Windows 8

Windows Store Applications for JavaScript developers

Mark Smith | @marksm | julmar.com

julmar.com
SOFTWARE DEVELOPMENT, ARCHITECTURE & TRAINING

DEVELOPMENTOR

Introduction

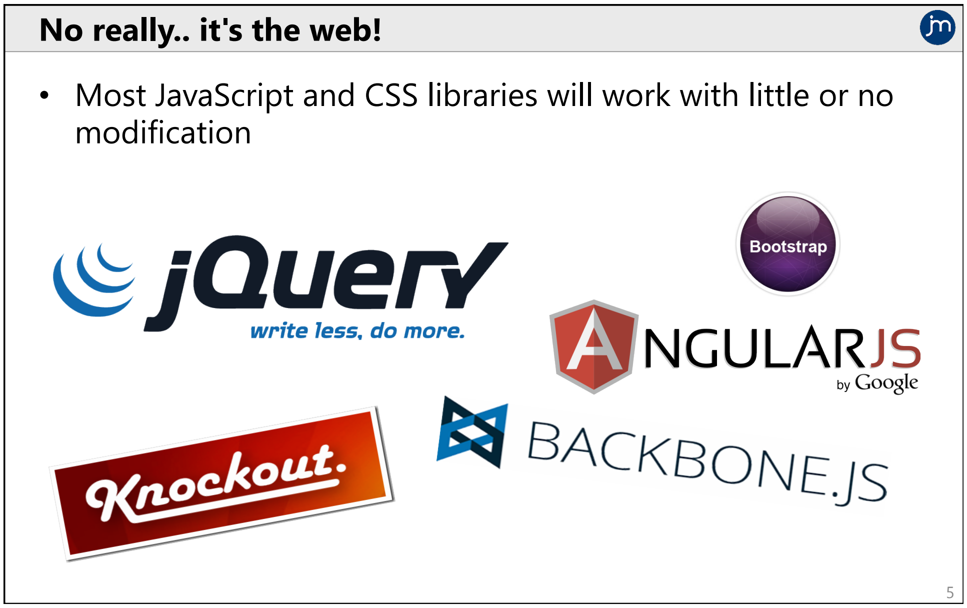
- My name is Mark Smith
- Background in Architecture, UI and Mobile apps
  - but once upon a time I worked with C, C++, MFC and COM
- Work with DevelopMentor as an author/trainer
- Run my own development & consulting company
- Microsoft MVP – Client Dev

Agenda

- How is this model the same?
- How is Windows/HTML programming different?
- Packaging
- What is WinJS?
- Navigation
- Application lifetime and session state
- Async execution and promises
- Binding and templates
- When HTML/JS isn't enough

It's just the web!

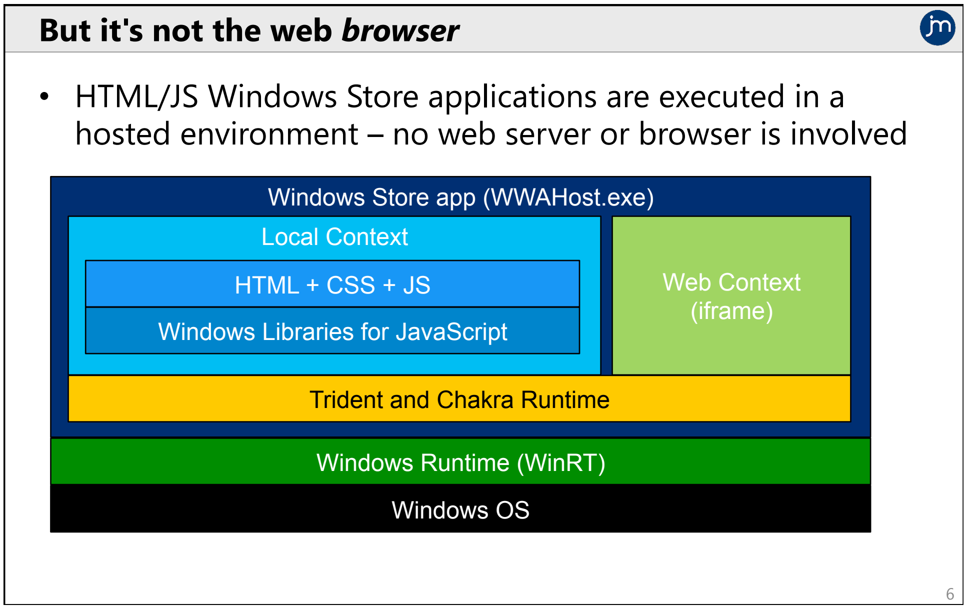
- Microsoft wants web developers to be able to build Windows Store apps using the skills they know and love



No really.. it's the web!

- Most JavaScript and CSS libraries will work with little or no modification

jQuery
write less, do more.
Bootstrap
ANGULARJS
by Google
Knockout.
BACKBONE.JS



But it's not the web *browser*

- HTML/JS Windows Store applications are executed in a hosted environment – no web server or browser is involved

Windows Store app (WWAHost.exe)

Local Context

HTML + CSS + JS

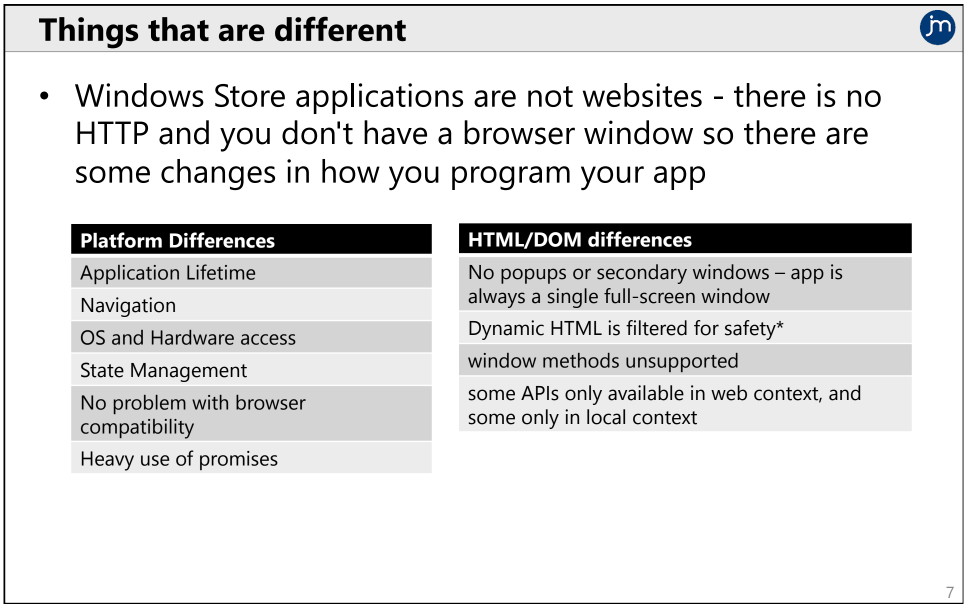
Windows Libraries for JavaScript

Web Context (iframe)

Trident and Chakra Runtime

Windows Runtime (WinRT)

Windows OS

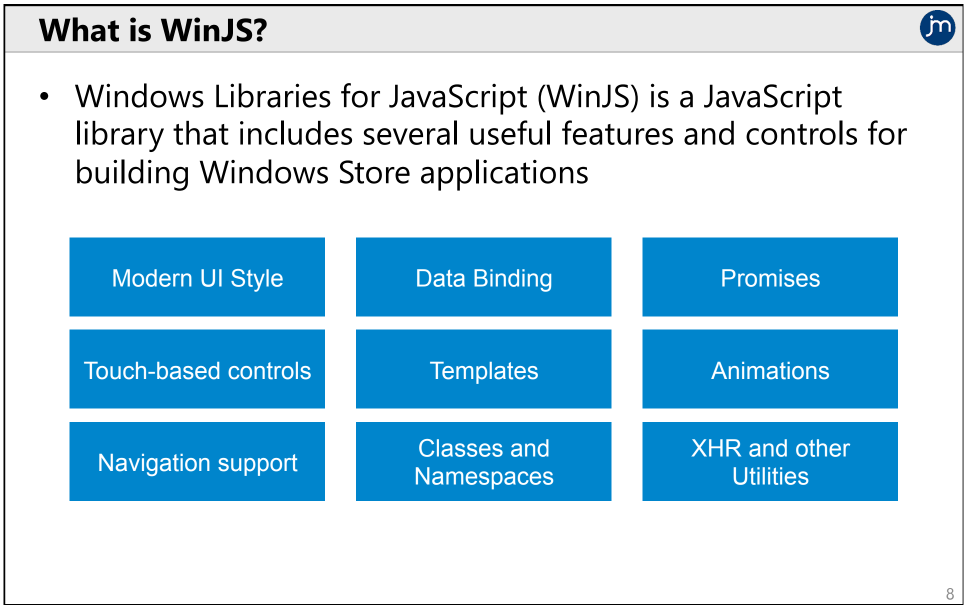


Things that are different

- Windows Store applications are not websites - there is no HTTP and you don't have a browser window so there are some changes in how you program your app

Platform Differences
Application Lifetime
Navigation
OS and Hardware access
State Management
No problem with browser compatibility
Heavy use of promises

HTML/DOM differences
No popups or secondary windows – app is always a single full-screen window
Dynamic HTML is filtered for safety*
window methods unsupported
some APIs only available in web context, and some only in local context



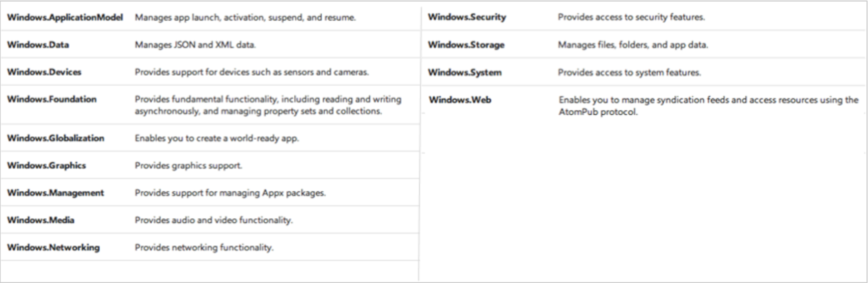
What is WinJS?

- Windows Libraries for JavaScript (WinJS) is a JavaScript library that includes several useful features and controls for building Windows Store applications

Modern UI Style	Data Binding	Promises
Touch-based controls	Templates	Animations
Navigation support	Classes and Namespaces	XHR and other Utilities

WinRT classes

- WinRT includes a library of classes you can use
  - provides access to system resources such as files, network, etc.
  - organized in namespaces, similar to .NET

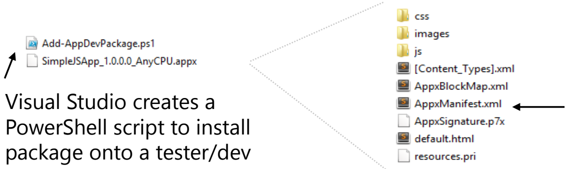


Windows.ApplicationModel	Manages app launch, activation, suspend, and resume.
Windows.Data	Manages JSON and XML data.
Windows.Devices	Provides support for devices such as sensors and cameras.
Windows.Foundation	Provides fundamental functionality, including reading and writing asynchronously, and managing property sets and collections.
Windows.Globalization	Enables you to create a world-ready app.
Windows.Graphics	Provides graphics support.
Windows.Management	Provides support for managing Appx packages.
Windows.Media	Provides audio and video functionality.
Windows.Networking	Provides networking functionality.
Windows.Security	Provides access to security features.
Windows.Storage	Manages files, folders, and app data.
Windows.System	Provides access to system features.
Windows.Web	Enables you to manage syndication feeds and access resources using the AtomPub protocol.

See http://msdn.microsoft.com/en-us/library/windows/apps/br211377.aspx for the full list

App Packaging and Deployment

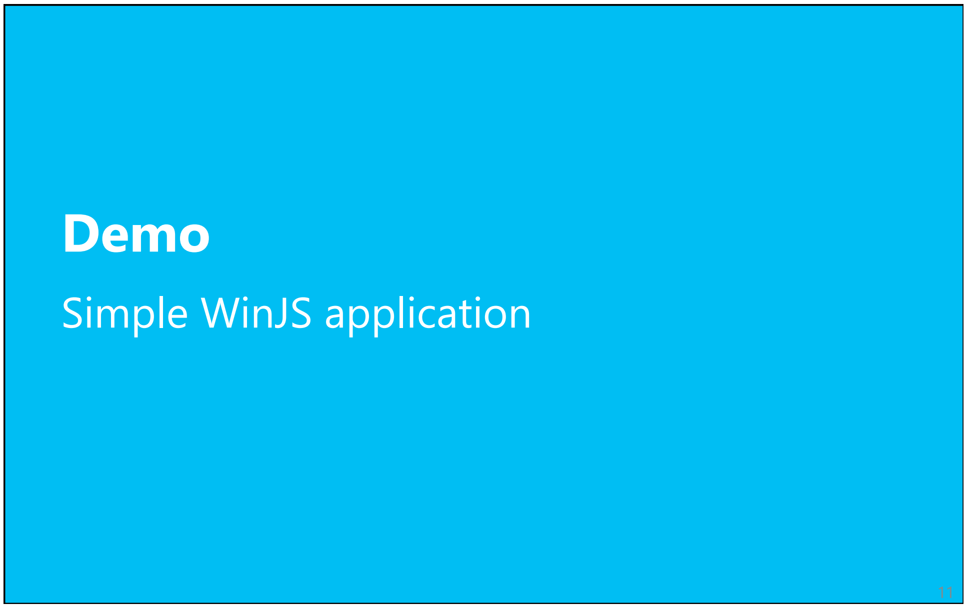
- Applications are packaged by Visual Studio into a zip (.appx) and installed locally onto the user's machine by the Windows Store



Visual Studio creates a PowerShell script to install package onto a tester/dev machine independent of the Windows Store

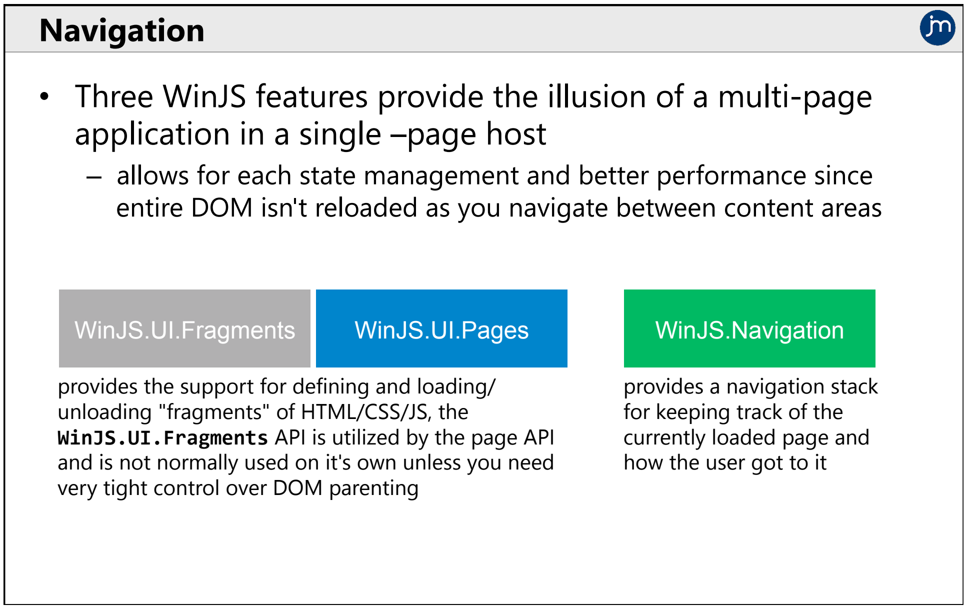
includes a manifest which includes app details and requirements necessary to run the application

all required assets (HTML, CSS, JS, images, etc.) are included in the package – along with some deployment/security information



Demo

Simple WinJS application



Navigation

- Three WinJS features provide the illusion of a multi-page application in a single –page host
  - allows for each state management and better performance since entire DOM isn't reloaded as you navigate between content areas

WinJS.UI.Fragments

WinJS.UI.Pages

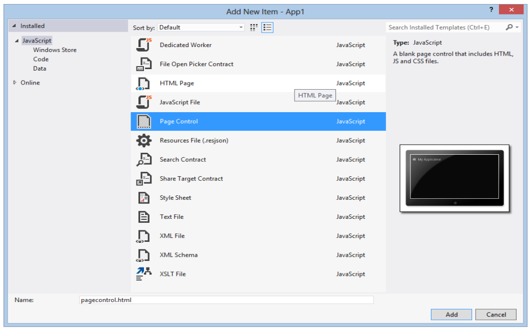
provides the support for defining and loading/ unloading "fragments" of HTML/CSS/JS, the **`WinJS.UI.Fragments`** API is utilized by the page API and is not normally used on it's own unless you need very tight control over DOM parenting

WinJS.Navigation

provides a navigation stack for keeping track of the currently loaded page and how the user got to it

WinJS.UI.Pages

- Page API allows each "page" in the application to be designed and packaged independently as HTML/JS/CSS
  - lower-level fragment API then "merges" these pages into place



Dedicated Worker	JavaScript
File Open Picker Contract	JavaScript
HTML Page	JavaScript
JavaScript File	JavaScript
Page Control	JavaScript
Resources File (.resjson)	JavaScript
Search Contract	JavaScript
Share Target Contract	JavaScript
Style Sheet	JavaScript
Text File	JavaScript
XML File	JavaScript
XML Schema	JavaScript
XSLT File	JavaScript

Type: JavaScript

A blank page control that includes HTML, JS and CSS files.

Name: pagecontrol.html

Page Control item template creates .html, .js and .css file to represent a single page

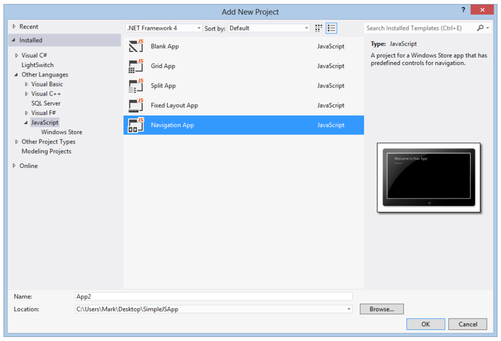
WinJS.Navigation

- WinJS provides browser-like navigation with history stack

```
// Not the real definition – really a namespace type
var WinJS.Navigation = {
   // Properties
   canGoForward : true/false,
   canGoBack: true/false,
   location: "xxx.html",
   state: [user defined],
   history: [history stack],
   // Methods
   forward: function (distance) { },
   back: function (distance) { },
   navigate: function (location, initialState) { },
   // Events
   beforenavigate,
   navigating,
   navigated,
},
```

Connecting the navigation stack with pages

- `WinJS.Navigation` is optional and not connected to the pages – you can create your own implementation, or use the Navigation VS project template which includes a piece of sample code that performs this job



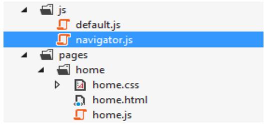
Type: JavaScript

A project for a Windows Store app that has predefined controls for navigation.

Name: App2

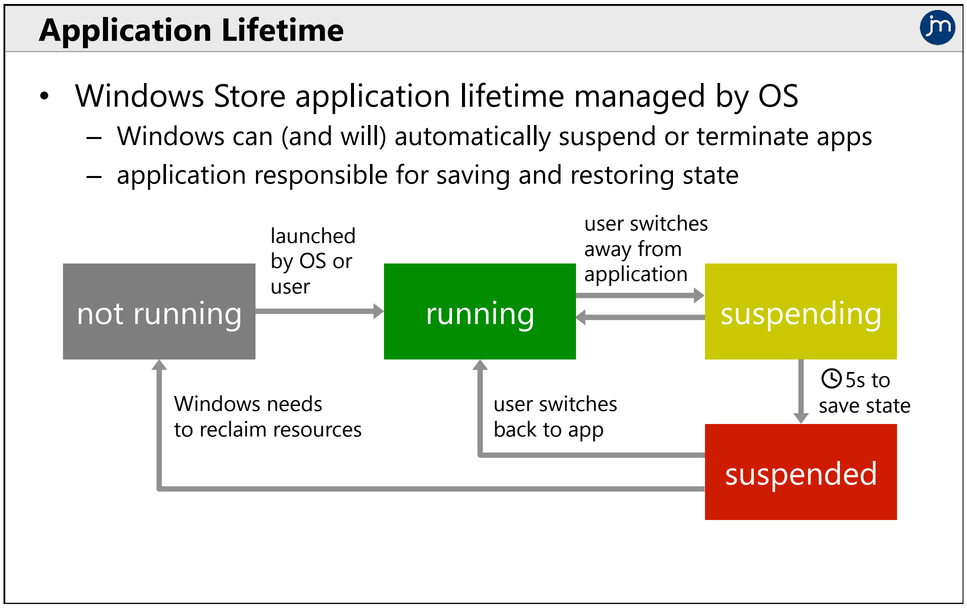
Location: C:\Users\Mark\Desktop\SimpleJSApp

adds navigator.js which provides a page navigator and creates an initial "home" page



Demo

Navigation



Application Lifetime

- Windows Store application lifetime managed by OS
  - Windows can (and will) automatically suspend or terminate apps
  - application responsible for saving and restoring state

not running

launched by OS or user

running

user switches away from application

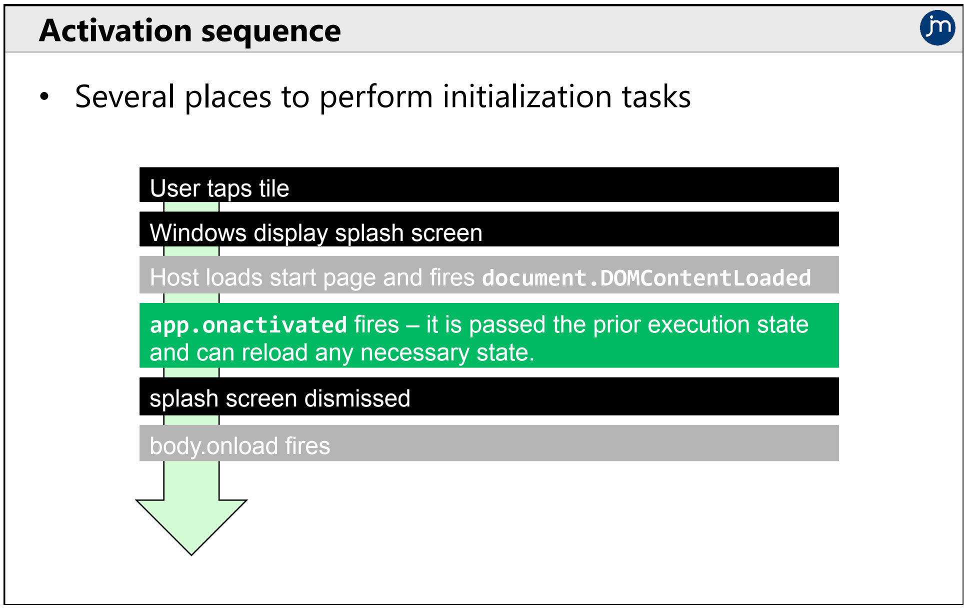
suspending

5s to save state

suspended

user switches back to app

Windows needs to reclaim resources



Activation sequence

- Several places to perform initialization tasks

User taps tile

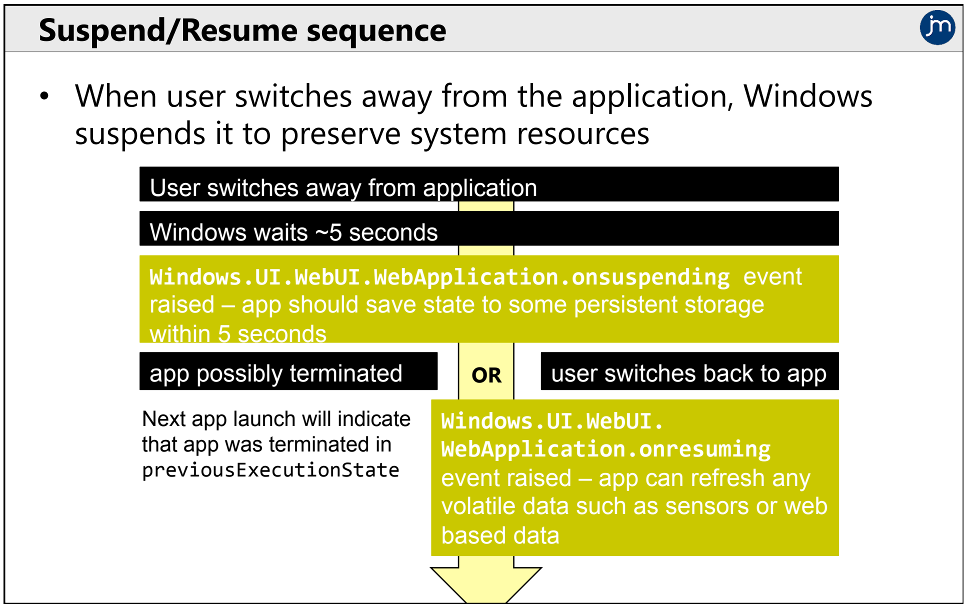
Windows display splash screen

Host loads start page and fires `document.DOMContentLoaded`

`app.onactivated` fires – it is passed the prior execution state and can reload any necessary state.

splash screen dismissed

body.onload fires



Suspend/Resume sequence

- When user switches away from the application, Windows suspends it to preserve system resources

User switches away from application

Windows waits ~5 seconds

`Windows.UI.WebUI.WebApplication.onsuspending` event raised – app should save state to some persistent storage within 5 seconds

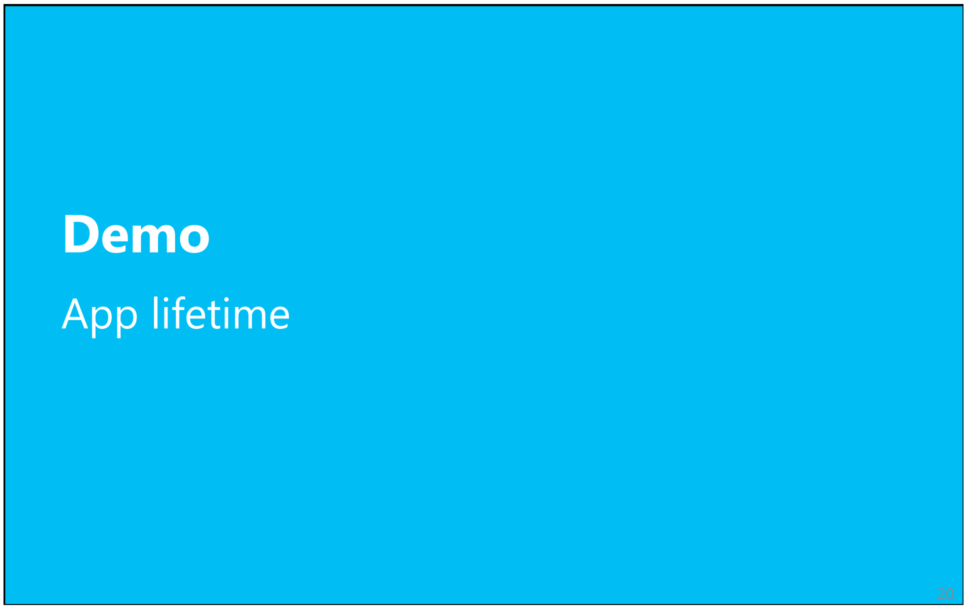
app possibly terminated

OR

user switches back to app

Next app launch will indicate that app was terminated in `previousExecutionState`

`Windows.UI.WebUI.WebApplication.onresuming` event raised – app can refresh any volatile data such as sensors or web based data

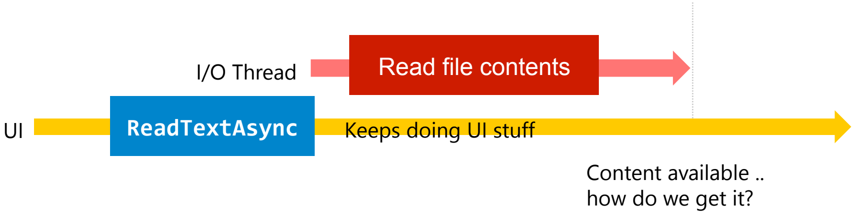


Demo

App lifetime

Async execution

- Performance was key criteria in the design of the WinRT API
  - anything that *could* take more than 50ms is asynchronous only
  - many functions support progress and cancelation as well
  - methods are suffixed with "**`Async`**" by convention
- This makes the coding harder
  - results for many APIs are returned at some point in the future
  - should never block the main thread



I/O Thread

Read file contents

UI

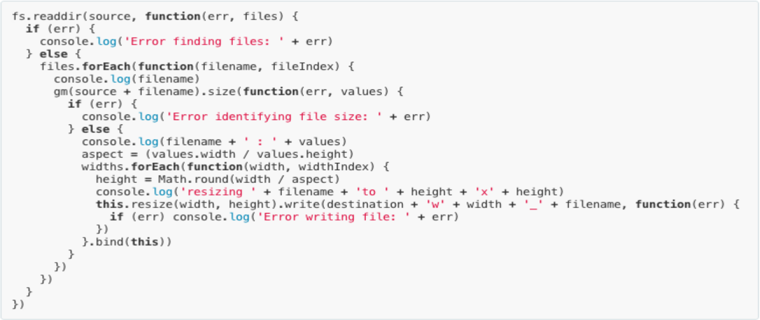
`ReadTextAsync`

Keeps doing UI stuff

Content available ..
how do we get it?

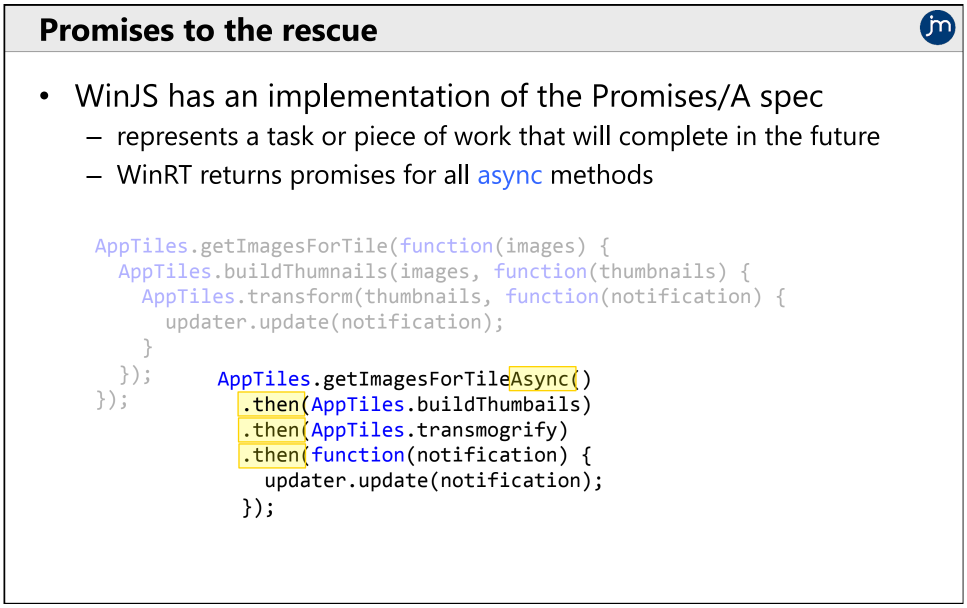
JavaScript Callbacks

- Traditional solution is to use callbacks
  - but functions must be deliberately coded to accept callback
  - can sometimes be hard to manage proper call context (**`this`**)
  - will often alter the flow of execution making the coding harder



```
fs.readdir(source, function(err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function(filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function(err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function(width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(destination + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

example taken from callbackhell.com



Promises to the rescue

- WinJS has an implementation of the Promises/A spec
  - represents a task or piece of work that will complete in the future
  - WinRT returns promises for all async methods

```
AppTiles.getImagesForTile(function(images) {
  AppTiles.buildThumnails(images, function(thumbnails) {
    AppTiles.transform(thumbnails, function(notification) {
      updater.update(notification);
    }
  });
});
```

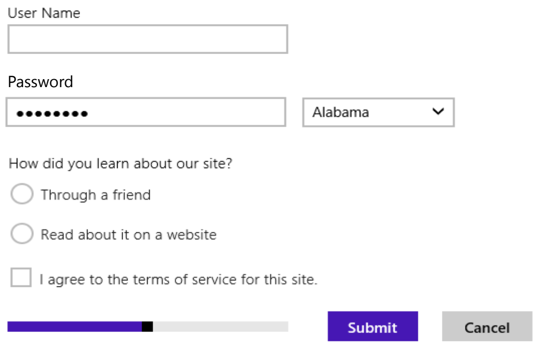
```
AppTiles.getImagesForTileAsync()
  .then(AppTiles.buildThumbails)
  .then(AppTiles.transmogrify)
  .then(function(notification) {
    updater.update(notification);
  });
```

Demo

Promises

Controls

- All standard HTML controls are available for use
  - automatically styled to either Dark or Light theme



User Name

Password

••••••••

Alabama

How did you learn about our site?

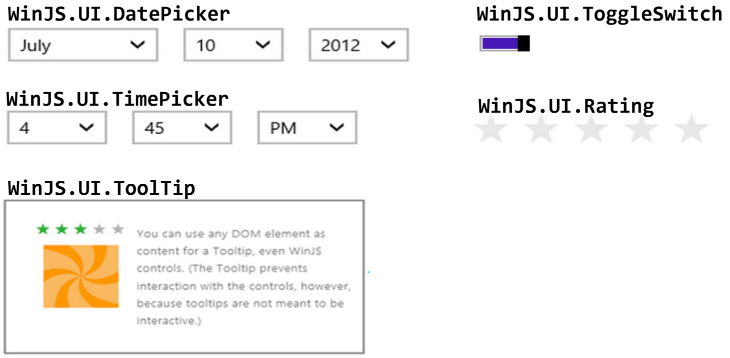
○ Through a friend

○ Read about it on a website

☐ I agree to the terms of service for this site.

WinJS controls

- WinJS adds a set of data controls to help apps match design guidelines



`WinJS.UI.DatePicker`

July | 10 | 2012

`WinJS.UI.ToggleSwitch`

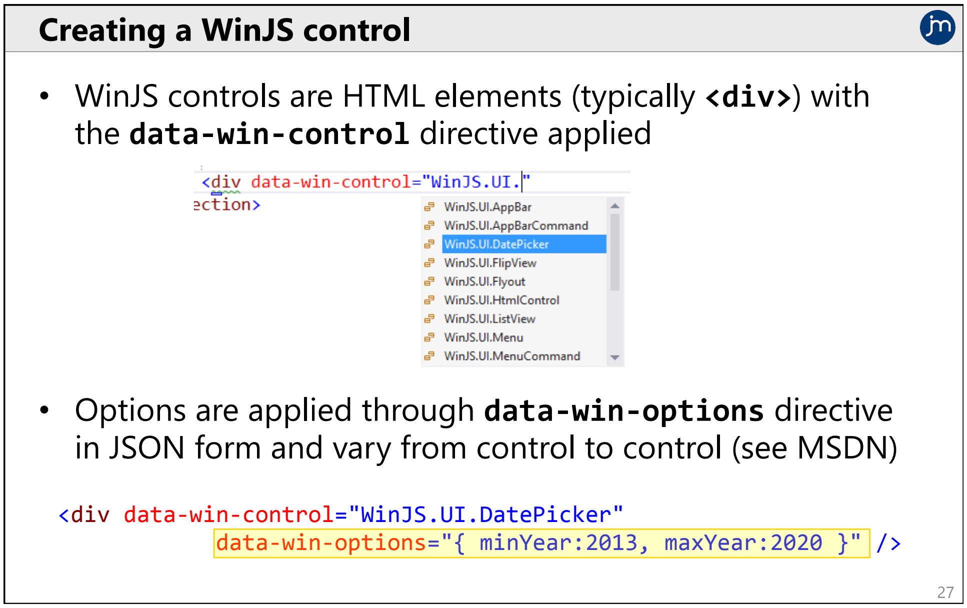
`WinJS.UI.TimePicker`

4 | 45 | PM

`WinJS.UI.Rating`

`WinJS.UI.ToolTip`

You can use any DOM element as content for a Tooltip, even WinJS controls. (The Tooltip prevents interaction with the controls, however, because tooltips are not meant to be interactive.)



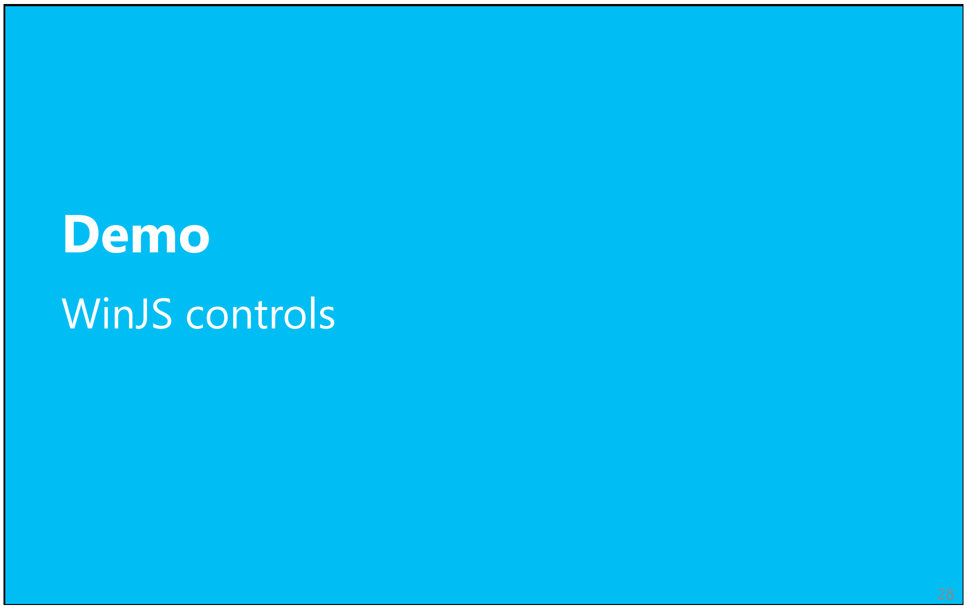
Creating a WinJS control

- WinJS controls are HTML elements (typically **`<div>`**) with the **`data-win-control`** directive applied

```
<div data-win-control="WinJS.UI."
ection>
```

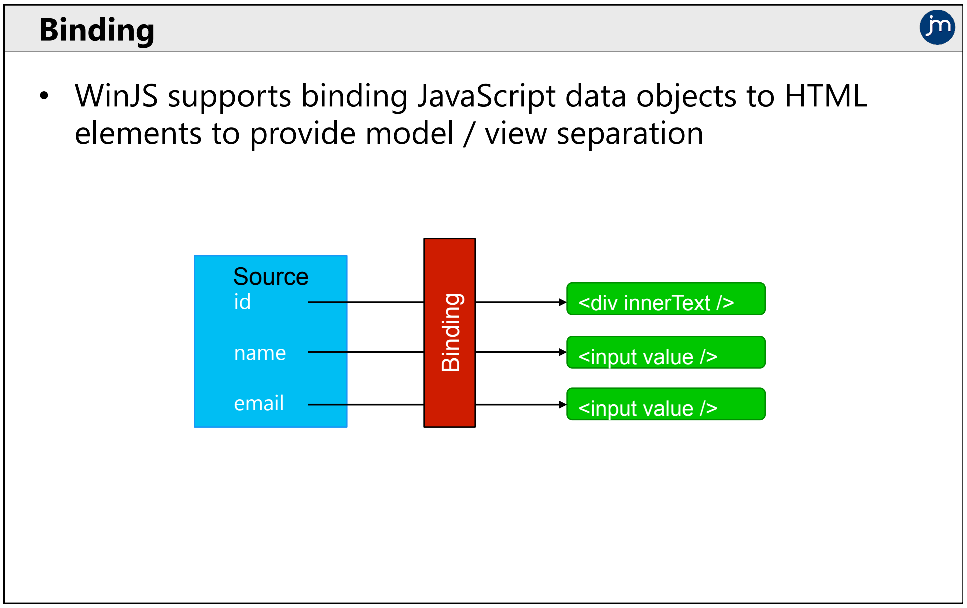
- Options are applied through **`data-win-options`** directive in JSON form and vary from control to control (see MSDN)

```
<div data-win-control="WinJS.UI.DatePicker"
            data-win-options="{ minYear:2013, maxYear:2020 }" />
```



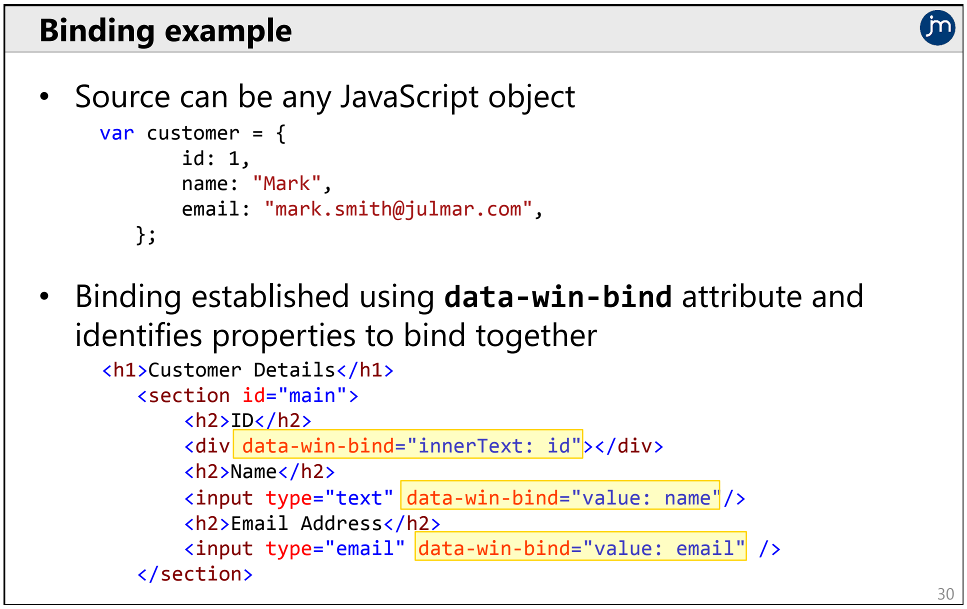
Demo

WinJS controls



Binding

- WinJS supports binding JavaScript data objects to HTML elements to provide model / view separation



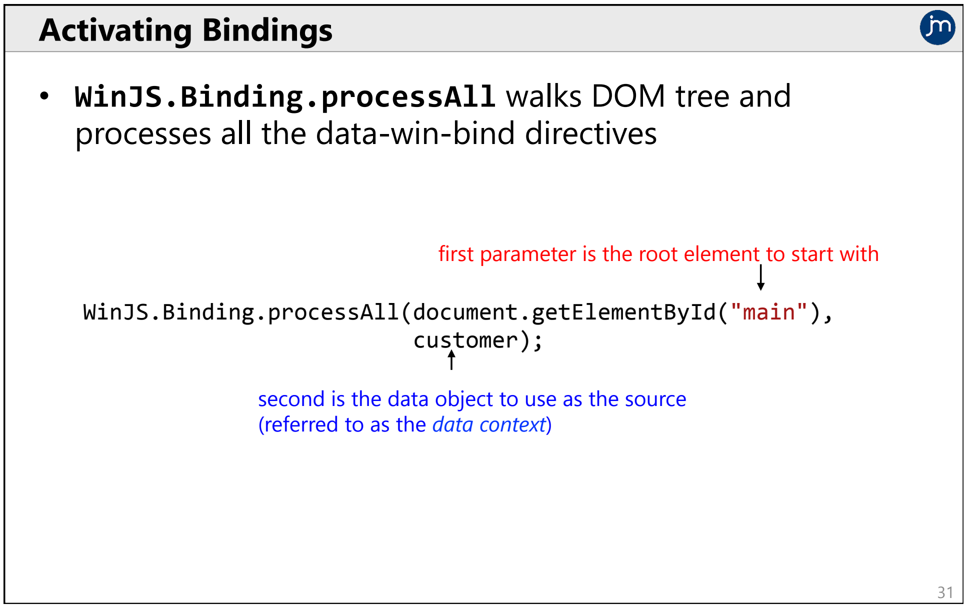
Binding example

- Source can be any JavaScript object

```
var customer = {
       id: 1,
       name: "Mark",
       email: "mark.smith@julmar.com",
   };
```

- Binding established using **`data-win-bind`** attribute and identifies properties to bind together

```
<h1>Customer Details</h1>
   <section id="main">
       <h2>ID</h2>
       <div data-win-bind="innerText: id"></div>
       <h2>Name</h2>
       <input type="text" data-win-bind="value: name"/>
       <h2>Email Address</h2>
       <input type="email" data-win-bind="value: email" />
   </section>
```



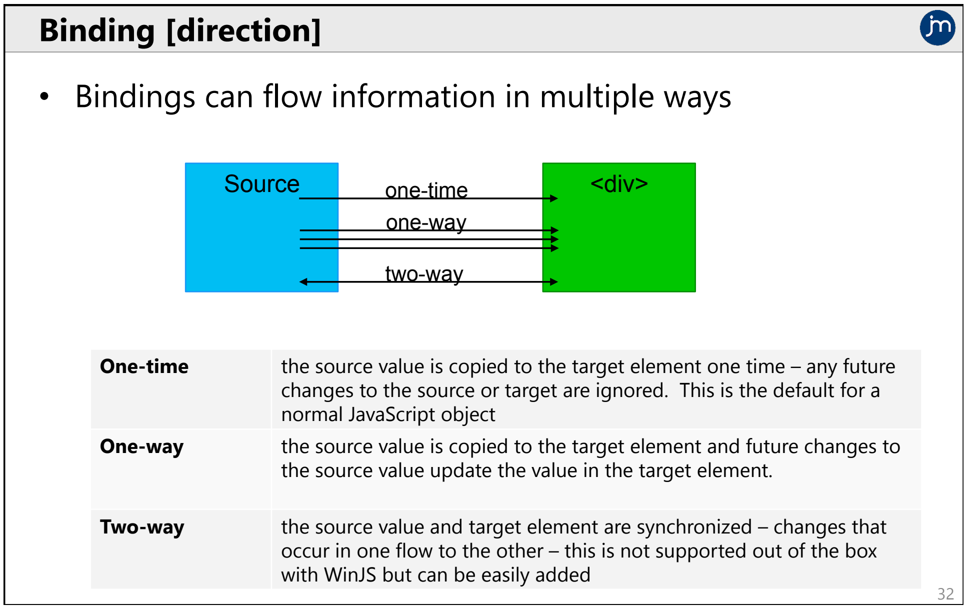
Activating Bindings

- **`WinJS.Binding.processAll`** walks DOM tree and processes all the data-win-bind directives

first parameter is the root element to start with

```
WinJS.Binding.processAll(document.getElementById("main"),
                         customer);
```

second is the data object to use as the source
(referred to as the *data context*)

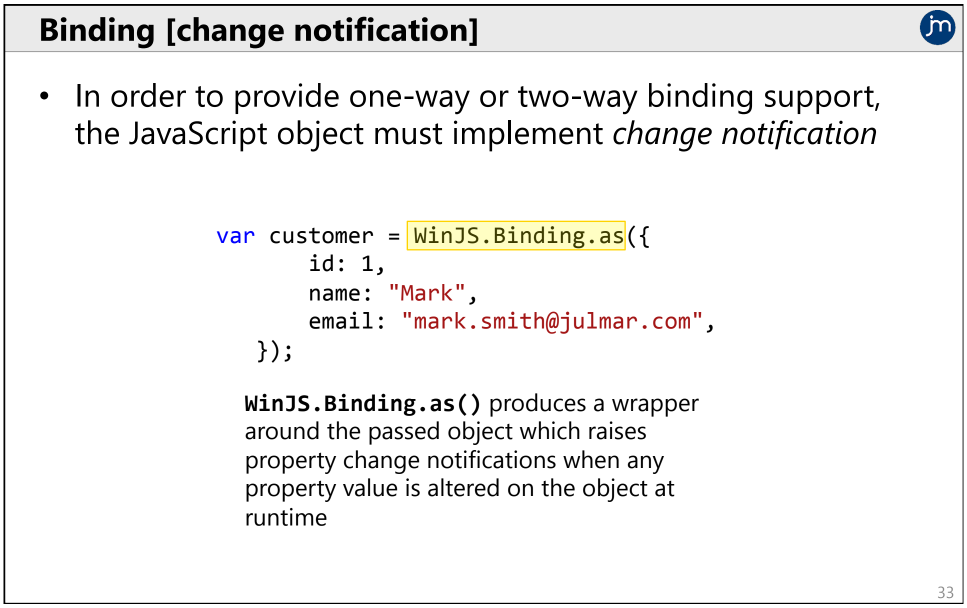


Binding [direction]

- Bindings can flow information in multiple ways

Source
one-time
one-way
two-way
<div>

One-time	the source value is copied to the target element one time – any future changes to the source or target are ignored. This is the default for a normal JavaScript object
One-way	the source value is copied to the target element and future changes to the source value update the value in the target element.
Two-way	the source value and target element are synchronized – changes that occur in one flow to the other – this is not supported out of the box with WinJS but can be easily added

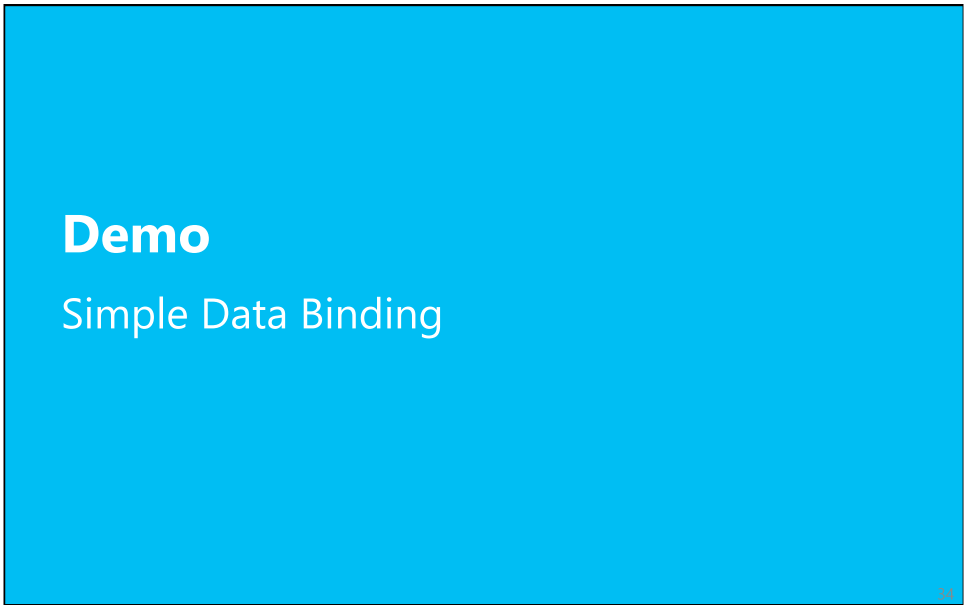


Binding [change notification]

- In order to provide one-way or two-way binding support, the JavaScript object must implement *change notification*

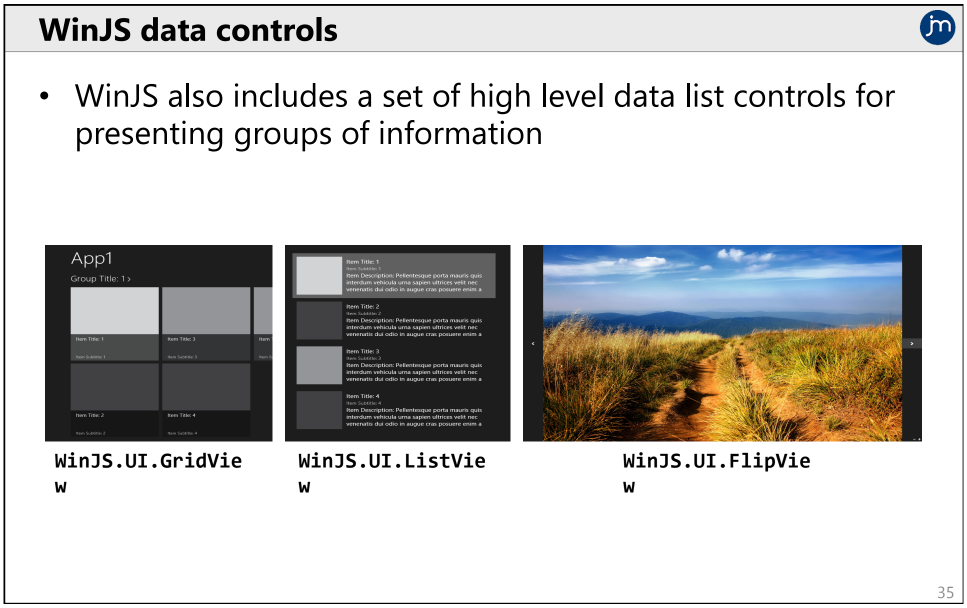
```
var customer = WinJS.Binding.as({
        id: 1,
        name: "Mark",
        email: "mark.smith@julmar.com",
    });
```

`WinJS.Binding.as()` produces a wrapper around the passed object which raises property change notifications when any property value is altered on the object at runtime



Demo

Simple Data Binding



WinJS data controls

- WinJS also includes a set of high level data list controls for presenting groups of information

`WinJS.UI.GridView`

`WinJS.UI.ListView`

`WinJS.UI.FlipView`

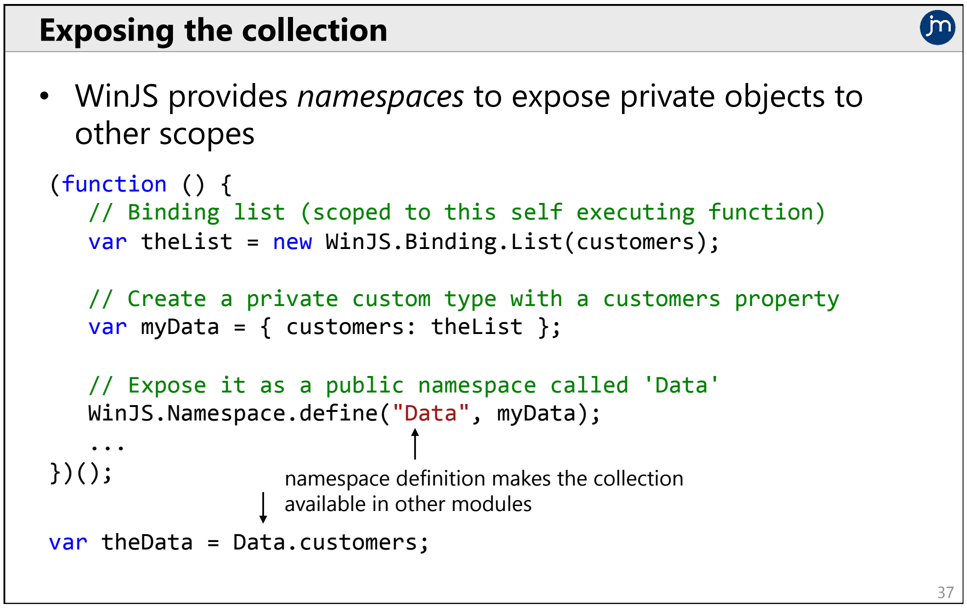
Binding to collections

- Data controls work on lists of data – use **`WinJS.Binding.List`** to hold collections of data that you want to display
    - supports sorting and grouping as well

```
var customers = [
  { id: 1, name: "Mark", email: "mark@julmar.com" },
  { id: 2, name: "Brock", email: "ballen@develop.com" },
];

// Can create from an existing JS array
var theList = new WinJS.Binding.List(customers);

// Can also edit directly
theList.push({ id: 3, name: "Dave", email: "dave@virtualdev.com" });
```



Exposing the collection

- WinJS provides *namespaces* to expose private objects to other scopes

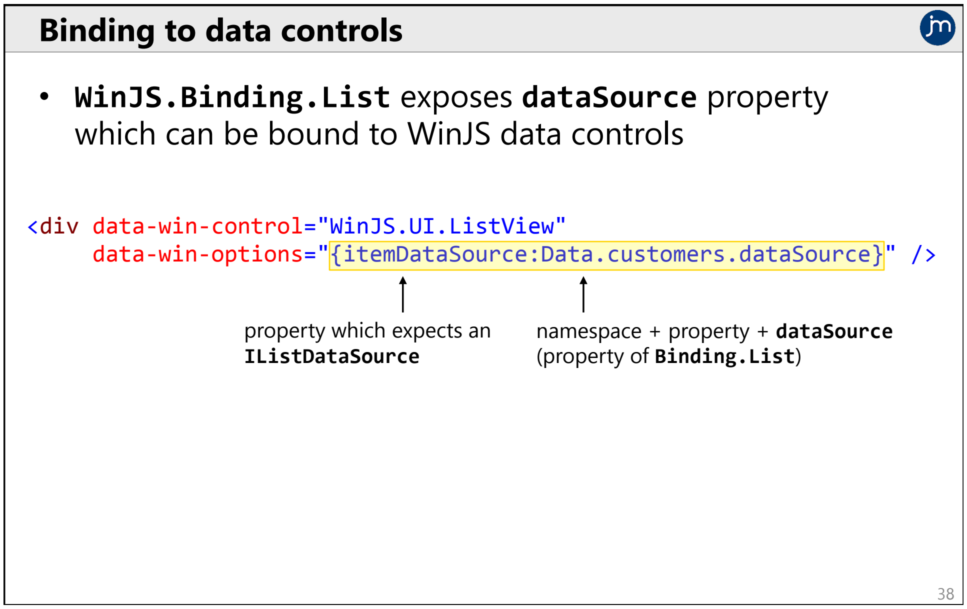
```
(function () {
   // Binding list (scoped to this self executing function)
   var theList = new WinJS.Binding.List(customers);

   // Create a private custom type with a customers property
   var myData = { customers: theList };

   // Expose it as a public namespace called 'Data'
   WinJS.Namespace.define("Data", myData);
   ...
})();
```

namespace definition makes the collection available in other modules

```
var theData = Data.customers;
```



Binding to data controls

- **`WinJS.Binding.List`** exposes **`dataSource`** property which can be bound to WinJS data controls

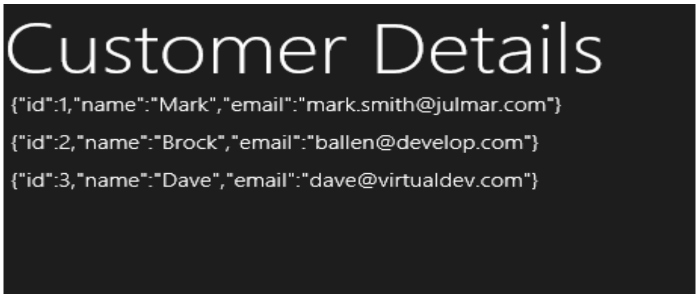
```
<div data-win-control="WinJS.UI.ListView"
     data-win-options="{itemDataSource:Data.customers.dataSource}" />
```

property which expects an **`IListDataSource`**

namespace + property + **`dataSource`** (property of **`Binding.List`**)

Providing a visualization for collections

- Default visualization for a collection is unexciting



Customer Details

```
{"id":1,"name":"Mark","email":"mark.smith@julmar.com"}
{"id":2,"name":"Brock","email":"ballen@develop.com"}
{"id":3,"name":"Dave","email":"dave@virtualdev.com"}
```

To fix this, WinJS provides a templating feature so we can describe what we want a single item to look like and then it will *inflate* the given template for each item found in the collection

Defining a template

- Templates are WinJS controls defined in HTML

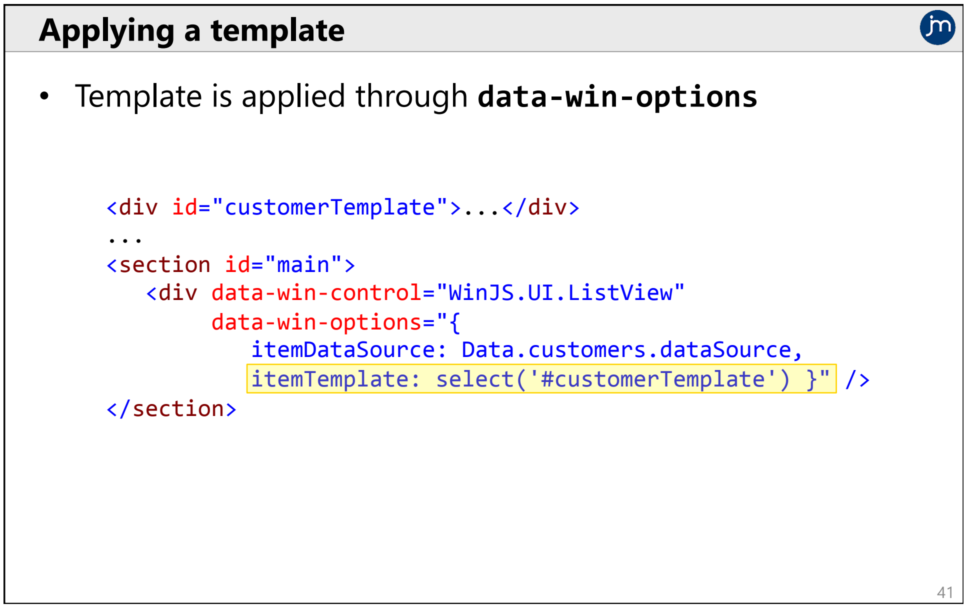
```
<body>

<div id="customerTemplate"
     data-win-control="WinJS.Binding.Template">

   <div data-win-bind="textContent: name"
        style="font-size:large"></div>
   <div data-win-bind="textContent: email"></div>

</div>
...
</body>
```

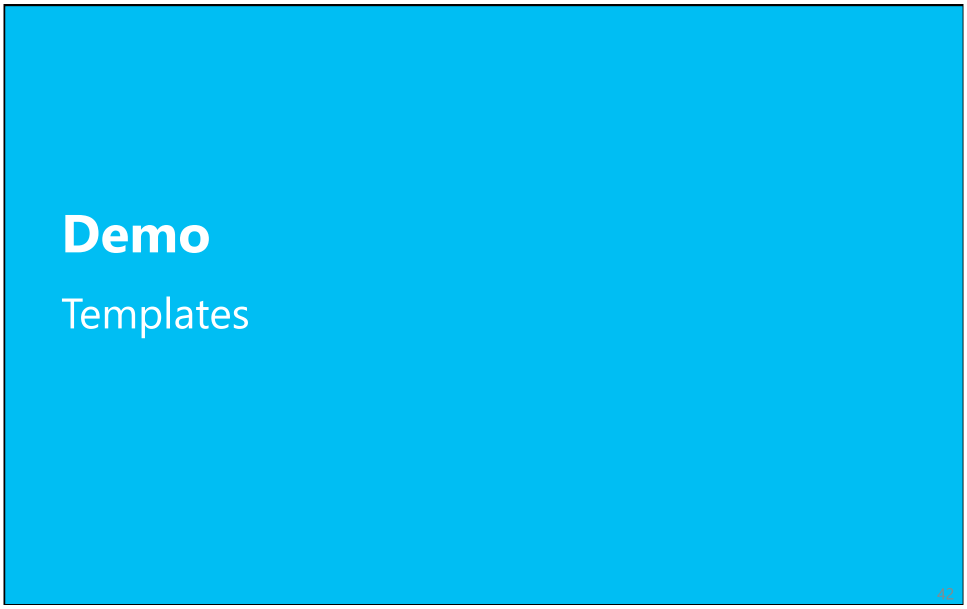
content for the template describes the shape of each element and uses data binding to provide placeholders for where the data goes



Applying a template

- Template is applied through **`data-win-options`**

```
<div id="customerTemplate">...</div>
...
<section id="main">
   <div data-win-control="WinJS.UI.ListView"
        data-win-options="{
           itemDataSource: Data.customers.dataSource,
           itemTemplate: select('#customerTemplate') }" />
</section>
```

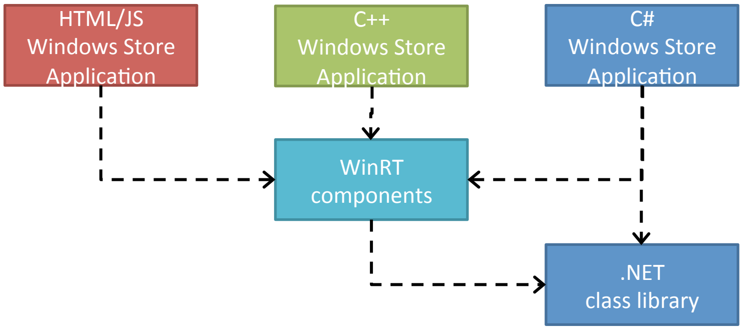


Demo

Templates

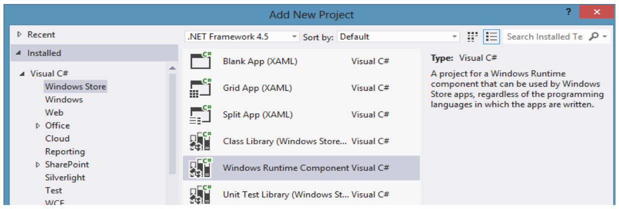
Going beyond JavaScript

- Everything we've done so far has been HTML and JS – but you can get out of the HTML box if necessary by writing a WinRT component library in C++ or C#/VB.NET


HTML/JS Windows Store Application
C++ Windows Store Application
C# Windows Store Application
WinRT components
.NET class library

Creating WinRT components

- WinRT components can be built in C++ or .NET
  - specific rules need to be followed, enforced by compiler

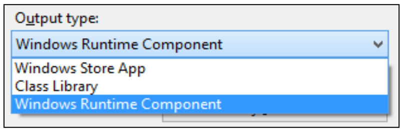

Add New Project
Recent
Installed
Visual C#
Windows Store
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
.NET Framework 4.5
Sort by: Default
Search Installed Te
Blank App (XAML) Visual C#
Grid App (XAML) Visual C#
Split App (XAML) Visual C#
Class Library (Windows Store... Visual C#
Windows Runtime Component Visual C#
Unit Test Library (Windows St... Visual C#

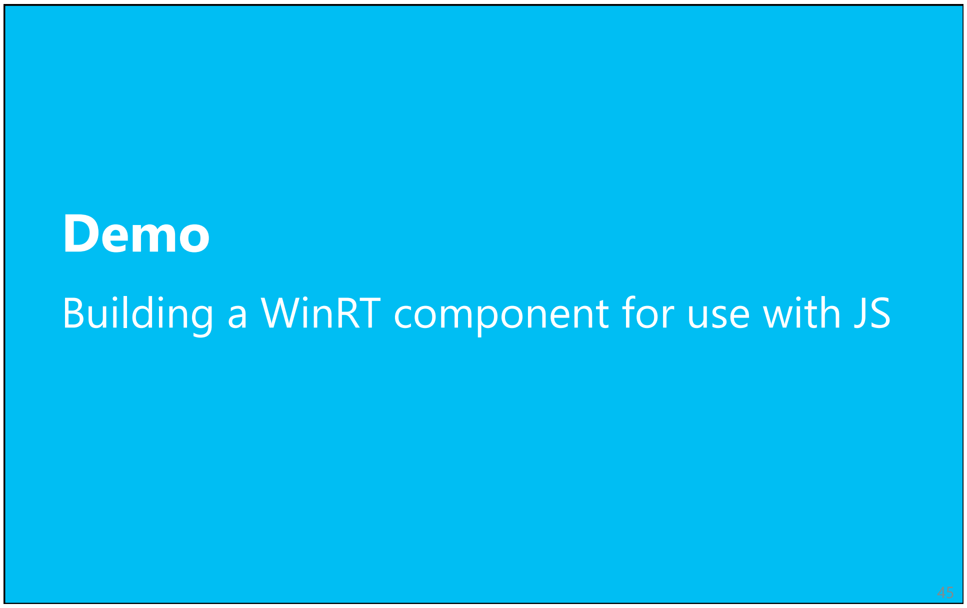
Type: Visual C#

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

can choose "Windows Runtime Component" from project type

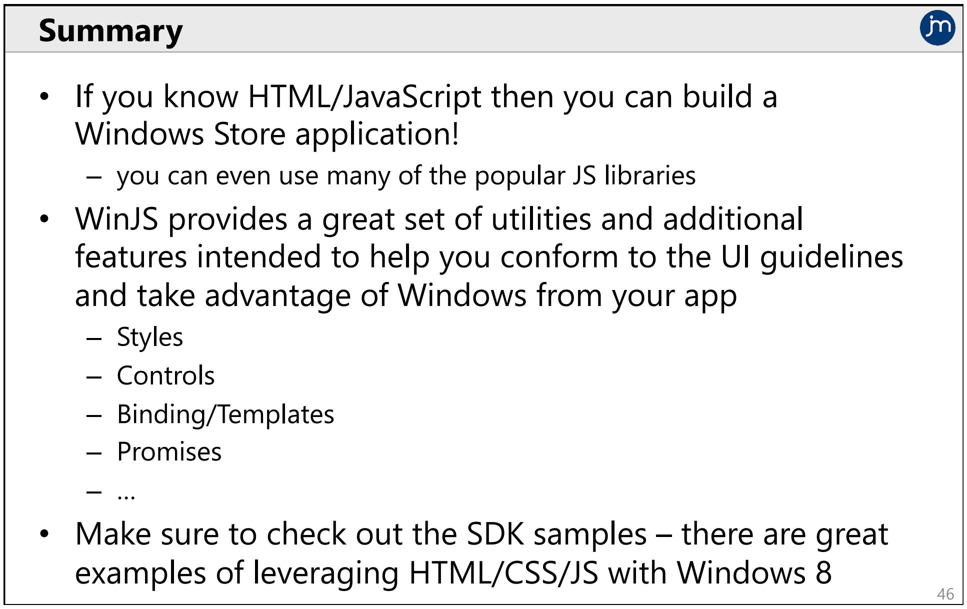
or build normal .NET class library and change output type to Windows Runtime Component


Output type: Windows Runtime Component



Demo

Building a WinRT component for use with JS



Summary

- If you know HTML/JavaScript then you can build a Windows Store application!
  - you can even use many of the popular JS libraries
- WinJS provides a great set of utilities and additional features intended to help you conform to the UI guidelines and take advantage of Windows from your app
  - Styles
  - Controls
  - Binding/Templates
  - Promises
  - …
- Make sure to check out the SDK samples – there are great examples of leveraging HTML/CSS/JS with Windows 8

Q & A

Slides and Samples will be
available at
www.julmar.com/blog

Mark Smith | @marksm | julmar.com