**Prerequisites**

We assume that you can create a basic Windows Store app using JavaScript as described in [Create your first Windows Store app using JavaScript](http://msdn.microsoft.com/en-us/library/windows/apps/br211385.aspx).

**Contents of a basic app**

When you use Microsoft Visual Studio Express 2012 for Windows 8 to create a new project that uses the **Blank Application** template, it creates an app that contains a handful of files:

* **default.html**

The app's start page. This is the page that shows when your app is launched. This file contains references to default.js, the Windows Library for JavaScript files and style sheets, and the app's own style sheet, default.css.

* **/js/default.js**

JavaScript code that specifies how the app behaves when it's started. This file contains code for handling your app's lifecycle. The template-generated default.js handles app activation, but a typical app will also handle suspending and resuming events.

* **/css/default.css**

The app's Cascading Style Sheets (CSS) styles. Here you can define your own styles or override existing the ones provided by the Windows Library for JavaScript. The default implementation makes the body element use a grid layout. For more info about using the grid layout, see [Grid alignment](http://msdn.microsoft.com/en-us/library/windows/apps/hh673533.aspx).

* **/References/Windows Library for JavaScript SDK**

The Windows Library for JavaScript: a set of styles, controls, and utilities for creating apps.

* **package.appmanifest**

The app manifest. This file lists the contents of your app and describes its capabilities, such as whether the app can access the user's webcam. It also specifies which page to use as the app's start page. When you use Visual Studio Express 2012 for Windows 8 to add more files to your app, it automatically updates your app manifest.

* **/images**

The template also includes several image files, like splashscreen.png for the splash screen image, and storelogo.png, which is used for the Windows Store.

**The default.html and default.js files**

Regardless of which Visual Studio Express 2012 for Windows 8 template you use to create your Windows Store app using JavaScript, it contains a default.html and default.js file.

Here's the markup for default.html that gets generated for you when you create a new project using the **Blank Application** template.

HTML

Copy

<!DOCTYPE html>

<html>

<head>

 <meta charset="utf-8">

 <title>BasicAppExample</title>

 <!-- WinJS references -->

 <link href="//Microsoft.WinJS.1.0.RC/css/ui-dark.css" rel="stylesheet">

 <script src="//Microsoft.WinJS.1.0.RC/js/base.js"></script>

 <script src="//Microsoft.WinJS.1.0.RC/js/ui.js"></script>

 <!-- BasicAppExample references -->

 <link href="/css/default.css" rel="stylesheet" />

 <script src="/js/default.js"></script>

</head>

<body>

 <p>Content goes here</p>

</body>

</html>

Here's the code for the default.js file that gets generated for you automatically when you create a new project.

JavaScript

Copy

(function () {

 "use strict";

 WinJS.Binding.optimizeBindingReferences = true;

 var app = WinJS.Application;

 var activation = Windows.ApplicationModel.Activation;

 WinJS.strictProcessing();

 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());

 }

 };

 app.oncheckpoint = function (args) {

 // TODO: This application is about to be suspended. Save any state

 // that needs to persist across suspensions here. You might use the

 // WinJS.Application.sessionState object, which is automatically

 // saved and restored across suspension. If you need to complete an

 // asynchronous operation before your application is suspended, call

 // args.setPromise().

 };

 app.start();

})();

**The app lifecycle**

Most of the code in the default.js file handles the app lifecycle. The app lifecycle begins when the app starts, and ends when the app is closed. The project templates include a generic pattern for managing the app lifecycle.

The code provided by the template checks whether the app was started from the Start screen and calls [**WinJS.UI.processAll**](http://msdn.microsoft.com/en-us/library/windows/apps/hh440975.aspx). The **WinJS.UI.processAll** loads all the Windows Library for JavaScript controls that you declared in your HTML file. If you have code that needs to run when the app sets up its initial state, include it in the handler for the [**activated**](http://msdn.microsoft.com/en-us/library/windows/apps/br212679.aspx) event.

**Note**  Performing a lot of work during activation will make your app seem unresponsive. If possible, defer expensive operations until after your app is loaded.

The template doesn't include code to handle suspending, resuming, or terminating the app. For more info about handling these states, see [Application lifecycle](http://msdn.microsoft.com/en-us/library/windows/apps/hh464925.aspx).

**Anonymous functions and strict mode**

If you're used to programming for websites, you might find a few aspects of the code unusual. For one, the entire file is wrapped in an anonymous function:

JavaScript

Copy

(function () {

 ...

})();

Why is that? JavaScript has two scopes: global and local. If you declare a variable outside of any function definition, it is a global variable, and its value is accessible and modifiable throughout your program. If you declare a variable inside of a function definition, that variable is local. It cannot be accessed by anything outside the function.

By wrapping your code in an anonymous function, you make it private. Wrapping your code in an anonymous function is a good coding practice because it limits the scope of your code and avoids polluting the global namespace. It also makes it easier to avoid naming conflicts or situations where you accidently modify a value that you didn't intend to.

Basically, you can use anonymous functions to create private members in JavaScript, a language that doesn't otherwise support them.

If you're wondering what the extra parenthesis at the end of the function declaration does, it invokes the anonymous function:

JavaScript

Copy

(function () {

 ...

})(); // Makes the function self-invoking.

The default.js file also declares that it runs in strict mode:

JavaScript

Copy

(function () {

 "use strict"; // Declares strict mode.

 ...

})();

Strict mode provides better error-checking for your JavaScript code. When you use **strict** mode, your code is subject to a tighter set of restrictions than JavaScript normally allows. For example, you can't use a variable without declaring it, write to a read-only property, or use a **with** statement. These restrictions help you write better code and reduce the likelihood of introducing errors into your app. For more info about **strict** mode, see [Strict Mode in the JavaScript Language Reference](http://msdn.microsoft.com/en-us/library/windows/apps/br230269.aspx).

**Adding event handlers**

While wrapping your code in an anonymous function provides many advantages, it forces you to change how you write your code. For example, you need to change how you handle events.

Suppose that you add a button to your default.html page. When the user clicks the button, you want to display the x- and y- coordinates of the point clicked, so you add a paragraph element for displaying that info. The next example adds these elements to the default.html file, and it adds a [**div**](http://msdn.microsoft.com/en-us/library/windows/apps/hh453133.aspx) element to contain them. (It sets the [**-ms-grid-row**](http://msdn.microsoft.com/en-us/library/windows/apps/hh466346.aspx) and [**-ms-grid-column**](http://msdn.microsoft.com/en-us/library/windows/apps/hh466324.aspx) of the **div** element, because the [**body**](http://msdn.microsoft.com/en-us/library/windows/apps/hh453011.aspx) element is using the [**-ms-grid**](http://msdn.microsoft.com/en-us/library/windows/apps/hh453434.aspx) **display** style.)

HTML

Copy

<!DOCTYPE html>

<html>

<head>

 <meta charset="utf-8">

 <title>BasicAppExample</title>

 <!-- WinJS references -->

 <link href="//Microsoft.WinJS.1.0.RC/css/ui-dark.css" rel="stylesheet">

 <script src="//Microsoft.WinJS.1.0.RC/js/base.js"></script>

 <script src="//Microsoft.WinJS.1.0.RC/js/ui.js"></script>

 <!-- BasicAppExample references -->

 <link href="/css/default.css" rel="stylesheet" />

 <script src="/js/default.js"></script>

</head>

<body>

 <button id="button1">An HTML button</button>

 <p id="button1Output"></p>

</body>

</html>

Now, define an event handler for the button's [**click**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441312.aspx) event in your default.js file. Most event handlers take a single argument, an [**Event**](http://msdn.microsoft.com/en-us/library/windows/apps/hh465785.aspx) object that contains info about the event. Other events might return other types of event info objects that provide info specific for that event.

The [**click**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441312.aspx) event provides a [**MouseEvent**](http://msdn.microsoft.com/en-us/library/windows/apps/hh465801.aspx) object that contains info about the event, such as which mouse button was pressed and which object fired the event. This example creates a **click** event handler that uses the **MouseEvent** object to obtain the x- and y-coordinates of the point that the user clicked.

(The [**click**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441312.aspx) event also responds to touch and keyboard interaction. The examples in this topic assume that the user is clicking with a mouse. For more info about interacting with touch and different devices, see [Responding to user interaction](http://msdn.microsoft.com/en-us/library/windows/apps/hh700412.aspx).)

JavaScript

Copy

(function () {

 "use strict";

 WinJS.Binding.optimizeBindingReferences = true;

 var app = WinJS.Application;

 var activation = Windows.ApplicationModel.Activation;

 WinJS.strictProcessing();

 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());

 }

 };

 app.oncheckpoint = function (args) {

 // TODO: This application is about to be suspended. Save any state

 // that needs to persist across suspensions here. You might use the

 // WinJS.Application.sessionState object, which is automatically

 // saved and restored across suspension. If you need to complete an

 // asynchronous operation before your application is suspended, call

 // args.setPromise().

 };

 // The click event handler for button1

 function button1Click(mouseEvent) {

 var button1Output = document.getElementById("button1Output");

 button1Output.innerText =

 mouseEvent.type

 + ": (" + mouseEvent.clientX + "," + mouseEvent.clientY + ")";

 }

 app.start();

})();

The only thing you need to do now is register the event handler with the button. If this were a website, you'd probably register the handler by setting the button's [**onclick**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441312.aspx) attribute.

HTML

Copy

 <button id="button1" onclick="button1Click(event)">An HTML button</button>

It doesn't work, though. When you run the app and click the button, you get this exception:

Unhandled exception at line 21, column 39 in ms-appx://77d9a962-d259-45b1-9f04-3abf289aff57-6mtxa77c2r4nt/default.html

0x800a1391 - JavaScript runtime error: 'button1Click' is undefined

But your default.js file contains a definition for button1Click. Why doesn't it work?

That's because everything inside the anonymous function in default.js is local, or private. Your default.html can't access the button1Click function.

JavaScript

Copy

(function () {

 // default.html can't access anything inside this function.

})();

There are two ways to work around this restriction. You can make button1Click publicly accessible, or you can attach the event handler in code. The recommended approach is to attach the event handler in code, but we show you both ways. (To jump right to the recommended approach, see [Attaching an event handler in code](http://msdn.microsoft.com/en-us/library/windows/apps/hh780660.aspx#attach_in_code).)

**Making a member publicly accessible**

Every Windows Store app using JavaScript includes the Windows Library for JavaScript. One of the useful things that the Windows Library for JavaScript provides is a way to define namespaces and classes. The [**WinJS.Namespace.define**](http://msdn.microsoft.com/en-us/library/windows/apps/br212667.aspx) function creates a public namespace. It takes two parameters: the name of the namespace to create, and an object that contains one or more property/value pairs. Each property is the public name of the member, and each value is a variable, property, or function to expose.

This example creates a namespace named startPage that contains a single member named clickEventHandler. The value of clickEventHandler is the button1Click function you defined earlier.

JavaScript

Copy

 var namespacePublicMembers = { clickEventHandler: button1Click };

 WinJS.Namespace.define("startPage", namespacePublicMembers);

Here's the complete default.js file, so you can see how the namespace declaration fits in with the rest of the file.

JavaScript

Copy

(function () {

 "use strict";

 WinJS.Binding.optimizeBindingReferences = true;

 var app = WinJS.Application;

 var activation = Windows.ApplicationModel.Activation;

 WinJS.strictProcessing();

 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());

 }

 };

 app.oncheckpoint = function (args) {

 // TODO: This application is about to be suspended. Save any state

 // that needs to persist across suspensions here. You might use the

 // WinJS.Application.sessionState object, which is automatically

 // saved and restored across suspension. If you need to complete an

 // asynchronous operation before your application is suspended, call

 // args.setPromise().

 };

 // The click event handler for button1

 function button1Click(mouseEvent) {

 var button1Output = document.getElementById("button1Output");

 button1Output.innerText =

 mouseEvent.type

 + ": (" + mouseEvent.clientX + "," + mouseEvent.clientY + ")";

 }

 var namespacePublicMembers = { clickEventHandler: button1Click };

 WinJS.Namespace.define("startPage", namespacePublicMembers);

 app.start();

})();

Now that the event handler is exposed publicly as startPage.clickEventHandler, you can use it to set the button's [**onclick**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441312.aspx) attribute.

HTML

Copy

 <button id="button1" onclick="startPage.clickEventHandler(event)">An HTML button</button>

Now when you run the app, the event handler works properly.

That's one way to connect an event handler. The other way is to attach the event handler in code. Let's take a look.

**Attaching an event handler in code**

Another way (the recommended way) to attach an event handler is to use JavaScript to retrieve the control, then use the [**addEventListener**](http://msdn.microsoft.com/en-us/library/windows/apps/hh441145.aspx) method to register the event. The question is, when should you retrieve the control? You could just add it anywhere to your JavaScript code, but then there's a chance it might get called before the control exists.

The answer for most HTML files is to provide a [**then**](http://msdn.microsoft.com/en-us/library/windows/apps/br229728.aspx) or [**done**](http://msdn.microsoft.com/en-us/library/windows/apps/hh701079.aspx) function for the [**Promise**](http://msdn.microsoft.com/en-us/library/windows/apps/br211867.aspx) returned by the [**WinJS.UI.processAll**](http://msdn.microsoft.com/en-us/library/windows/apps/hh440975.aspx) method. (For [Page controls](http://msdn.microsoft.com/en-us/library/windows/apps/hh770100.aspx), you can use the [**ready**](http://msdn.microsoft.com/en-us/library/windows/apps/hh770590.aspx) function instead.)

What's a [**Promise**](http://msdn.microsoft.com/en-us/library/windows/apps/br211867.aspx)? To provide responsive user experience, many Windows Library for JavaScript and Windows Runtime functions execute asynchronously. That way your app can continue to respond to user interactions while performing work in the background. Instead of directly returning a value, an asynchronous function returns a **Promise** for a value. For more info about asynchronous programming, see [Asynchronous programming in JavaScript](http://msdn.microsoft.com/en-us/library/windows/apps/hh700330.aspx).

JavaScript

Copy

 args.setPromise(WinJS.UI.processAll().done(function () {

 var button1 = document.getElementById("button1");

 button1.addEventListener("click", button1Click, false);

 })

 );

Here's the complete code for default.html and default.js:

HTML

Copy

<!DOCTYPE html>

<html>

<head>

 <meta charset="utf-8">

 <title>BasicAppExample</title>

 <!-- WinJS references -->

 <link href="//Microsoft.WinJS.1.0.RC/css/ui-dark.css" rel="stylesheet">

 <script src="//Microsoft.WinJS.1.0.RC/js/base.js"></script>

 <script src="//Microsoft.WinJS.1.0.RC/js/ui.js"></script>

 <!-- BasicAppExample references -->

 <link href="/css/default.css" rel="stylesheet" />

 <script src="/js/default.js"></script>

</head>

<body>

 <button id="button1">An HTML button</button>

 <p id="button1Output"></p>

</body>

</html>

JavaScript

Copy

(function () {

 "use strict";

 WinJS.Binding.optimizeBindingReferences = true;

 var app = WinJS.Application;

 var activation = Windows.ApplicationModel.Activation;

 WinJS.strictProcessing();

 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().done(function () {

 var button1 = document.getElementById("button1");

 button1.addEventListener("click", button1Click, false);

 })

 );

 }

 };

 app.oncheckpoint = function (args) {

 // TODO: This application is about to be suspended. Save any state

 // that needs to persist across suspensions here. You might use the

 // WinJS.Application.sessionState object, which is automatically

 // saved and restored across suspension. If you need to complete an

 // asynchronous operation before your application is suspended, call

 // args.setPromise().

 };

 // The click event handler for button1

 function button1Click(mouseEvent) {

 var button1Output = document.getElementById("button1Output");

 button1Output.innerText =

 mouseEvent.type

 + ": (" + mouseEvent.clientX + "," + mouseEvent.clientY + ")";

 }

 var namespacePublicMembers = { clickEventHandler: button1Click };

 WinJS.Namespace.define("startPage", namespacePublicMembers);

 app.start();

})();

**Adding pages and navigating between them**

The **Blank Application** template works well for a very simple app, but when you write a more complex app, you'll probably want to divide your content into multiple files.

A traditional website might have a series of pages that you navigate between using hyperlinks. Each page has its own set of JavaScript functions and data, a new set of HTML to display, style info, and so on. This navigation model is known as *multi-page navigation*.

Unlike a traditional website, a Windows Store app using JavaScript works best when it uses the *single-page navigation* model. In this model, you use a single page for our app and load additional data into that page as needed.

That means that your app never navigates away from its default.html page. Always make the default.html and default.js files your app's start up page. They define the outmost UI for your app (such as the [**AppBar**](http://msdn.microsoft.com/en-us/library/windows/apps/br229670.aspx)) and handle the application lifecyle.

If you never navigate away from default.html, how do you bring in content from other pages? There are a few different ways.

* You can use [**HtmlControl**](http://msdn.microsoft.com/en-us/library/windows/apps/hh700625.aspx) to display HTML from another page that doesn't contain interactivity (**HtmlControl** doesn't support the loading of JavaScript).
* You can create a [**PageControl**](http://msdn.microsoft.com/en-us/library/windows/apps/jj126158.aspx) and display it on your main page. A **PageControl** is set of HTML, JavaScript, and CSS that you can display inside another HTML page, much like you would another control. For more info about creating [Page](http://msdn.microsoft.com/en-us/library/windows/apps/hh770100.aspx) controls, see Adding Page controls.
* You can use an [**iframe**](http://msdn.microsoft.com/en-us/library/windows/apps/hh465955.aspx) to display content from another page. This is the least efficient way to incorporate content form another page, but it's your only option if you want to display an external web page.
* You can also use DOM methods to incorporate content from other pages.