(StorageFile file)
        {
            using (var memoryStream = await CopyToMemoryStream(file))
            {
                using (var archive = new ZipArchive(memoryStream))
                {
                    // Return the first opened file
                    if (archive.Entries.Count > 0)
                    {
                        return CopyStreamToString(archive.Entries[0].Open());
                    }
                    else
                    {
                        return String.Empty; // default case where there aren't any files
                    }
                }
            }
        }
```

7. Hit F5 in the first instance of VS. Make sure that the script debugger is selected.
8. Now select the Debug … Attach to process menu option. Make sure that the Managed (v4.5, v4.0) code debugger option is selected in the Attach to drop-down and that your app (hosted inside of WWAHost.exe – Editor app is selected.


Attach to Process

Transport: Default

Qualifier: JFLAM-DESKTOP1

Transport Information

The default transport lets you select processes on this computer or a remote computer running the Microsoft Visual Studio Remote Debugging Monitor (MSVSMON.EXE).

Attach to: Managed (v4.5, v4.0) code

Available Processes

Process	ID	Title	Type	User Name	Session
WUDFHost.exe	2744		x64		0
WWAHost.exe	12316	Editor4	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	7944	Store	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	7392	Video	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	5480	SkyDrive	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	6876	Music	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	148	People	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	8208	News	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	6580	Travel	x64, Script	REDMOND\jflam [lo...	1
WWAHost.exe	4216	INRIX Traffic	x64, Script	REDMOND\jflam [lo...	1

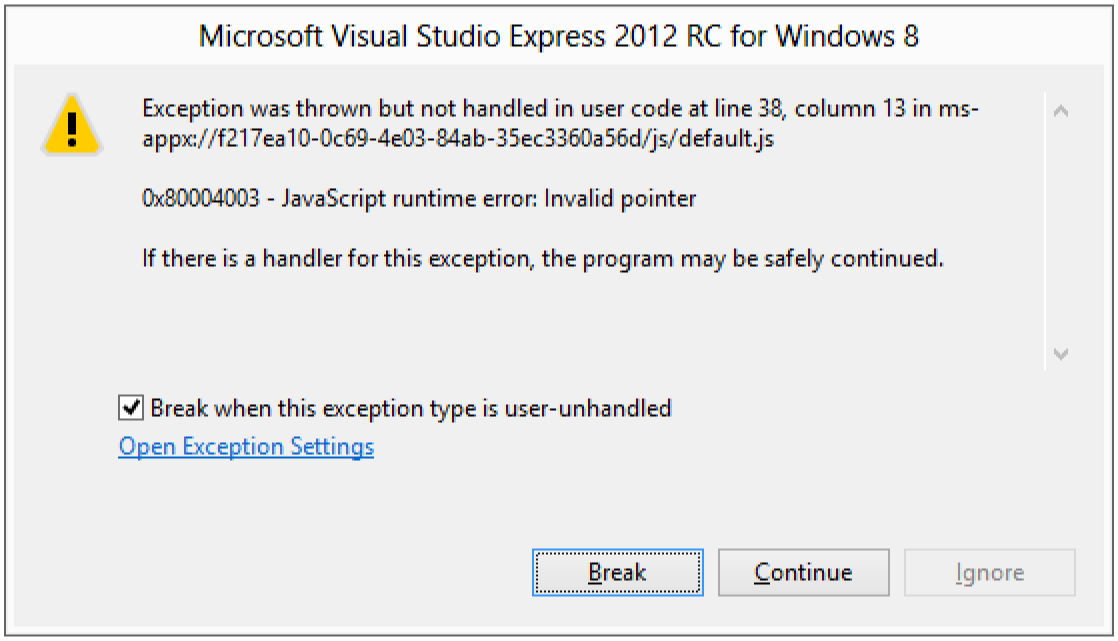
☑ Show processes from all users

9. Now step into the app first function by opening a file. You should see that you are stopped at the breakpoint on Zip.open line. Now hit F5 to continue. You should now see that you are stopped in the other instance of Visual Studio in the C# breakpoint. Now hit F5 again and you should re-emerge on the other side in the JS delegate.

Error handling

Let's see what happens when our app does something that we don't expect. There is a bug in the app – it will crash with a null reference exception if we click cancel instead of clicking on the open button in the File Open Picker.

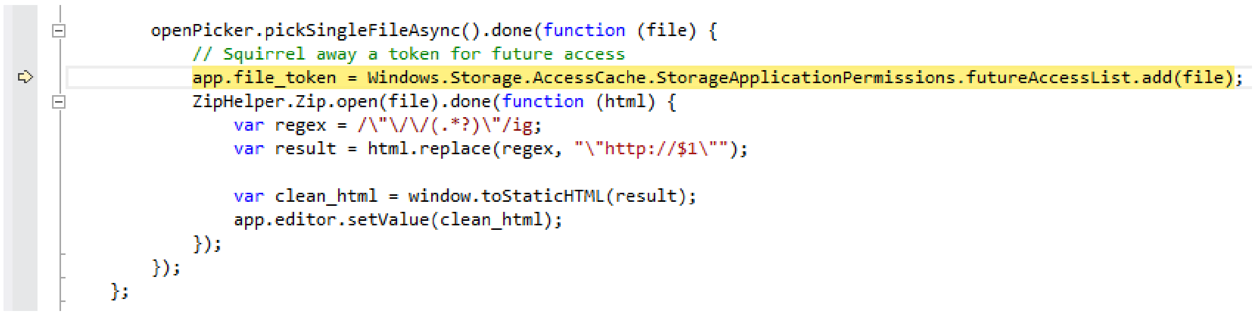
1. Run the app using F5 in Visual Studio.
2. Open a file open dialog box and click on the cancel button. You should see this appear in VS with the line of code highlighted in the editor:


Microsoft Visual Studio Express 2012 RC for Windows 8

Exception was thrown but not handled in user code at line 38, column 13 in ms-appx://f217ea10-0c69-4e03-84ab-35ec3360a56d/js/default.js

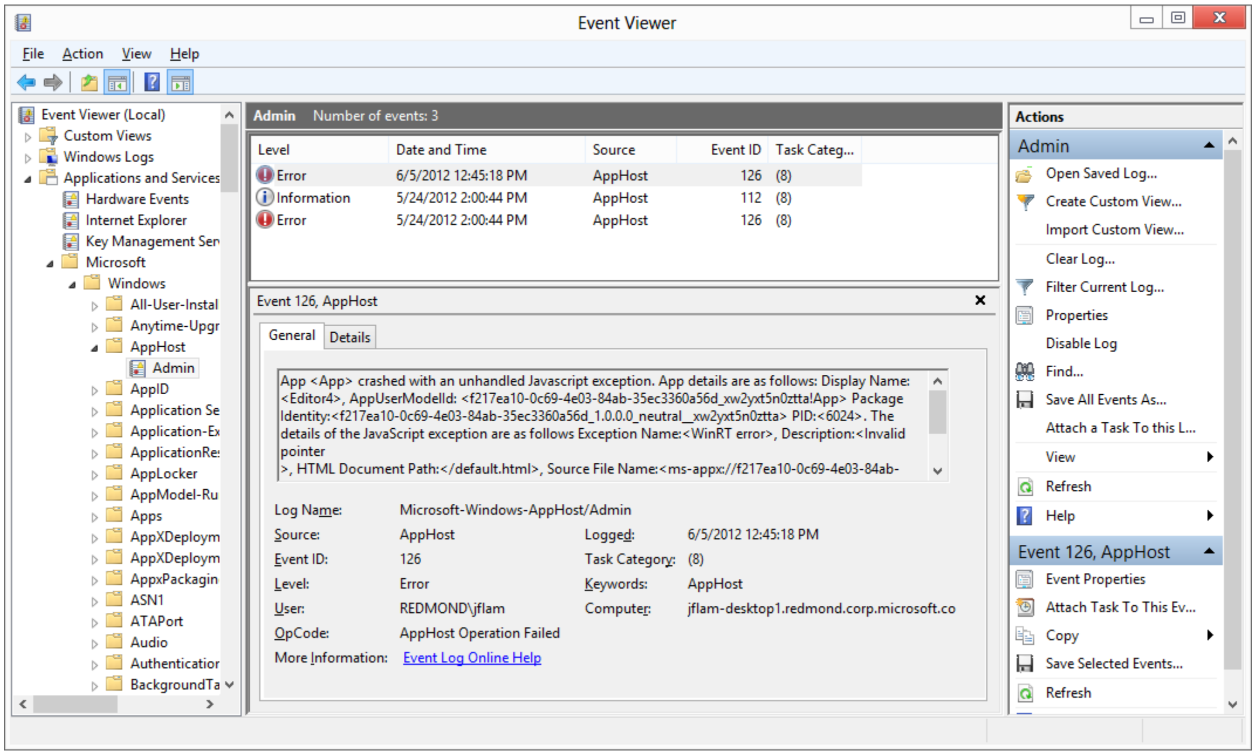
0x80004003 - JavaScript runtime error: Invalid pointer

If there is a handler for this exception, the program may be safely continued.



```
        openPicker.pickSingleFileAsync().done(function (file) {
            // Squirrel away a token for future access
            app.file_token = Windows.Storage.AccessCache.StorageApplicationPermissions.futureAccessList.add(file);
            ZipHelper.Zip.open(file).done(function (html) {
                var regex = /\"\/\/(.*?)\"/ig;
                var result = html.replace(regex, "\"http://$1\"");

                var clean_html = window.toStaticHTML(result);
                app.editor.setValue(clean_html);
            });
        });
    };
```

3. Now let’s run the app without the debugger attached using CTRL-F5 in Visual Studio. When you click Cancel did you notice that the app just “disappeared” and you were left looking at the Start Screen? This is the crash experience that a user experiences.
4. If we were testing this on a machine that didn’t have Visual Studio running, how would we know what happened? Fortunately, when an app crashes there is useful information deposited on your machine. Since we are building a Metro style HTML app, we can use Event Viewer to see what happened.
5. Click on the Windows key and type **eventvwr**. This will launch Event Viewer.
6. Expand the Applications and Services event group and drill into **Microsoft/Windows/AppHost/Admin**



Admin Number of events: 3

Level	Date and Time	Source	Event ID	Task Categ...
Error	6/5/2012 12:45:18 PM	AppHost	126	(8)
Information	5/24/2012 2:00:44 PM	AppHost	112	(8)
Error	5/24/2012 2:00:44 PM	AppHost	126	(8)

Event 126, AppHost

General Details

App <App> crashed with an unhandled Javascript exception. App details are as follows: Display Name:<Editor4>, AppUserModelId: <f217ea10-0c69-4e03-84ab-35ec3360a56d_xw2yxt5n0ztta!App> Package Identity:<f217ea10-0c69-4e03-84ab-35ec3360a56d_1.0.0.0_neutral__xw2yxt5n0ztta> PID:<6024>. The details of the JavaScript exception are as follows Exception Name:<WinRT error>, Description:<Invalid pointer
>, HTML Document Path:</default.html>, Source File Name:<ms-appx://f217ea10-0c69-4e03-84ab-

Log Name:	Microsoft-Windows-AppHost/Admin		
Source:	AppHost	Logged:	6/5/2012 12:45:18 PM
Event ID:	126	Task Category:	(8)
Level:	Error	Keywords:	AppHost
User:	REDMOND\jflam	Computer:	jflam-desktop1.redmond.corp.microsoft.co
OpCode:	AppHost Operation Failed		
More Information:	Event Log Online Help		

7. Copy and paste the General section and you'll get some details of the JS exception that was thrown (lightly formatted):

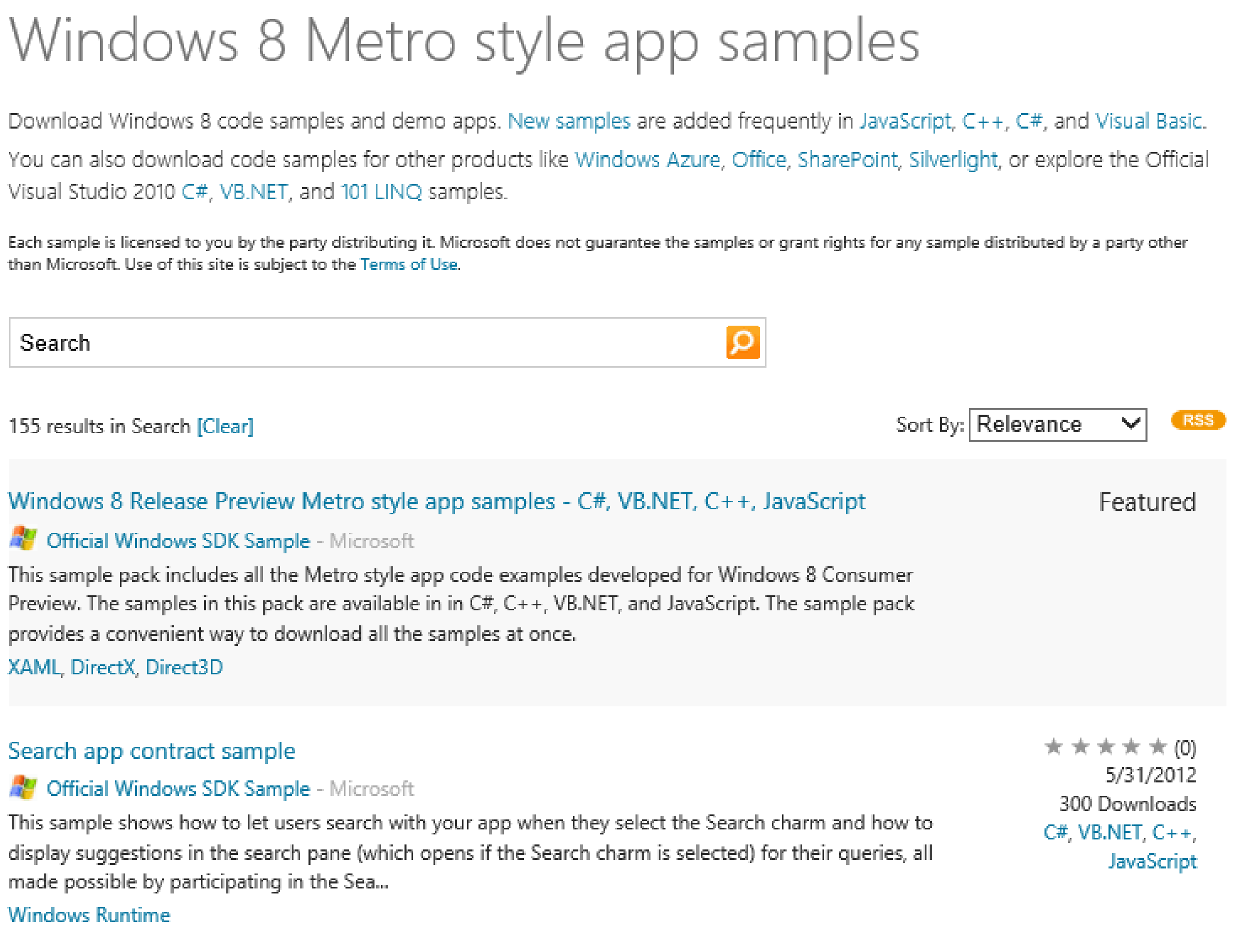
```
App <App> crashed with an unhandled Javascript exception.
App details are as follows: Display Name:<Editor4>,
AppUserModelId: <f217ea10-0c69-4e03-84ab-35ec3360a56d_xw2yxt5n0ztta!App>
Package Identity:<f217ea10-0c69-4e03-84ab-
35ec3360a56d_1.0.0.0_neutral__xw2yxt5n0ztta>
PID:<6024>.
The details of the JavaScript exception are as follows
Exception Name:<WinRT error>,
Description:<Invalid pointer>,
HTML Document Path:</default.html>,
Source File Name:<ms-appx://f217ea10-0c69-4e03-84ab-35ec3360a56d/default.html>,
Source Line Number:<157>,
Source Column Number:<28>,
and Stack Trace: ms-appx://f217ea10-0c69-4e03-84ab-35ec3360a56d/default.html:
157:28          Anonymous function().
```

Integrating the Windows Search experience

Integrating into Windows often involves adding support for one or more contracts. One of the most prominent new features of Windows 8 is the Search contract. It gives customers a consistent experience searching for content via the Search charm. Not only can you handle what the user types into the search box, you can also provide hints for what to search for as well. In this part of the app, we'll see how we can integrate this into our app.

We've gotten consistent feedback from developers that one of the first places that they turn to for help are the Windows Metro style samples on the Windows Developer Center.

1. Got to http://dev.windows.com and click on Metro style apps. You can search for the Search sample:



Windows 8 Metro style app samples

Download Windows 8 code samples and demo apps. New samples are added frequently in JavaScript, C++, C#, and Visual Basic. You can also download code samples for other products like Windows Azure, Office, SharePoint, Silverlight, or explore the Official Visual Studio 2010 C#, VB.NET, and 101 LINQ samples.

Each sample is licensed to you by the party distributing it. Microsoft does not guarantee the samples or grant rights for any sample distributed by a party other than Microsoft. Use of this site is subject to the Terms of Use.

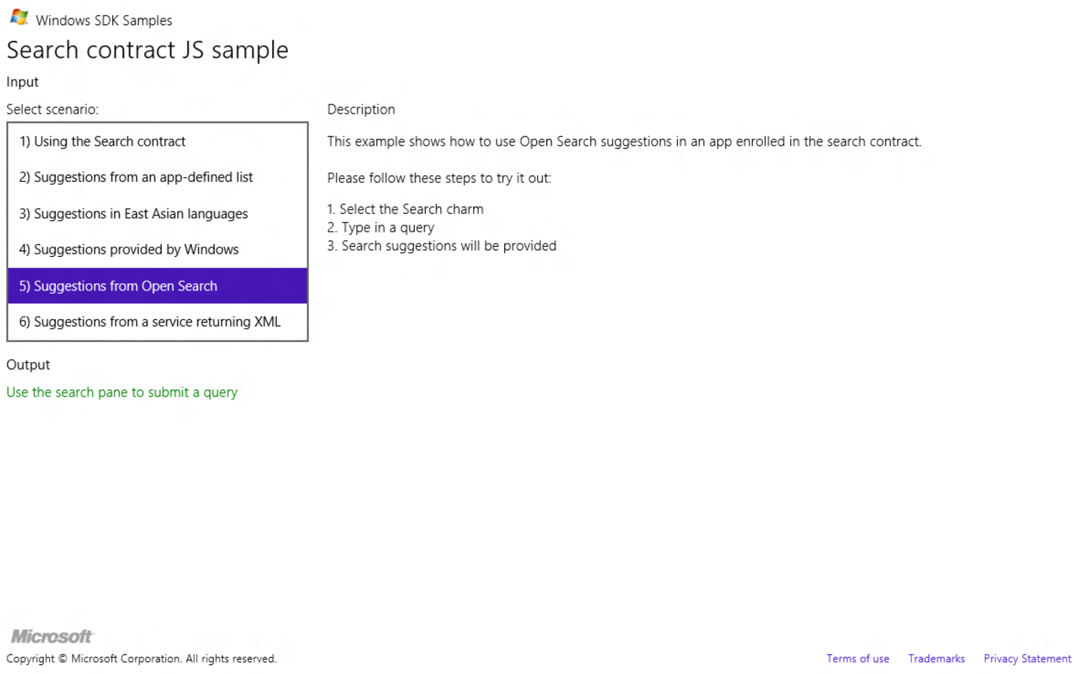
Search

155 results in Search [Clear]　Sort By: Relevance　RSS

Windows 8 Release Preview Metro style app samples - C#, VB.NET, C++, JavaScript　Featured
Official Windows SDK Sample - Microsoft
This sample pack includes all the Metro style app code examples developed for Windows 8 Consumer Preview. The samples in this pack are available in in C#, C++, VB.NET, and JavaScript. The sample pack provides a convenient way to download all the samples at once.
XAML, DirectX, Direct3D

Search app contract sample　★★★★★ (0) 5/31/2012 300 Downloads C#, VB.NET, C++, JavaScript
Official Windows SDK Sample - Microsoft
This sample shows how to let users search with your app when they select the Search charm and how to display suggestions in the search pane (which opens if the Search charm is selected) for their queries, all made possible by participating in the Sea...
Windows Runtime

2. Download the Javascript code for the Search app contract sample and open it up in Visual Studio and run via F5:


Windows SDK Samples

Search contract JS sample

Input

Select scenario:

1) Using the Search contract
2) Suggestions from an app-defined list
3) Suggestions in East Asian languages
4) Suggestions provided by Windows
5) Suggestions from Open Search
6) Suggestions from a service returning XML

Description

This example shows how to use Open Search suggestions in an app enrolled in the search contract.

Please follow these steps to try it out:

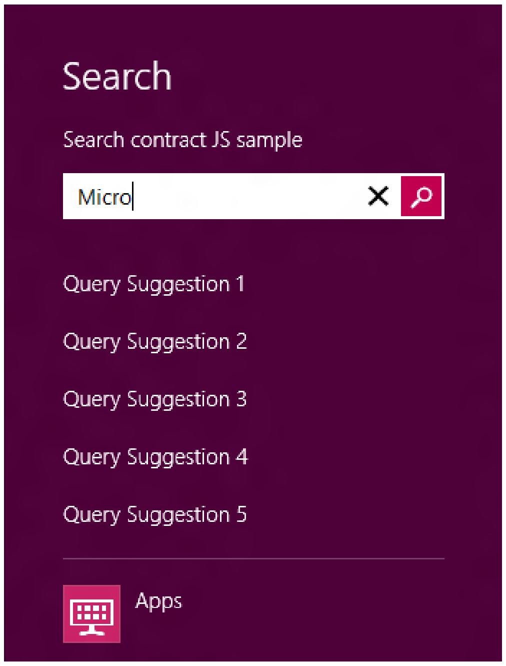
1. Select the Search charm
2. Type in a query
3. Search suggestions will be provided

Output

Use the search pane to submit a query

Copyright © Microsoft Corporation. All rights reserved.

3. The highlighted scenario (5) is the one that we are interested in: Suggestions from Open Search.
4. Run the sample and select scenario 5: Suggestions from Open Search
    a. Open search charm using Windows C
    b. Type something into the search box and see the suggestions appear



5. Open up the code for Scenario 5 – it’s under js\scenario5.js
6. Let’s extract the code for search suggestions and add it to our app. Copy the code that implements the **onsuggestionsrequested** event into your **app.onload** method.

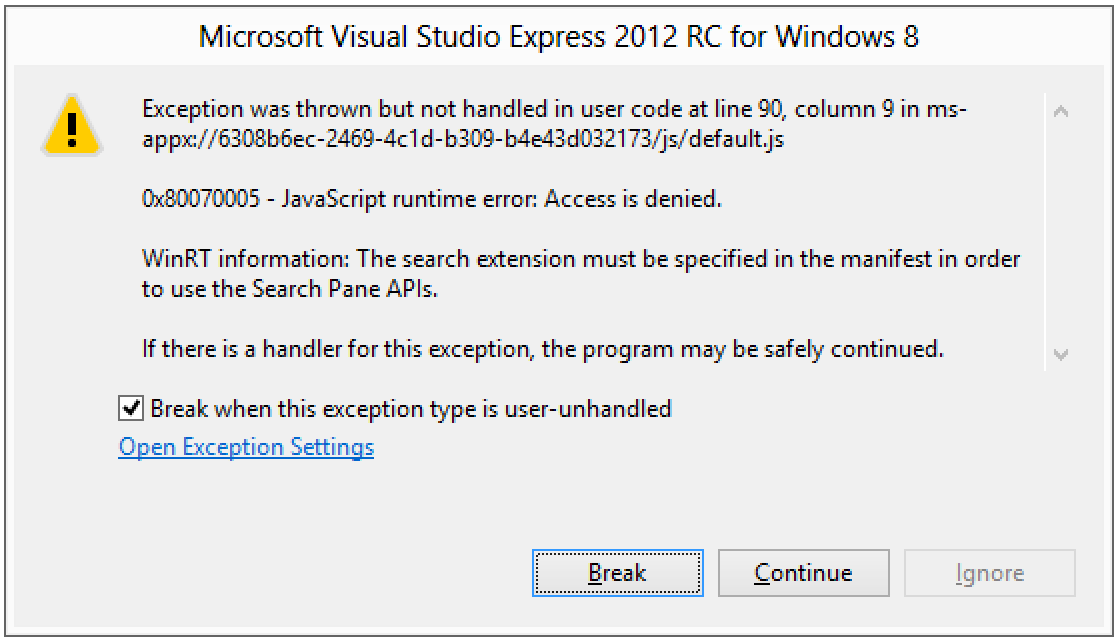
```
// Provide suggestions using an URL that supports the Open Search suggestion
format.
// Scenarios 2-6 introduce different methods of providing suggestions. The
registration for the onsuggestionsrequested
// event is added in a local scope for this sample's purpose, but in the common
case, you should place this code in the
// global scope, e.g. default.js, to run as soon as your app is launched. This way
your app can provide suggestions
// anytime the user brings up the search pane.
Windows.ApplicationModel.Search.SearchPane.getForCurrentView().onsuggestionsreques
ted = function (eventObject) {
    var queryText = eventObject.queryText, language = eventObject.language,
suggestionRequest = eventObject.request;

    // The deferral object is used to supply suggestions asynchronously for
example when fetching suggestions from a web service.
    // Indicate that we'll obtain suggestions asynchronously:
    var deferral = suggestionRequest.getDeferral();

    // This refers to a local package file that contains a sample JSON response.
    // You can update the Uri to a service that supports this standard in order to
see suggestions come from a web service.
    var suggestionUri = "jsonSuggestionService/exampleJsonResponse.json";
    // If you are using a webservice,the query string should be encoded into the
URI. See example below:
    //// suggestionUri += encodeURIComponent(queryText);

    // Cancel the previous suggestion request if it is not finished.
    if (xhrRequest && xhrRequest.cancel) {
        xhrRequest.cancel();
    }

    // Create request to obtain suggestions from service and supply them to the
Search Pane.
    xhrRequest = WinJS.xhr({ url: suggestionUri });
    xhrRequest.done(
        function (request) {
            if (request.responseText) {
                var parsedResponse = JSON.parse(request.responseText);
                if (parsedResponse && parsedResponse instanceof Array) {
                    var suggestions = parsedResponse[1];
                    if (suggestions) {

suggestionRequest.searchSuggestionCollection.appendQuerySuggestions(suggestions);
                        WinJS.log && WinJS.log("Suggestions provided for query: "
+ queryText, "sample", "status");
                    } else {
                        WinJS.log && WinJS.log("No suggestions provided for query:
" + queryText, "sample", "status");
                    }
                }
            }

            deferral.complete(); // Indicate we're done supplying suggestions.
        },
        function (error) {
            WinJS.log && WinJS.log("Error retrieving suggestions for query: " +
queryText, "sample", "status");
```

```
                // Call complete on the deferral when there is an error.
                deferral.complete();
            });
    };
```

7. Add this line to the method as well:

   ```
   var xhrRequest = null;
   ```

8. Modify **suggestionUri** to point to Wikipedia endpoint for Open Search: http://en.wikipedia.org/w/api.php?action=opensearch&search=
9. Uncomment the line of code that is below suggestionUri that will append a URI encoded query string to the URI.
10. Run the app. See this error:



Microsoft Visual Studio Express 2012 RC for Windows 8

Exception was thrown but not handled in user code at line 90, column 9 in ms-appx://6308b6ec-2469-4c1d-b309-b4e43d032173/js/default.js

0x80070005 - JavaScript runtime error: Access is denied.

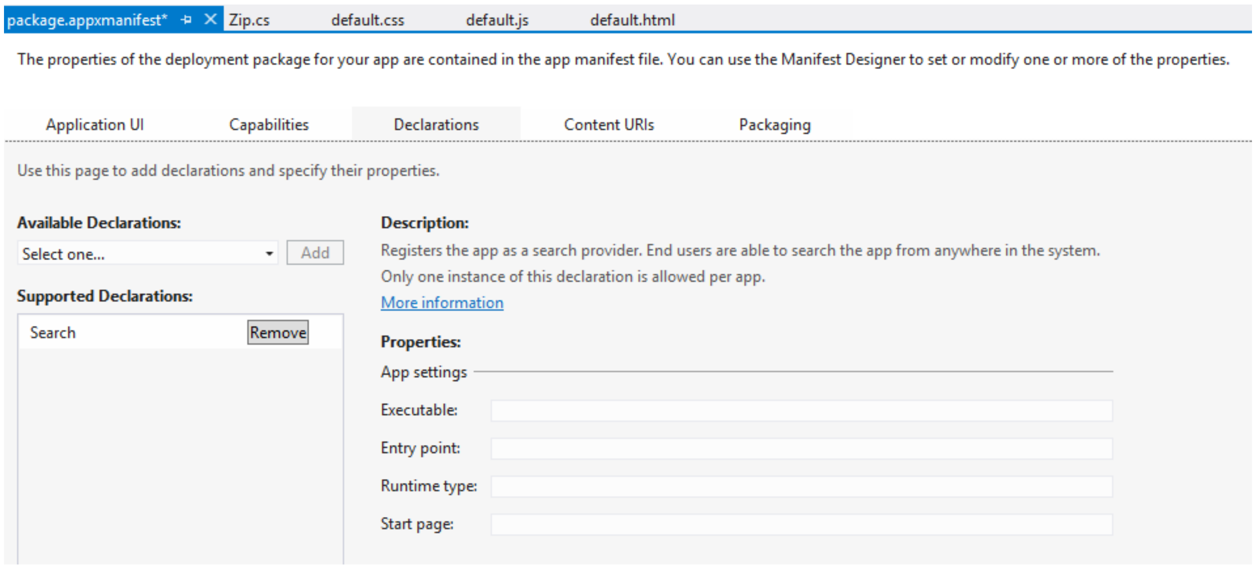
WinRT information: The search extension must be specified in the manifest in order to use the Search Pane APIs.

If there is a handler for this exception, the program may be safely continued.

☑ Break when this exception type is user-unhandled

Open Exception Settings

11. We need to remember to manifest the Search contract declaration. Double click on **package.appxmanifest** in the Solution Explorer. Click on the Declarations tab. Click on the Add button and select Search as a declaration



The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Use this page to add declarations and specify their properties.

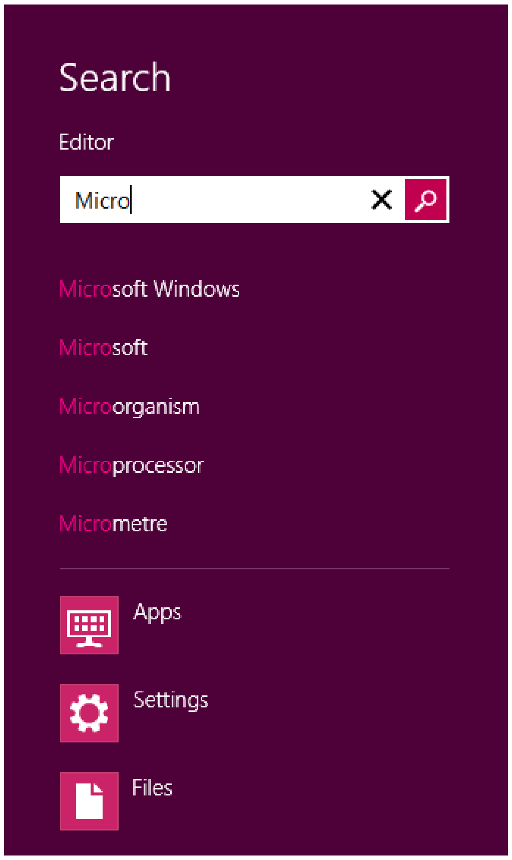
Available Declarations: Select one... Add

Supported Declarations: Search Remove

Description:
Registers the app as a search provider. End users are able to search the app from anywhere in the system.
Only one instance of this declaration is allowed per app.
More information

Properties:
App settings
Executable:
Entry point:
Runtime type:
Start page:

12. See the suggestions:



13. The final step in our app is adding the code that will retrieve a page from Wikipedia based on the search suggestion that was provided to the user. This lets us retrieve an article quickly without having to implement a full search results UI (that exercise is left to the reader ☺). We'll replace the existing code in **app.activate** with code that uses the **WinJS.xhr** object to retrieve

the HTML from Wikipedia. When your app is activated, Windows tells your app why it was activated. In this case we're going to add some code to handle the case where we were activated for **ActivationKind.search**.

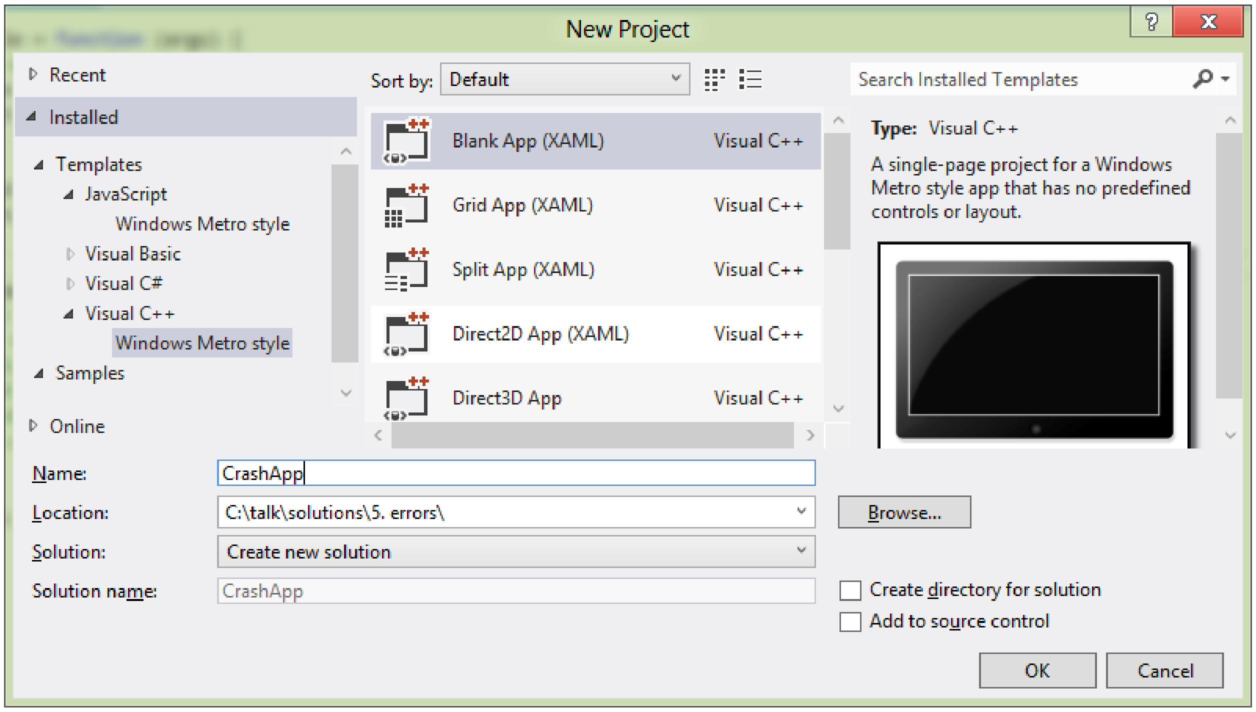
```
app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
        if (args.detail.previousExecutionState !==
activation.ApplicationExecutionState.terminated) {
            // TODO: This application has been newly launched. Initialize
            // your application here.
        } else {
            // TODO: This application has been reactivated from suspension.
            // Restore application state here.
        }
        args.setPromise(WinJS.UI.processAll());
    } else if (args.detail.kind === activation.ActivationKind.search) {
        var term = args.detail.queryText;
        var url = "http://en.wikipedia.org/wiki/" +
encodeURIComponent(term.replace(" ", "_"));
        WinJS.xhr({ url: url }).done(function (response) {
            if (response.responseText) {
                app.editor.setValue(response.responseText);
            }
        });
    }
};
```

Post mortem debugging

We saw earlier how you can post mortem debug a Javascript app. We can do the same thing for a native C++ application as well. In this case we actually have more information available, since for native applications we'll get a stack trace that contains locals and parameters as well.

1. Create a new Visual C++ XAML project

r


New Project

Sort by: Default

Search Installed Templates

- Recent
- Installed
  - Templates
    - JavaScript
      - Windows Metro style
    - Visual Basic
    - Visual C#
    - Visual C++
      - Windows Metro style
  - Samples
- Online

Blank App (XAML)	Visual C++
Grid App (XAML)	Visual C++
Split App (XAML)	Visual C++
Direct2D App (XAML)	Visual C++
Direct3D App	Visual C++

Type: Visual C++

A single-page project for a Windows Metro style app that has no predefined controls or layout.

Name:	CrashApp
Location:	C:\talk\solutions\5. errors\
Solution:	Create new solution
Solution name:	CrashApp

Browse...

Create directory for solution

Add to source control

OK Cancel

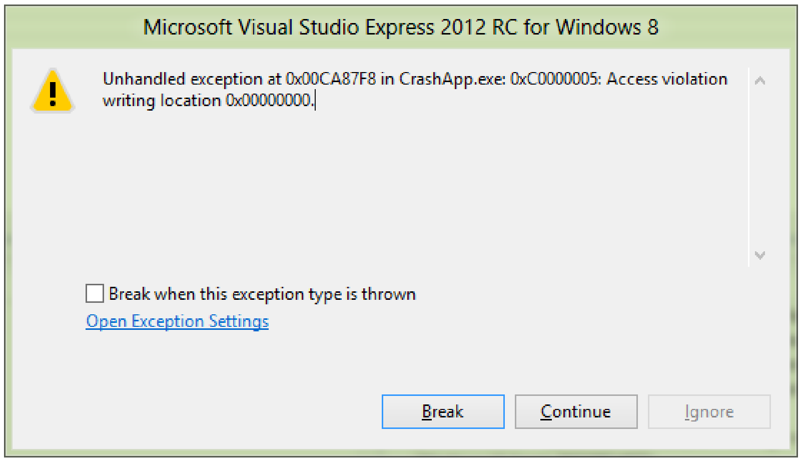
2. Add a button to the app bar:

```
    <Page.BottomAppBar>
        <AppBar >
            <Button x:Name="Save" Style="{StaticResource SaveAppBarButtonStyle}"
Click="Save_Click" />
        </AppBar>
    </Page.BottomAppBar>
```

3. Add some code to the event handler:

```
void CrashApp::MainPage::Save_Click(Platform::Object^ sender,
                                    Windows::UI::Xaml::RoutedEventArgs^ e)
{
    int *j = 0;
    *j = 42;
}
```

4. Run the app and click on the Save AppBar button. You should see this:

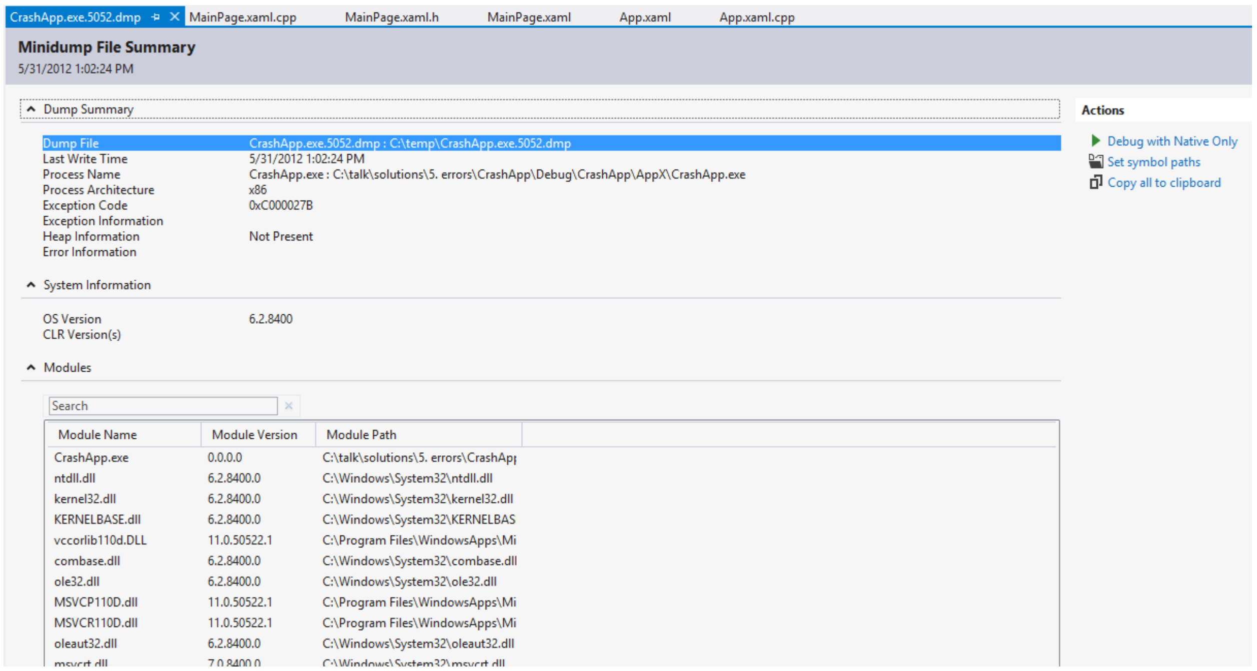

Microsoft Visual Studio Express 2012 RC for Windows 8

Unhandled exception at 0x00CA87F8 in CrashApp.exe: 0xC0000005: Access violation writing location 0x00000000.

5. Now, let's run the app outside of Visual Studio (you can do so using CTRL-F5). However, since this is a native application, we'll need to turn on the local dumps feature of Windows Error Reporting (WER) so that we can collect a local crash dump.
6. You can create a local dump using this .reg file:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error
Reporting\LocalDumps]
"DumpFolder"="c:\\temp"
"DumpType"=dword:00000001
```

7. The **DumpFolder** string value identifies the directory where you want the dumps to be created in. The **DumpType** DWORD value tells WER that we want to collect a mini dump. For more details see the MSDN doc page for activating Local Dumps.
8. You'll find the dump in your c:\temp directory. It will be named CrashApp.exe.NNNN.dmp. Open the dump using Visual Studio:



Minidump File Summary
5/31/2012 1:02:24 PM

Dump Summary

Dump File	CrashApp.exe.5052.dmp : C:\temp\CrashApp.exe.5052.dmp
Last Write Time	5/31/2012 1:02:24 PM
Process Name	CrashApp.exe : C:\talk\solutions\5. errors\CrashApp\Debug\CrashApp\AppX\CrashApp.exe
Process Architecture	x86
Exception Code	0xC000027B
Exception Information	
Heap Information	Not Present
Error Information	

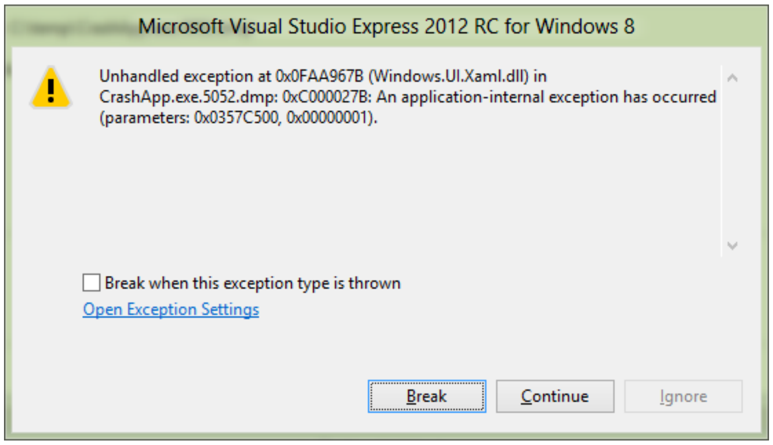
System Information

OS Version	6.2.8400
CLR Version(s)	

Modules

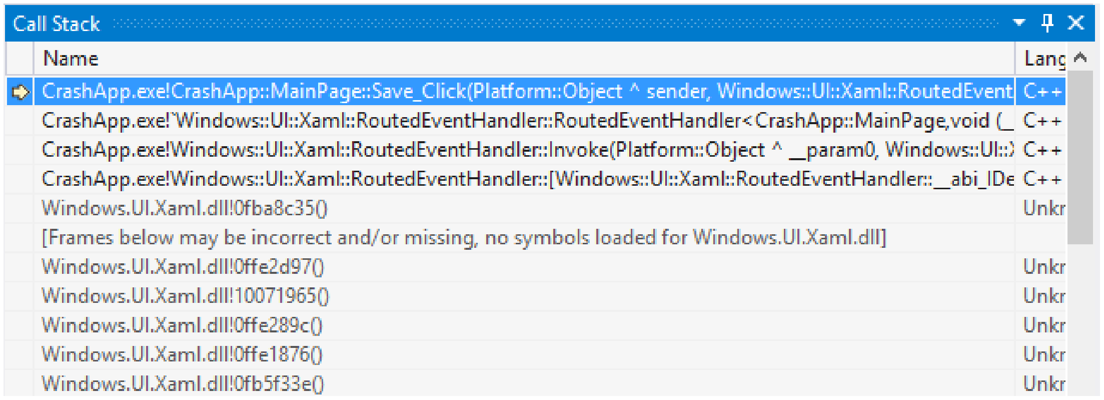
Module Name	Module Version	Module Path
CrashApp.exe	0.0.0.0	C:\talk\solutions\5. errors\CrashApp
ntdll.dll	6.2.8400.0	C:\Windows\System32\ntdll.dll
kernel32.dll	6.2.8400.0	C:\Windows\System32\kernel32.dll
KERNELBASE.dll	6.2.8400.0	C:\Windows\System32\KERNELBAS
vccorlib110d.DLL	11.0.50522.1	C:\Program Files\WindowsApps\Mi
combase.dll	6.2.8400.0	C:\Windows\System32\combase.dll
ole32.dll	6.2.8400.0	C:\Windows\System32\ole32.dll
MSVCP110D.dll	11.0.50522.1	C:\Program Files\WindowsApps\Mi
MSVCR110D.dll	11.0.50522.1	C:\Program Files\WindowsApps\Mi
oleaut32.dll	6.2.8400.0	C:\Windows\System32\oleaut32.dll
msvcrt.dll	7.0.8400.0	C:\Windows\System32\msvcrt.dll

9. When you click on the Debug with Native Only button, you'll see this dialog appear:


Microsoft Visual Studio Express 2012 RC for Windows 8

Unhandled exception at 0x0FAA967B (Windows.UI.Xaml.dll) in CrashApp.exe.5052.dmp: 0xC000027B: An application-internal exception has occurred (parameters: 0x0357C500, 0x00000001).

10. This is the same as if the app had crashed locally under the debugger. Notice how the call stack is the same as in the live debugging session:

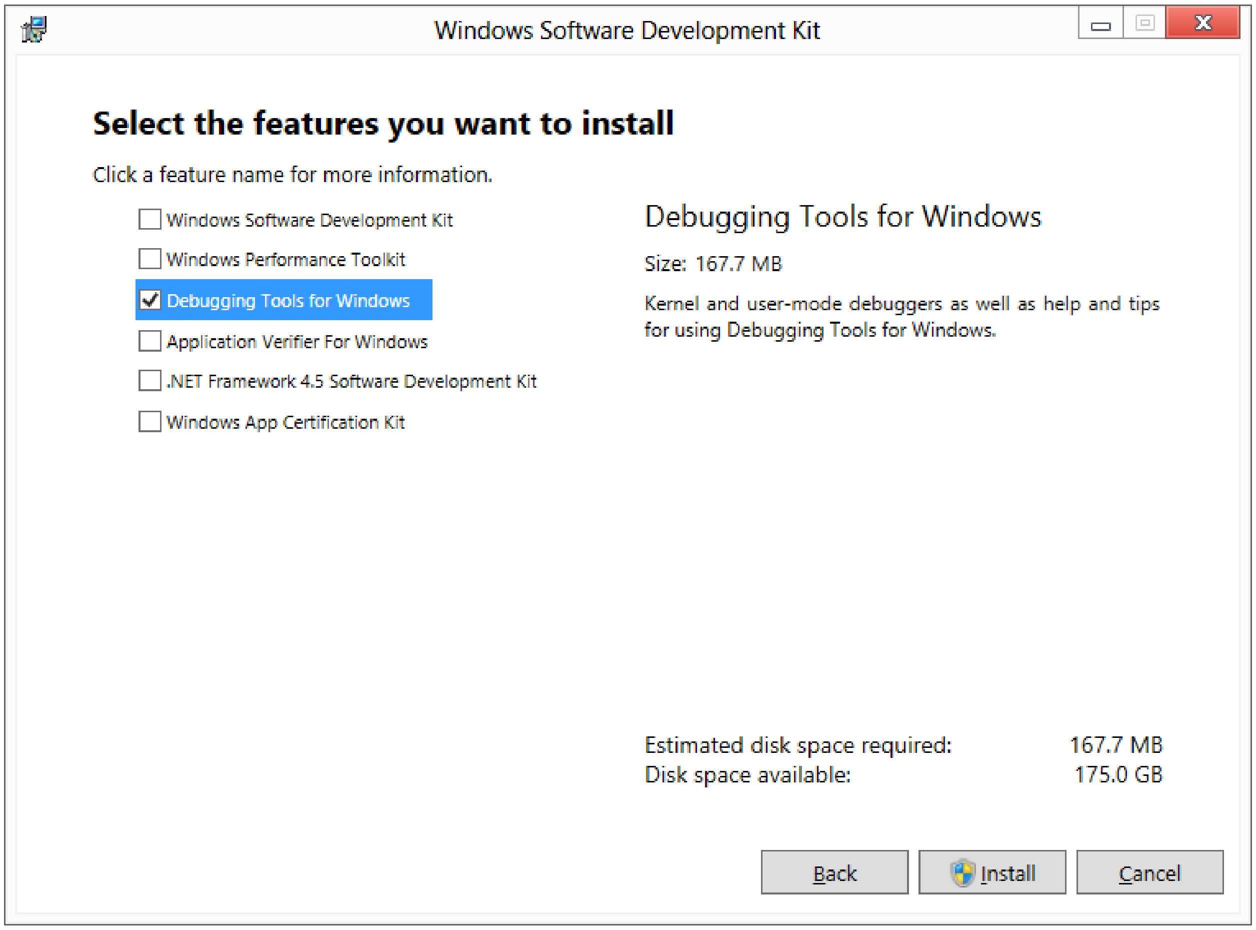


Name	Lang
CrashApp.exe!CrashApp::MainPage::Save_Click(Platform::Object ^ sender, Windows::UI::Xaml::RoutedEvent	C++
CrashApp.exe!`Windows::UI::Xaml::RoutedEventHandler::RoutedEventHandler<CrashApp::MainPage,void (_	C++
CrashApp.exe!Windows::UI::Xaml::RoutedEventHandler::Invoke(Platform::Object ^ __param0, Windows::UI::)	C++
CrashApp.exe!Windows::UI::Xaml::RoutedEventHandler::[Windows::UI::Xaml::RoutedEventHandler::__abi_IDe	C++
Windows.UI.Xaml.dll!0fba8c35()	Unkr
[Frames below may be incorrect and/or missing, no symbols loaded for Windows.UI.Xaml.dll]	
Windows.UI.Xaml.dll!0ffe2d97()	Unkr
Windows.UI.Xaml.dll!10071965()	Unkr
Windows.UI.Xaml.dll!0ffe289c()	Unkr
Windows.UI.Xaml.dll!0ffe1876()	Unkr
Windows.UI.Xaml.dll!0fb5f33e()	Unkr

11. However, it is not the same as live debugging since the dump does not contain heap data, only stack data. However, in many cases this is sufficient for diagnosing the problem and preparing a fix.

Crash dumps on an ARM device

1. To debug crash dumps that come from an ARM device, you'll need to install the Debugging Tools for Windows from the Windows Developer Center.



Windows Software Development Kit

Select the features you want to install

Click a feature name for more information.

- [ ] Windows Software Development Kit
- [ ] Windows Performance Toolkit
- [x] Debugging Tools for Windows
- [ ] Application Verifier For Windows
- [ ] .NET Framework 4.5 Software Development Kit
- [ ] Windows App Certification Kit

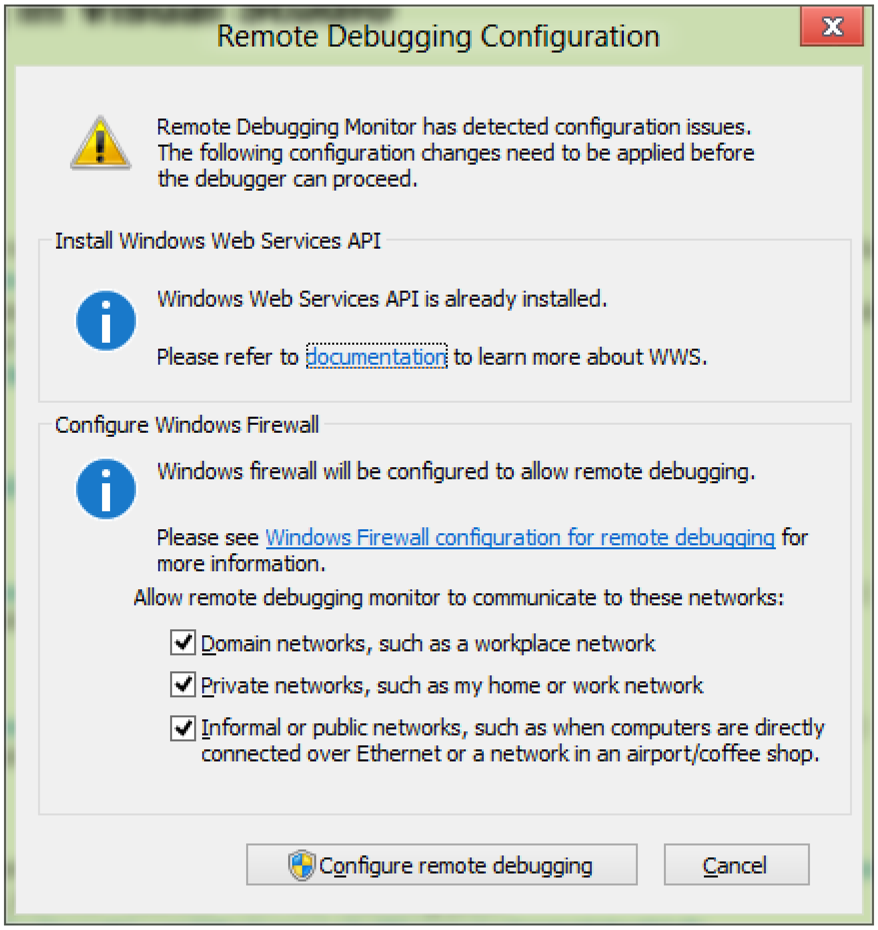
Debugging Tools for Windows

Size: 167.7 MB

Kernel and user-mode debuggers as well as help and tips for using Debugging Tools for Windows.

Estimated disk space required:	167.7 MB
Disk space available:	175.0 GB

2. This is not intended to be a guide to how to use WinDBG – the online documentation that ships with the tool is excellent. However, we will walk through how to do a quick dump analysis using the built-in features of WinDBG.
3. You'll need to setup the ARM device for remote debugging and deployment. First we need to install Remote Tools for Visual Studio 2012 RC. This will be pre-staged on the device since this tool isn't readily available now since ARM devices are not readily available now.
4. Start up the debugger on the device and keep an eye on the IP address and/or machine name. Depending on local network configuration I may be able to connect using machine name or IP
5. First run experience on device will pop up Firewall configuration. Accept defaults and continue.



Remote Debugging Configuration

Remote Debugging Monitor has detected configuration issues. The following configuration changes need to be applied before the debugger can proceed.

Install Windows Web Services API

Windows Web Services API is already installed.

Please refer to documentation to learn more about WWS.

Configure Windows Firewall

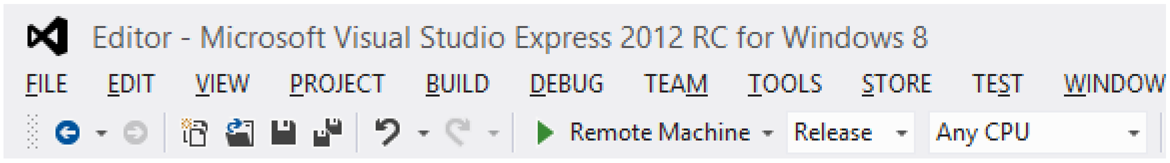
Windows firewall will be configured to allow remote debugging.

Please see Windows Firewall configuration for remote debugging for more information.

Allow remote debugging monitor to communicate to these networks:

- [✓] Domain networks, such as a workplace network
- [✓] Private networks, such as my home or work network
- [✓] Informal or public networks, such as when computers are directly connected over Ethernet or a network in an airport/coffee shop.

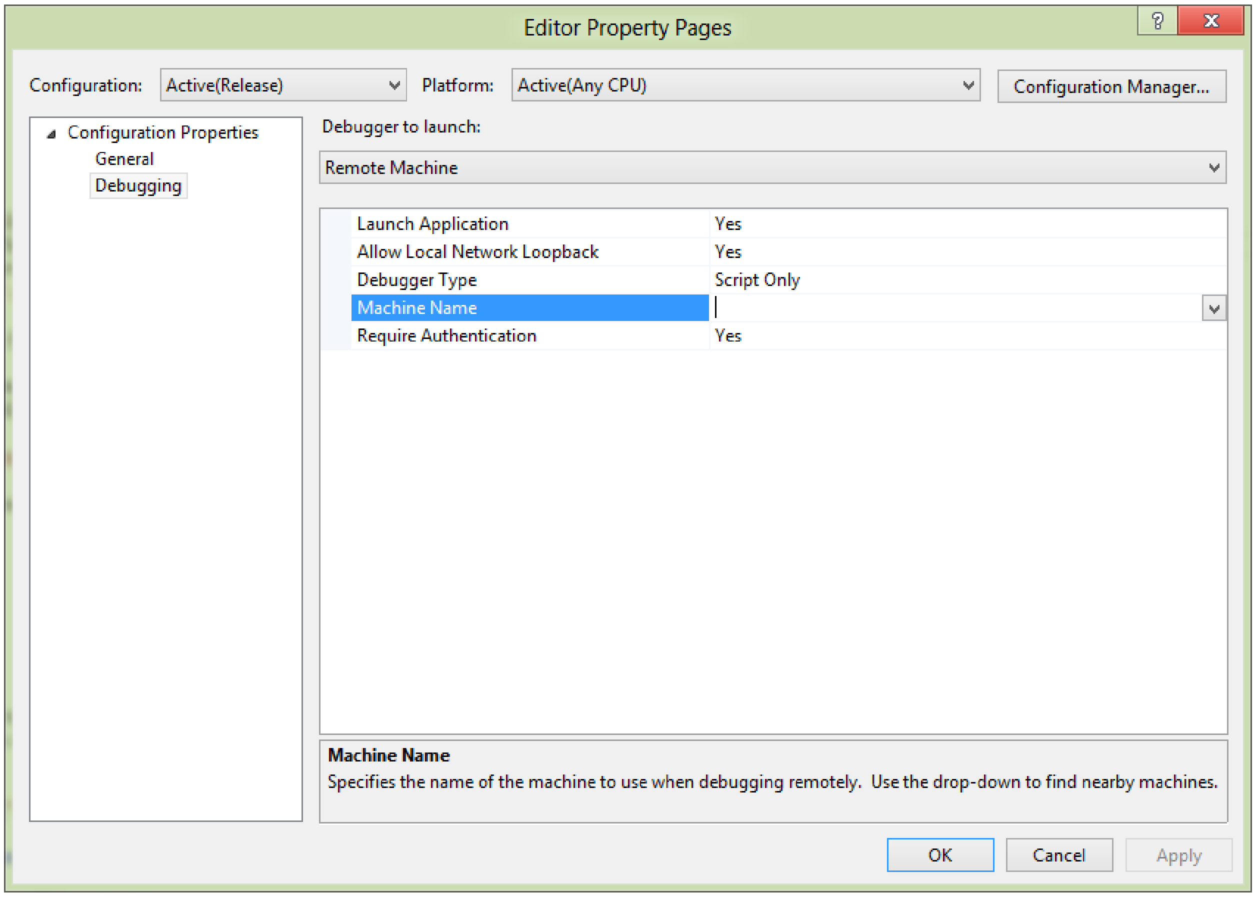
6. To deploy app from VS select Remote Machine from the toolbar:



Editor - Microsoft Visual Studio Express 2012 RC for Windows 8

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS STORE TEST WINDOW
Remote Machine
Release
Any CPU

7. Now run the app on the device by hitting F5. It will prompt you for the name of the device. Either get the IP address of the device or connect by name.



Configuration: Active(Release) Platform: Active(Any CPU)

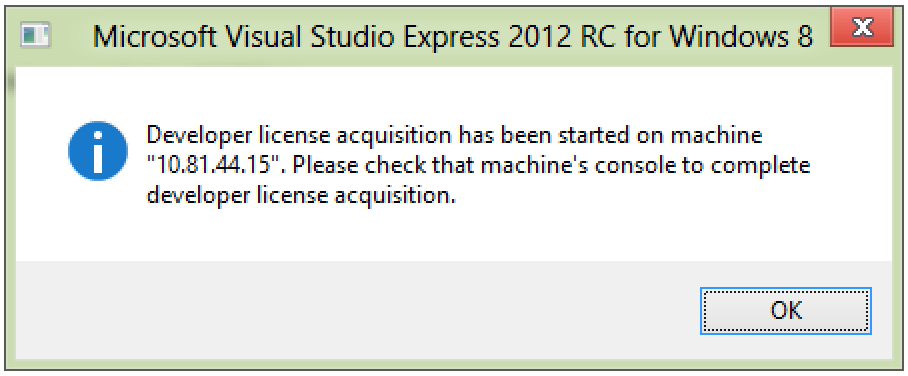
- Configuration Properties
  - General
  - Debugging

Debugger to launch: Remote Machine

Launch Application	Yes
Allow Local Network Loopback	Yes
Debugger Type	Script Only
Machine Name	
Require Authentication	Yes

Machine Name
Specifies the name of the machine to use when debugging remotely. Use the drop-down to find nearby machines.

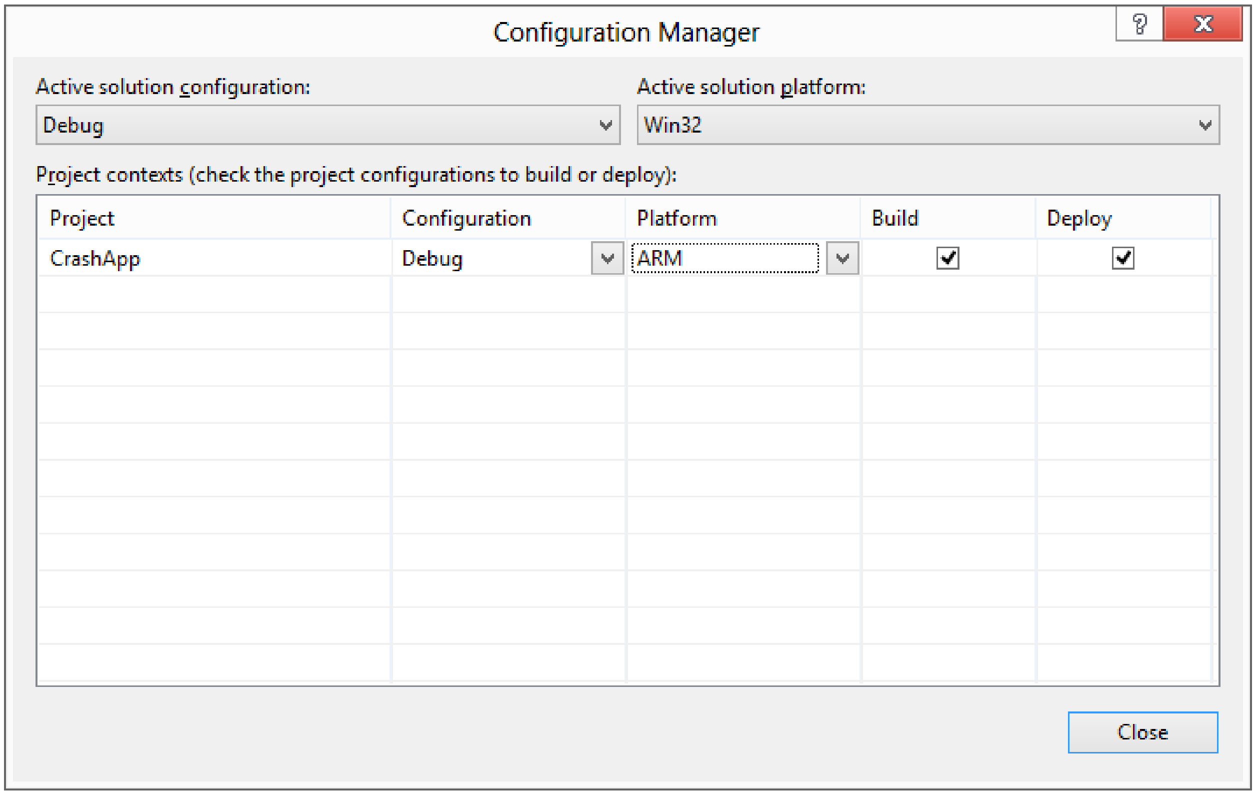
8. Type in your local account creds for the ARM device. If you are using a Microsoft account, use your Microsoft Account ID and password (it works!)
9. First run it will kick off a Get a developer license for Windows RT dialog on your machine. You will see this prompt:



Developer license acquisition has been started on machine "10.81.44.15". Please check that machine's console to complete developer license acquisition.

10. Enter your Microsoft account credentials to complete getting a developer license.
11. Switch back to your dev box and dismiss the dialog from before. You'll now see your app starting up on your ARM device.
12. If you click on the button you'll see the same crash experience inside of Visual Studio as you had before.

13. Now let's do a post mortem debug. Recompile using the ARM target and deploy your app to your ARM device. You'll need to switch your build configuration first:



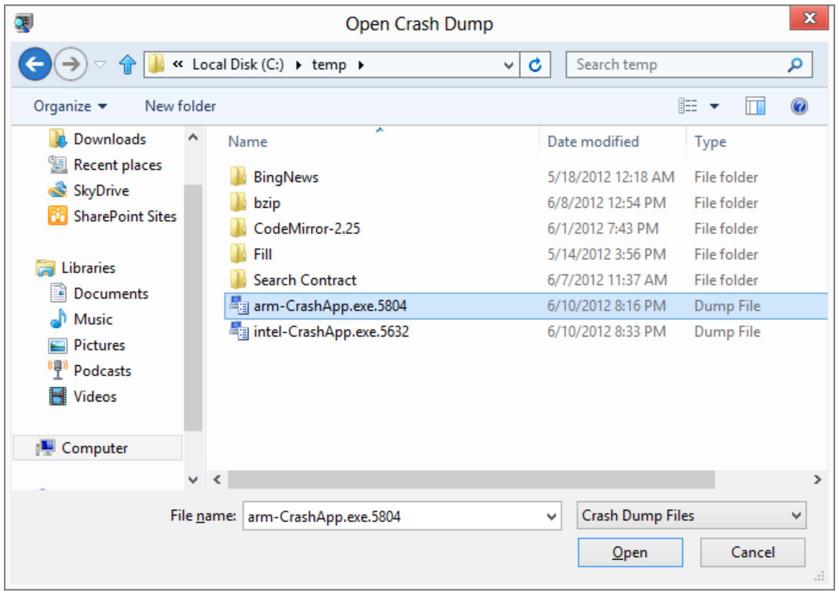
Active solution configuration: Debug

Active solution platform: Win32

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
CrashApp	Debug	ARM	✓	✓

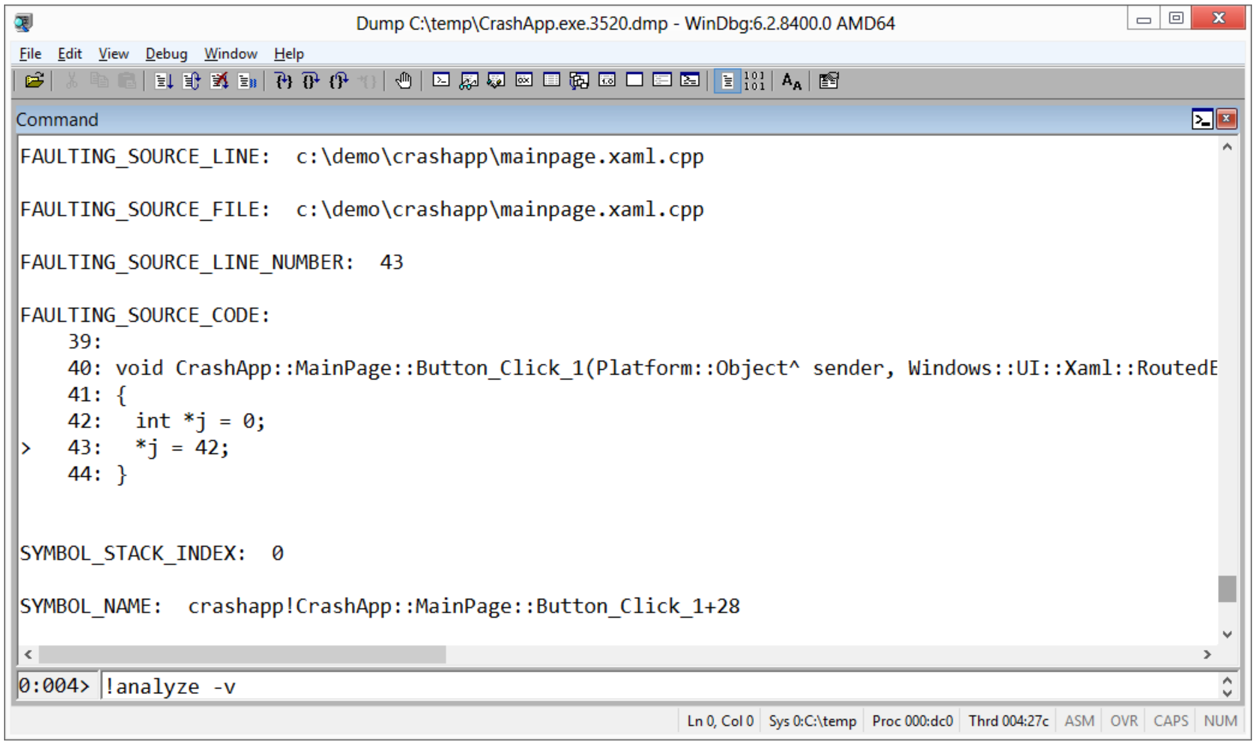
14. Switch to the ARM device and run the app by clicking on the tile from the Start screen. The ARM device must be configured to collect local dumps using the same registry settings as we used earlier.
15. Open up the dump file using WinDBG:



Name	Date modified	Type
BingNews	5/18/2012 12:18 AM	File folder
bzip	6/8/2012 12:54 PM	File folder
CodeMirror-2.25	6/1/2012 7:43 PM	File folder
Fill	5/14/2012 3:56 PM	File folder
Search Contract	6/7/2012 11:37 AM	File folder
arm-CrashApp.exe.5804	6/10/2012 8:16 PM	Dump File
intel-CrashApp.exe.5632	6/10/2012 8:33 PM	Dump File

File name: arm-CrashApp.exe.5804   Crash Dump Files

16. Type !analyze –verbose. This should give you a nice source-based dump of where the crash occurred:


Dump C:\temp\CrashApp.exe.3520.dmp - WinDbg:6.2.8400.0 AMD64
File Edit View Debug Window Help
Command

```
FAULTING_SOURCE_LINE:  c:\demo\crashapp\mainpage.xaml.cpp

FAULTING_SOURCE_FILE:  c:\demo\crashapp\mainpage.xaml.cpp

FAULTING_SOURCE_LINE_NUMBER:  43

FAULTING_SOURCE_CODE:
    39:
    40: void CrashApp::MainPage::Button_Click_1(Platform::Object^ sender, Windows::UI::Xaml::RoutedE
    41: {
    42:   int *j = 0;
>   43:   *j = 42;
    44: }


SYMBOL_STACK_INDEX:  0

SYMBOL_NAME:  crashapp!CrashApp::MainPage::Button_Click_1+28
```

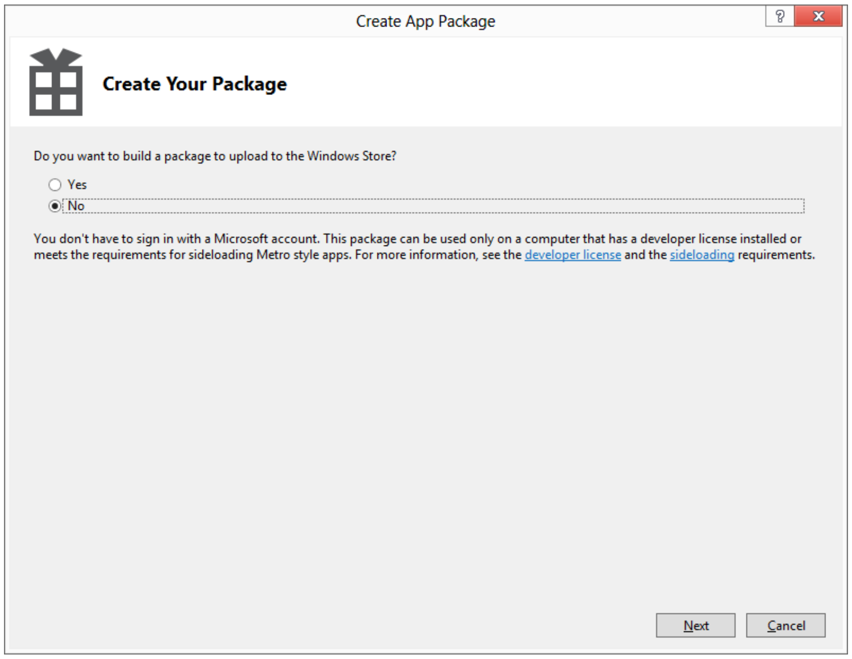
```
0:004> !analyze -v
```

Ln 0, Col 0 | Sys 0:C:\temp | Proc 000:dc0 | Thrd 004:27c | ASM | OVR | CAPS | NUM

Using the WACK

Now let’s get ready to submit our app to the store. Let’s run the packaging experience wizard from VS.

1. To prepare your app for upload to the store you’ll first need to create an app package. Do this by selecting Create App Package from the STORE menu in VS.
2. Make sure to compile as Release mode
3. Say No to the first dialog – you want to create a local package as opposed to a package that can be uploaded to the Store. For that to work you’ll need a Store developer account, and as of the time of this writing the Store is only open by invitation only.



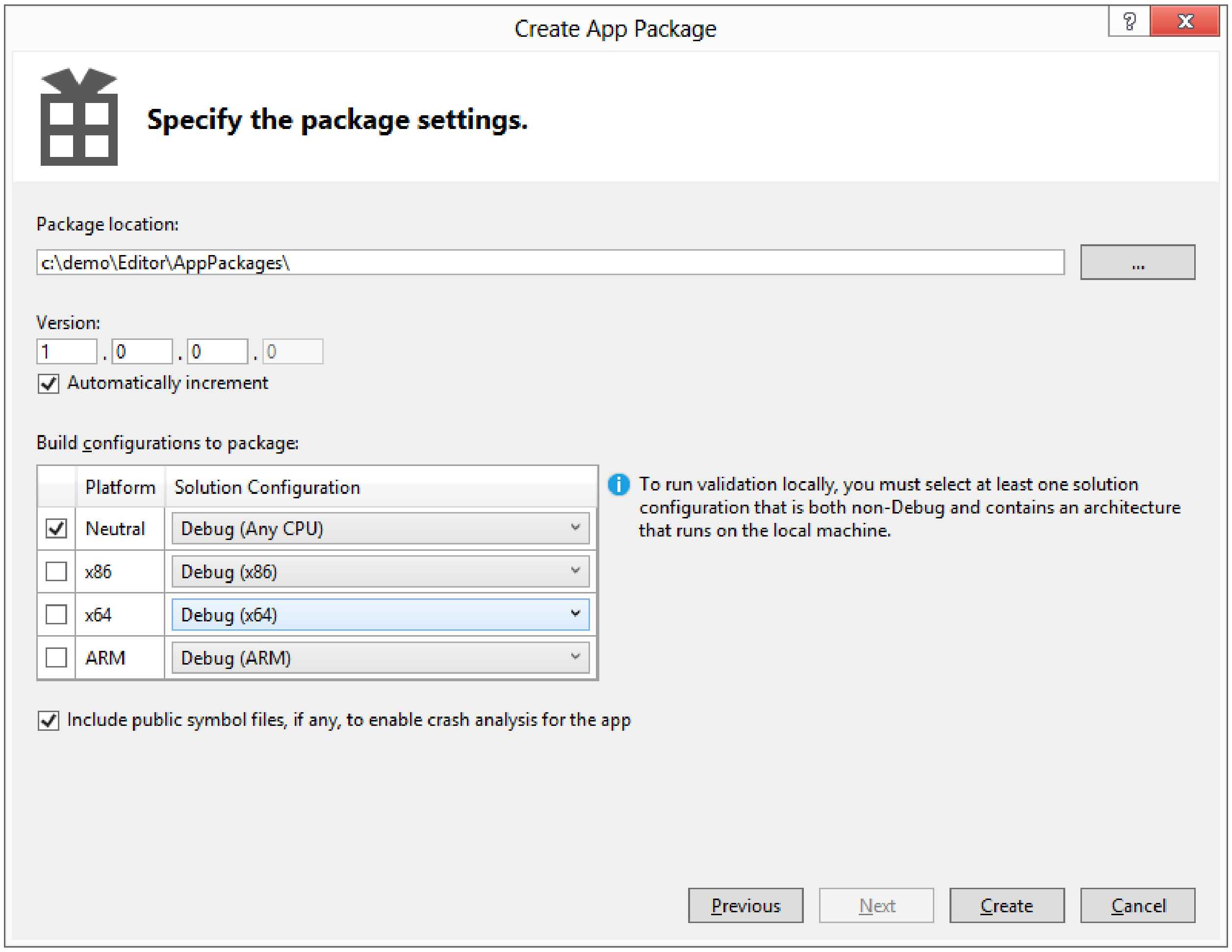
Create Your Package

Do you want to build a package to upload to the Windows Store?

Yes [✓ No]

You don't have to sign in with a Microsoft account. This package can be used only on a computer that has a developer license installed or meets the requirements for sideloading Metro style apps. For more information, see the developer license and the sideloading requirements.

4. Select the defaults on the next page:



Specify the package settings.

Package location: c:\demo\Editor\AppPackages\

Version: 1 . 0 . 0 . 0

☑ Automatically increment

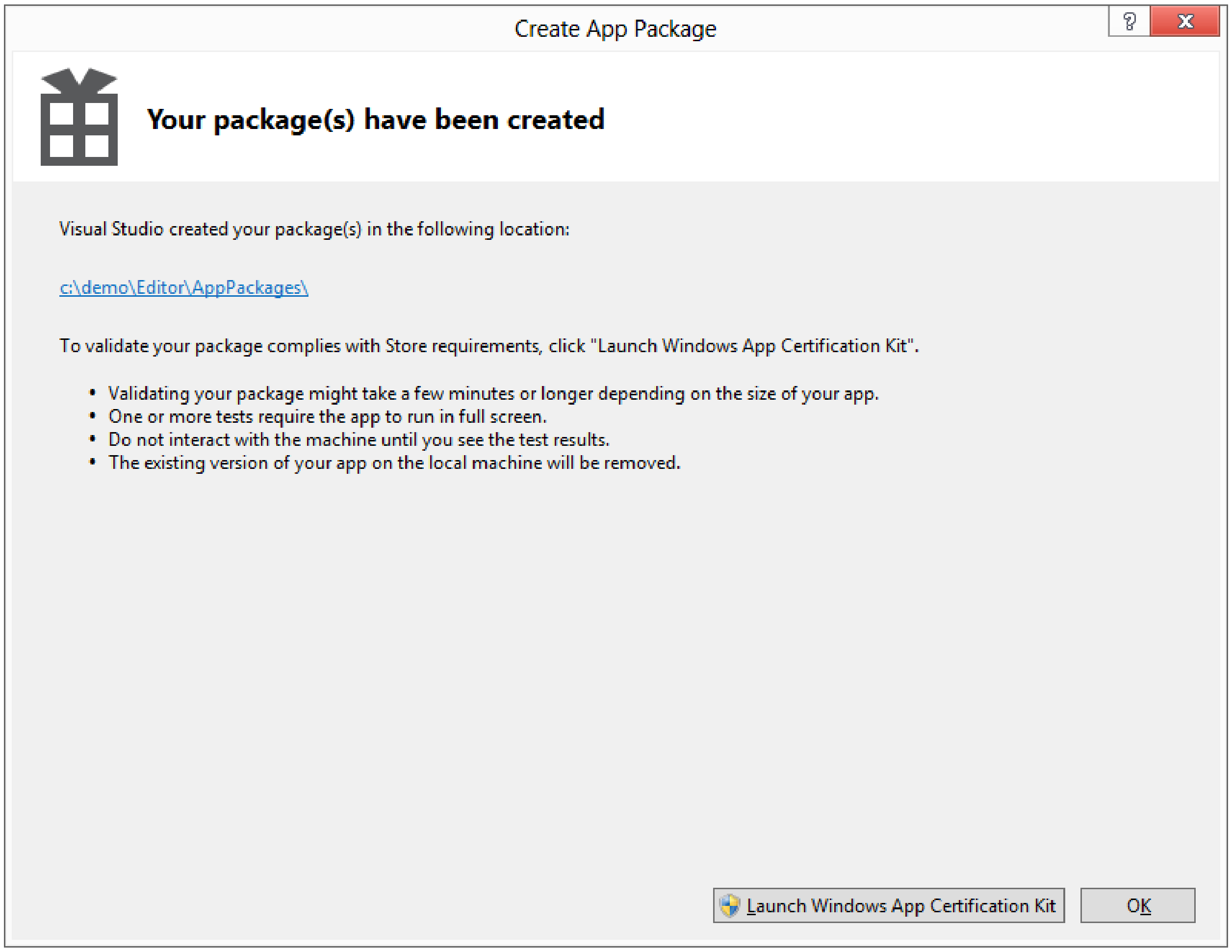
Build configurations to package:

	Platform	Solution Configuration
☑	Neutral	Debug (Any CPU)
☐	x86	Debug (x86)
☐	x64	Debug (x64)
☐	ARM	Debug (ARM)

To run validation locally, you must select at least one solution configuration that is both non-Debug and contains an architecture that runs on the local machine.

☑ Include public symbol files, if any, to enable crash analysis for the app

5. Click on the Create button to complete creating your package.
6. Click on the Launch Windows App Certification Kit button in the next dialog:



Create App Package

Your package(s) have been created

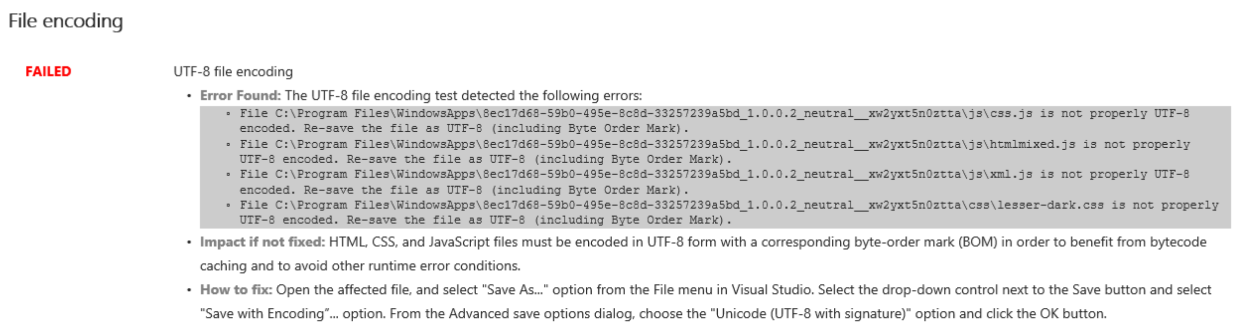
Visual Studio created your package(s) in the following location:

c:\demo\Editor\AppPackages\

To validate your package complies with Store requirements, click "Launch Windows App Certification Kit".

- Validating your package might take a few minutes or longer depending on the size of your app.
- One or more tests require the app to run in full screen.
- Do not interact with the machine until you see the test results.
- The existing version of your app on the local machine will be removed.

7. Now sit back and wait for the results. Note that encoding errors are also common if you use 3rd party libraries – you will need to open and re-save the files in Visual Studio using UTF-8 with a byte-order mark.



File encoding

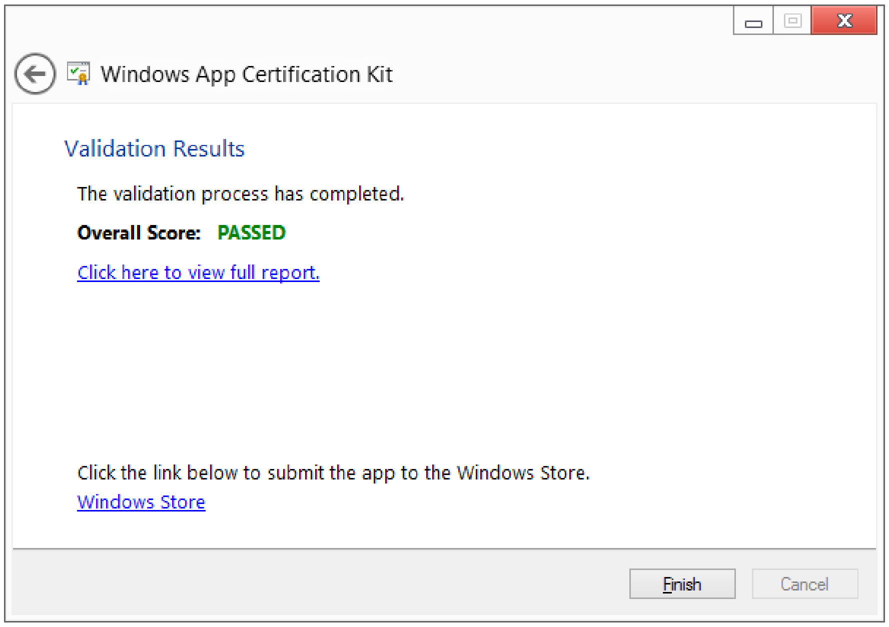
FAILED UTF-8 file encoding

- **Error Found:** The UTF-8 file encoding test detected the following errors:

```
▪ File C:\Program Files\WindowsApps\8ec17d68-59b0-495e-8c8d-33257239a5bd_1.0.0.2_neutral__xw2yxt5n0ztta\js\css.js is not properly UTF-8
  encoded. Re-save the file as UTF-8 (including Byte Order Mark).
▪ File C:\Program Files\WindowsApps\8ec17d68-59b0-495e-8c8d-33257239a5bd_1.0.0.2_neutral__xw2yxt5n0ztta\js\htmlmixed.js is not properly
  UTF-8 encoded. Re-save the file as UTF-8 (including Byte Order Mark).
▪ File C:\Program Files\WindowsApps\8ec17d68-59b0-495e-8c8d-33257239a5bd_1.0.0.2_neutral__xw2yxt5n0ztta\js\xml.js is not properly UTF-8
  encoded. Re-save the file as UTF-8 (including Byte Order Mark).
▪ File C:\Program Files\WindowsApps\8ec17d68-59b0-495e-8c8d-33257239a5bd_1.0.0.2_neutral__xw2yxt5n0ztta\css\lesser-dark.css is not properly
  UTF-8 encoded. Re-save the file as UTF-8 (including Byte Order Mark).
```

- **Impact if not fixed:** HTML, CSS, and JavaScript files must be encoded in UTF-8 form with a corresponding byte-order mark (BOM) in order to benefit from bytecode caching and to avoid other runtime error conditions.
- **How to fix:** Open the affected file, and select "Save As..." option from the File menu in Visual Studio. Select the drop-down control next to the Save button and select "Save with Encoding"... option. From the Advanced save options dialog, choose the "Unicode (UTF-8 with signature)" option and click the OK button.

8. This is what a SUCCESS result looks like:


Windows App Certification Kit

Validation Results

The validation process has completed.

Overall Score: **PASSED**

Click here to view full report.

Click the link below to submit the app to the Windows Store.
Windows Store

Windows App Certification Kit - Test Results

App name:	**580338a5-58f3-4645-8c3b-10b04e853440**
App version:	**1.0.0.0**
App publisher:	**jflam**
OS Version:	**Microsoft Windows 8 Release Preview (6.2.8400.0)**
Report time:	**6/11/2012 10:53:35 AM**

Overall result: PASSED

Crashes and hangs test

PASSED	App launch tests
PASSED	Crashes and hangs

App manifest compliance test

PASSED	App manifest