JavaScript for C# Developers

Module 4: Practical Lessons

Shawn Wildermuth
Wilder Minds
wilderminds.com

pluralsight
hardcore developer training

Agenda

- **Practical Lessons**
  - Strictness
  - Iteration
  - Nature of JavaScript APIs
  - Architecting Code
  - "Compiling" JavaScript

Strictness in JavaScript

- **JavaScript Has a History of Loose Interpretation of Code**
  - i.e. JavaScript allows techniques that can introduce bugs
  - Newer Versions of JavaScript (ECMAScript v5) Allow More Strictness

```
// JavaScript
"use strict";

var x = "hello";  // works in strict mode
y = "good bye";   // doesn't work in strict mode
```

Strictness in JavaScript

- **JavaScript Has a History of Loose Interpretation of Code**
  - i.e. JavaScript allows techniques that can introduce bugs
  - Newer Versions of JavaScript (ECMAScript v5) Allow More Strictness

```
// JavaScript
function () {
  "use strict";

  var x = "hello";  // works in strict mode
  y = "good bye";   // doesn't work in strict mode
}
// unstrict code allowed here
```

Strictness in JavaScript

- **What's Disallowed?**
  - Use of Undefined Variables

```
// JavaScript
y = "good bye";                          // exception
```

Strictness in JavaScript

- **What's Disallowed?**
  - Use of Undefined Variables
  - Duplicate Object Properties

```
// JavaScript
var x = {name: "me", name: "us" }; // exception
```

Strictness in JavaScript

- **What's Disallowed?**
  - Use of Undefined Variables
  - Duplicate Object Properties
  - Writing to Read-Only Properties

```
// JavaScript
var s = "hello";
s.length = 5;                          // exception
```

Strictness in JavaScript

- **What's Disallowed?**
  - Use of Undefined Variables
  - Duplicate Object Properties
  - Writing to Read-Only Properties
  - Modifying arguments Object

```
// JavaScript
function () {
  arguments = [];                // exception
}
```

Iteration

- **C#'s foreach is Not JavaScript's for...in**

```
// C#
var a = new[] { "one", "two", "three" };
foreach (var o in a)
{
  // o is each string
}
```

```
// JavaScript
var a = [ "one", "two", "three" ];
for (var o in a) {
  log(o);                  // 0,1,2 huh?
}
```

Iteration

- **C#'s foreach is Not JavaScript's for...in**

```
// JavaScript
var a = [ "one", "two", "three" ];
for (var o = 0; o < a.length; o++) {
  log(a[o]);                    // "one", "two"...
}
```

```
// JavaScript
var a = [ "one", "two", "three" ];
for (var o in a) {
  log(a[o]);                    // "one", "two"...
}
```

Nature of JavaScript APIs

- **In .NET:**
  - Passing of Objects as Parameters is Commonplace

```
// C#
var svr = new SmtpClient();

// Construct Parameter
var msg = new MailMessage("shawn@foo.com",
                          "shawn@foo.com");
msg.Body = "Hello";
msg.Subject = "Test Msg";

// Call Method
svr.Send(msg);
```

Nature of JavaScript APIs

- **In JavaScript:**
  - Duck Typing Means Construction of Ad-hoc Objects Instead

```
// JavaScript
var svr = new SmtpClient();

// Construct Parameter
var msg = {
  to: "shawn@foo.com",
  from: "shawn@foo.com",
  body: "Hello",
  subject: "Test Msg"
};

// Call Method
svr.send(msg);
```

Nature of JavaScript APIs

- **In JavaScript:**
  - Duck Typing Means Construction of Ad-hoc Objects Instead

```
// JavaScript
var svr = new SmtpClient();

// Call Method
svr.send({
  to: "shawn@foo.com",
  from: "shawn@foo.com",
  body: "Hello",
  subject: "Test Msg"
});
```

Nature of JavaScript APIs

- **In JavaScript:**
  - Commonplace to accept an object and have defaults too

```
// JavaScript
var svr = new SmtpClient();

// Call Method
svr.send({
  to: "shawn@foo.com",
  body: "Hello",
  subject: "Test Msg"
});
// from is implied but can override
```

Nature of JavaScript APIs

- **In JavaScript:**
  - Implementing Default Properties

```
// JavaScript
function SmtpClient() {
}

SmtpClient.prototype.send = function(msg) {
  if (!msg.hasOwnProperty("to")) {
    msg.to = "shawn@foo.com";
  }

  if (!msg.hasOwnProperty("mailServer")) {
    msg.mailServer = "smtp.foo.com";
  }
  var to = msg.to;
};
```

Nature of JavaScript APIs

- **In JavaScript:**
  - Implementing Default Properties

```
// JavaScript
function SmtpClient() {
}

SmtpClient.prototype.send = function(msg) {
  var defaults = {
    to: "shawn@foo.com",
    mailServer: "smtp.foo.com"
  };

  $.extend(defaults, msg); // jQuery

  var to = defaults.to;
};
```

Nature of JavaScript APIs

- **How Does JavaScript Handle Events?**
  - Passing In Callbacks is More Common

```
// JavaScript
svr.send({
  from: "shawn@foo.com",
  body: "Hello",
  subject: "Test Msg",
  complete: function (r) {
    alert("Success: " + r);
  },
  error: function (e) {
    alert("Failed: " + e);
  }
});
```

Architecting Large JavaScript Codebases

- **In C#, Assembly is Package**
  - JavaScript Feels Like File Based
  - Two Real Options
    - Build for Coexistence
    - Require.js

Architecting Large JavaScript Codebases

- **Isolating Scripts with Namespaces**

```
<html>
  ...
  <script source="first.js"></script>
  <script source="second.js"></script>
</html>
```

```
// first.js
(function(ns) {
  ns.Customer = function() {
    this.name = "";
  };
}(window.WilderMinds = window.WilderMinds || {}));
```

```
// second.js
(function(ns) {
  ns.Order = function() { ... };
}(window.WilderMinds = window.WilderMinds || {}));
```

Isolating with Namespaces

Demo

Architecting Large JavaScript Codebases

- **Require.js**
  - http://requirejs.org/
  - Uses the Asynchronous Module Definition (AMD) pattern
  - Dependency Injection for JavaScript
  - Loads Scripts as they are needed instead of all at start

Architecting Large JavaScript Codebases

- **Require.js**
  - Include the Script
  - Add attribute for the startup script

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pluralsight</title>
</head>
<body>
  <script src="scripts/require.js"
          data-main="scripts/lib/main.js">
  </script>
</body>
</html>
```

Architecting Large JavaScript Codebases

- **Require.js**
  - Main Script is executed on startup

```
// main.js
require(["Customer"], // Requires the Customer Module
  function (Customer) {  // Call with required Module(s)

    // Your Initialization Code
    var c = new Customer("A Customer");
    var name = c.name;

  }
);
```

Architecting Large JavaScript Codebases

- **Require.js**
  - Module Defined in similar way with define()

```
// Customer.js
define( [],   // Required Scripts (None)
  function(){ // Gets any required modules here like main
    function Customer (name) {
      this.name = name
    }

    return Customer; // Return the object that Requires
                     // constructor to allow you to call it
  }
);
```

Using Require.js

Demo

"Compiling" JavaScript

- **JavaScript is Interpreted...So What About Compilation?**
  - Compilation in C# Performs Two Important Functions
    - Verifies That Code is Syntactically "Correct"
    - Creates Intermediate Language (IL) Code Packages
  - How to Get These Services for JavaScript?
    - JSLint to Check for Correctness
    - Minification for Packaging

"Compiling" JavaScript

- **How Do You "Compile" Your JavaScript?**
  - ASP.NET MVC 4 Supports Packaging (For Minimizing)
    - Other Solutions include SquishIt, Cassette and Chirpy
  - JSLint available as a Visual Studio Add-in for Automatic running of JSLint
    - http://jslint4vs2010.codeplex.com/

Summary

- **Practical JavaScript**
  - As a C# developer, you need to learn more than the language
  - Learning the patterns of frameworks and architecture are key
  - Structuring your own code with namespaces and/or modules is helpful
  - "Compiling" your code can help you deliver better code too