2011 DotNetNuke™ WORLD
ORLANDO

Developing DotNetNuke 6 Modules

Ian Robinson

Sr. Software Engineer

DotNetNuke Corporation

House Keeping

- Breaks
  - 10:30 & 2:30 (15 Min)
- Lunch
  - 12:00 – 1:00
- Questions

About Me / Resources

Top 10% for jquery asp.net cms dotnetnuke

Top 20% for javascript css

- Twitter @irobinson
- GitHub http://github.com/irobinson
- Professional Profile: http://bit.ly/irobinson
- Email: ian@dnncorp.com

Agenda

- Crash Course: DotNetNuke Module Development
  - *High level tour. Key concerns, strategies and techniques.*
- Deep Dive: ***WebFormsMVP***
  - How and why? Unit testing! Diving into code.
- Introducing: Client Resource Management
  - Key concepts and strategies for working with JS and CSS

Section 1
DotNetNuke Module Development Crash Course

Section 1
DotNetNuke Module Development

- Roles
- Goals
- How DNN Works
- Bare minimum configuration
- Project Structuring
- Development styles
- Key API Methods
- Form patterns & jQuery plugins

The Spectrum of Roles

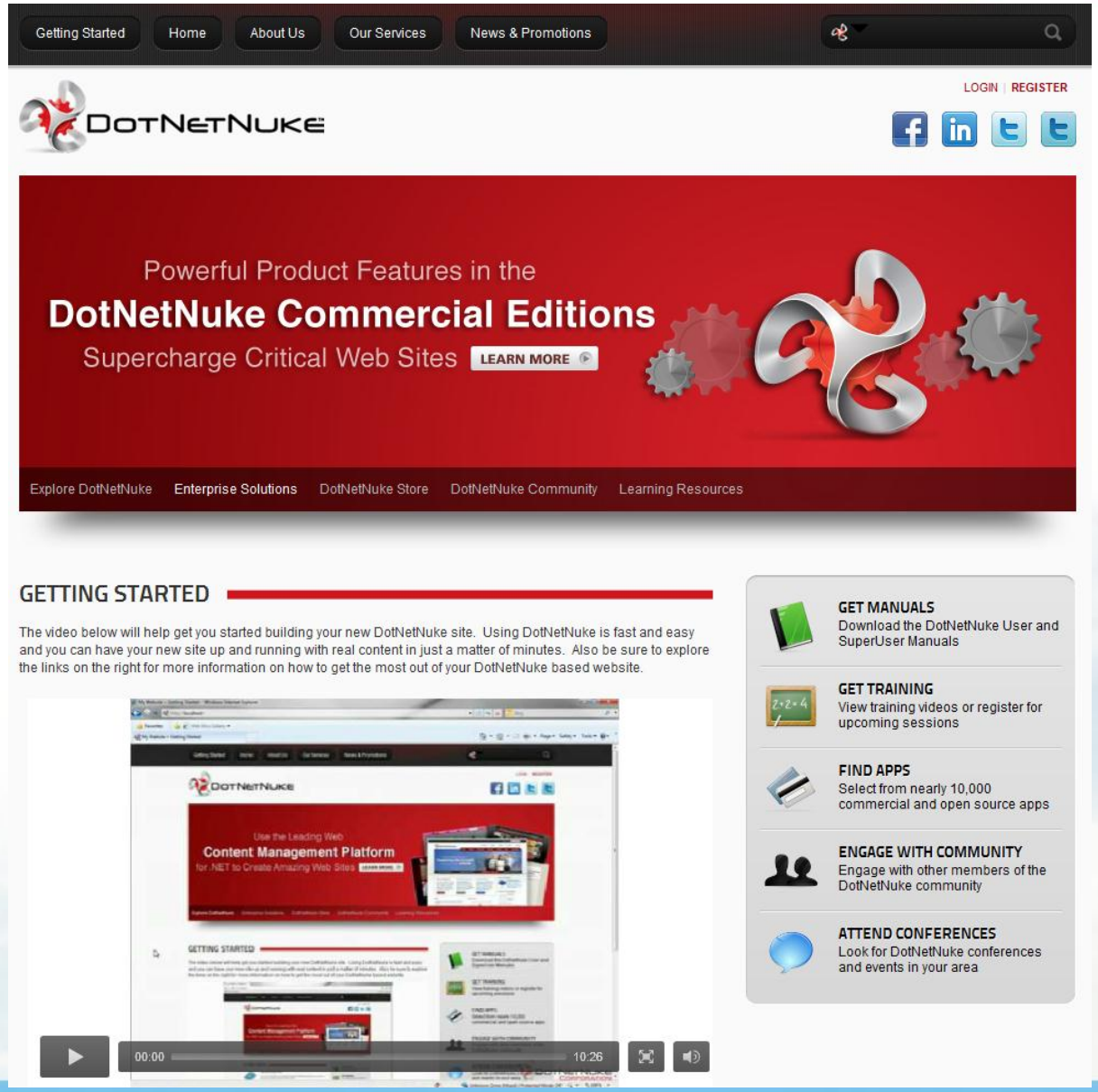
- Savvy Hobbyist
  - Developing in their free time, maybe pro-bono work, etc
- Small biz
  - The “how many hats can I wear at once?” developer.
- Big biz
  - Focused on building modules
- Consultant/integrator
  - Builds solutions for clients
- Commercial Module Developer
  - Sells modules on the interwebs

The Goals of Module Developers

- Develop “stand-alone” code, make it show up in DNN
  - Learn how to plug code in to DNN
  - What are the bare minimum requirements to get going?
- Tight integration with DNN features
  - Leverage the API, get to know how DNN works
  - Learn UI and functional paradigms

So, how does DNN work?

- Default.aspx
  - User controls
    - Within user controls
      - Within user controls



The Bare Minimum Module

- One user control
- Place in a folder within DesktopModules
- Inherit from common base class
  - PortalModuleBase

HelloWorld.ascx

Server Objects & Events | (No Events)

```
<%@ Control Language="C#" Inherits="DotNetNuke.Entities.Modules.PortalModuleBase" %>
<p>Hello world!</p>
```

  - Register module through Host -> Extension

Development Styles & Strategies

- *Environment choice + development strategy*

- Razor Modules
- Inline Script
- ASP.NET Web Site Project (WSP)
- ASP.NET Web Application Project (WAP)
- WebFormsMVP (Using WAP)

Some things to keep in mind…

- Project Size
- Team Size
- Team Experience
- Business Urgency
- Problem Complexity

Razor Modules

- First, what is Razor?
  - Code-focused templating
  - HTML Generation
  - MVC View Engine

Razor:

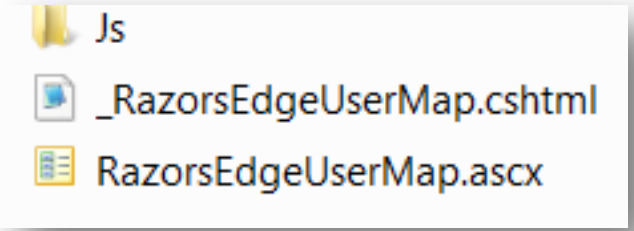
```
<ul id="products">
@foreach (var p in products) {
    <li>@p.Name ($@p.Price)</li>
}
</ul>
```

ASPX:

```
<ul id="products">
<% foreach(var p in products) { %>
    <li><%= p.Name %> (%<%= p.Price%>)</li>
<% } %>
</ul>
```

Razor Modules

- Razor in DNN
  - Razor Host Module
  - DNN context available
  - Packaging/ASCX Buddy System



  - ***Reference: Razor's Edge User Map***

Inline Scripting

- Quick and easy
- Dynamically compiled
- Well known for average .NET developer
- Main benefit: All code in ASCX files

- ***Reference: Customer Feedback Module***

VS Web Site Projects

- Integrates into App_Code and DesktopModules
- Main benefit: dynamically compiled
- More organization + scales better than inline script

- ***Reference: IWeb***

VS Web Application Projects

- WAP is just the project environment
- The strategy is generally larger scale, Object Oriented, n-tier development, etc.
- File system/Project in DesktopModules
  - Build to /bin
  - Include own assemblies
- ***References: DnnSimpleArticle***

WebFormsMVP

- Demo: coming up!
- Project environment is the same as a WAP
- Strategy is leveraging the MVP pattern

Key DNN API

- Current Context
- Localization
- Navigation
- Module Settings

Current Context

- Portal
  - What portal am I in?
- Page
  - What page am I on?
- User
  - Who is the current user (if logged in)?
- Module
  - What settings do I have?

Localization

- Resource Files / Keys
  - Local Resource File
  - Convention based discovery
- Declarative
  - In markup
  - ResourceKey=“”
- Imperative
  - In code
  - LocalizeString(“”)
  - Localization.GetString(“”, this.LocalResourceFile)

Module Registration

- Control Types
  - View
    - Primary type for public facing module views
  - Edit
    - Focus
  - Settings
    - Integrate with “out of the box” module settings
- Manifest Files
  - XML files defines key points
  - Useful for packaging and installation

Navigation

- NavigateURL()
  - Returns the URL of the default view control for your module
- NavigateURL(TabID)
  - Returns the URL of the specific tab
- NavigateURL(TabID, String.Empty, “itemId=4”, “mode=2”)
  - Returns the target tab URL with custom parameters

Tab == Page

Navigation

- EditUrl
  - Helps you to navigate to a specific control
  - Specify by key used during registration
  - Module Isolation

Tab == Page

Module Settings

- All modules have settings
- Custom Module settings
  - Build your own settings control
  - Settings base class
  - Two key methods (Load and Save)
  - Register appropriately
- Module Settings
- Tab Module Settings

General UI Guidelines

- User-facing UI is pretty open ended
- Be a good DNN citizen
  - Semantic well-structured markup (paragraphs, lists, etc)
  - Common sense look and feel
- You're not a designer, so keep it simple
  - Setting different font-sizes? Sure…
  - Setting different type-faces? Not so much…

Admin User Interface

- DNN Admin UI Patterns
- Allow for complete consistency across modules
- Take the guesswork out of implementing the UI
- Messaging! Info, Success, Error, Warning

- ***Reference: Crispy's Contest Module***

Section 2
Deep Dive: WebFormsMVP

Section 2
WebFormsMVP

- Simple and Easy
- WebFormsMVP
- Unit Testing
- Converting DnnSimpleArticle Module
  - Review the starting module structure
  - Convert the edit functionality to the MVP pattern
  - Refactoring existing logic for testability
  - Review the unit tests

Simple and Easy in Software

- Simple
  - One role or task
  - Opposite of complex/intertwined
- Easy
  - Close at hand, known, familiar
- Often we choose easy, at the expense of simplicity
- *Simple is objective, easy is relative*

Examining your project needs

- Quick? Throw-away? Trivial? Low-impact?
  - Sure, go with what you know: easy.
- Project with any kind of longevity or significant importance?
  - Spend the time up front to make things simpler.
- A bug makes it past the type checker, the tests, and then what?

Separation of Concerns

- Each piece has focus
  - One role, task, concept, dimension
  - Lack of interleaving
- Final product has **many** *focused* pieces
- Rather than **few** *overloaded* pieces

WebFormsMVP

- Separation of concerns
- Increased Testability
- “Cleaner Architecture”
- Retains productive characteristics of WebForms

The Players

- View
  - Displays UI, has simple behavior
  - Does not do “heavy lifting”
- Model
  - Holds data, updated by both view and model.
  - Used to transfer data back and forth
- Presenter
  - Isolated from view
  - Performs business logic
  - Ideally, free of hard dependencies

WebFormsMVP Characteristics

- Requires the view to give control to Presenter
- The view and presenter pass around a model
- Interfaces are used to break hard dependencies
  - This supports unit testing

WebFormsMVP in DNN

- View
  - ModuleView base class
  - IModuleView interface
- Model
- Presenter
  - ModulePresenter base class

Diving in: DnnSimpleArticle

- Started as “normal” Web Application Project
  - ASPX pages with UI logic and some business logic in code behind
- “Edit” view converted to WebFormsMVP
  - More players, each with a more focused role

Unit Testing Benefits [1]

- Facilitates Change
- Simplifies integration
- Provides Documentation
- Design

[1] http://en.wikipedia.org/wiki/Unit_testing

What do unit tests test?

- Smallest piece of testable software[3]
- Logical and/or complex code
- Core business logic
- High traffic code / mission critical
- Or, in TDD: everything!

[3] http://msdn.microsoft.com/en-us/library/aa292197.aspx

Simple Code

```
namespace SimpleLibrary
{
    using System;

    public class Fibonacci
    {
        public static int Calculate(int x)
        {
            if (x < 0)
                throw new ArgumentOutOfRangeException("x",
                    "Must be greater than or equal to zero.");

            return (x > 1) ? Calculate(x - 1) + Calculate(x - 2) : x;
        }
    }
}
```

Simple Test

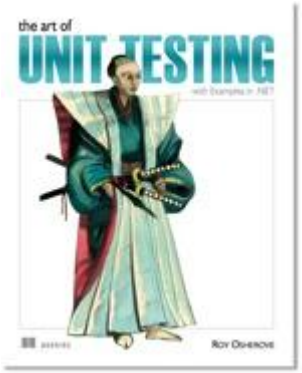
```
namespace SimpleLibrary.Tests
{
    using ...

    public class FibonacciTest
    {
        [Test]
        public void Fibonacci_NumberGreaterThanOne_ReturnsExpectedResult()
        {
            int result = Fibonacci.Calculate(12);
            Assert.AreEqual(144, result);
        }
    }
}
```

Unit Test Characteristics

Unit tests should be: [2]

- Automated & Repeatable
- Easy to implement
- Once written, it remains for future use
- Anyone should be able to run it
- It should run at the push of a button
- Run quickly



[2] The Art of Unit Testing, Roy Osherove, Manning Publications 2009

Testing Frameworks

- Unit Testing “Glue”
  - (Addresses the Automated & Repeatable)
- csUnit, **MbUnit**, MSTest, Nunit, xUnit
- Pick one and go

We use MBUnit & Gallio

- MbUnit
  - Class and Method Attributes
  - Assertions
- Gallio Test Runner
  - Displays results
  - Visual Studio Integration
  - ReSharper Integration

What is testable code?

- Is all code testable?
- Single Responsibility/Separation of Concerns
- Dealing with dependencies

Seams

- "…place where you can alter behavior of your program without editing it in that place" – (Feathers, 2005)

- Plug-in / Injection

Seam / Constructor Injection

```
public class Fibonacci
{
    private readonly IGenericDependency genericDependency;

    public Fibonacci()
    {
        genericDependency = new GenericDependency();
    }

    public Fibonacci(IGenericDependency dependency)
    {
        genericDependency = dependency;
    }

    public int Calculate(int x)...
}
```

Seam / Property Injection

```
public class Fibonacci
{
    public public IGenericDependency Dependency { get; set; }

    public Fibonacci()
    {
        Dependency = new GenericDependency();
    }

    public int Calculate(int x)...
}
```

Two Consumers

- Production Code
- Tests
- Seams help cater to the test

Fakes!

- Stubs
  - Return results
  - Throw exceptions
- Mocks
  - Verify behavior

Isolation Frameworks

- Help us create fakes
- And verify expectations
- Moq, Nmock, Rhino Mocks, Typemock Isolator

My Setup

- MBUnit – Unit testing framework
- MOQ – Isolation framework
- Gallio – Test runner
  - Integrated into Visual Studio / Resharper

Unit Testing: Quick Summary

- Unit testing increases confidence in your code. Code is more changeable.
- Therefore Unit Testing is a desirable practice
- WebFormsMVP allows us to better conduct unit testing in DotNetNuke

Section 3
Introducing: Client Resource Management

Section 3
Client Resource Management

- Background/overview
- Key characteristics
- Client Dependency Framework
- JavaScript and CSS Registration
- Implementation details
- A new development approach

Overview

- Performance is a feature *(an important one)*
- Fast sites lead to satisfied users
- DotNetNuke is largely optimized on the server side, was not so much on the client side

Client Side Performance

80% *of the end-user response time is* ***spent on the front-end****. Most of this time is tied up in* ***downloading all the components*** *in the page: images,* ***stylesheets, scripts****, Flash, etc.* ***Reducing the number of components*** *in turn reduces the number of HTTP requests required to render the page.* ***This is the key to faster pages****.*

-Yahoo! Exceptional Performance Team

DotNetNuke 6 – Resources Overview

- Clean install, home page
  - unauthenticated
    - 6 CSS Files
    - 13 JavaScript Files
  - Logged in as host
    - 8 CSS Files
    - 22 JavaScript Files

Goals for Improvement

- Reduce the file size of each resource
- Only deliver a resource that is needed
- Combine resources into as few as possible

Client Resource Management: Key Characteristics

- Resource Registration API
  - Request a JS or CSS resource be loaded
- File combination
  - Combine all requests of a given type into one file
- Caching / Persistence
  - Cache the combined file / save it to disk
- Reuse
  - Reuse cached files across pages if appropriate
- Versioning
  - Allow for cache busting based on versioning

Client Dependency Framework

- Open Source Framework
- Microsoft Public License (Ms-PL)
- Originally released Early 2010
- Supports MVC & WebForms
- Used in Umbraco
- **Meets all key characteristics on the previous slide**

Step 1: Resource Registration

- Script Loader on page
- Register in code

```
var clientDependencyLoader = (ClientDependencyLoader)page.FindControl("Loader");
clientDependencyLoader.RegisterDependency(styleSheet, ClientDependencyType.Css);
```

- Or register in markup

```
<%@ Register Namespace="ClientDependency.Core.Controls" Assembly="ClientDependency.Core" TagPrefix="CD" %>
<CD:JsInclude runat="server" FilePath="~/Resources/Shared/Scripts/jquery/jquery.hoverIntent.min.js" />
<CD:JsInclude runat="server" FilePath="~/Portals/_default/Skins/DarkKnight/jquery.cycle.min.js" />
<CD:CssInclude runat="server" FilePath="/Portals/_default/Skins/DarkKnight/DNNMega/dnnmega.css" />
```

Resource Registration w/ DNN API

- Wrapped script loader control in Default.aspx
- Register in code using DNN API

```
ClientResourceManager.RegisterScript(this.Page, "~/Resources/Shared/Scripts/jquery/jquery.tmpl.js");
```

- Or register in markup using wrapped controls

```
<%@ Register TagPrefix="dnn" Namespace="DotNetNuke.Web.Client.ClientResourceManagement" Assembly="DotNetNuke.Web.Client" %>
```

```
<dnn:DnnJsInclude runat="server" FilePath="~/Resources/Shared/Scripts/jquery/jquery.hoverIntent.min.js" />
```

DNN 6.1 w/ Client Dependency

- Home page, clean install
  - **Unauthenticated**
    - Debug
      - 8 CSS Files
      - 14 JS Files
      - **22 Total**
    - Release
      - 1 CSS Files
      - 7 JS Files
      - **8 Total**
- **14 Fewer Requests**

debug="true"

- Scripts
  - ControlPanel.debug.js
  - Search.js
  - Telerik.Web.UI.WebResource.axd
  - WebResource.axd
  - dnn.jquery.js
  - dnn.js
  - dnn.modalpopup.js
  - dnncore.js
  - initWidgets.js
  - jquery-ui.min.js
  - jquery.cycle.min.js
  - jquery.dnnmega.debug.js
  - jquery.hoverIntent.min.js
  - jquery.min.js
- Stylesheets
  - ComboBox.Default.css
  - container.css
  - default.css
  - dnnmega.css
  - module.css
  - module.css
  - portal.css
  - skin.css

debug="false"

- Scripts
  - 3cf3e9bbd449c4a24e72230220e645f4.8.js
  - 5af2e6604d72229c207549b4dd6e5572.8.js
  - 99c132b2674a7dc4914a158447ec2b43.8.js
  - Telerik.Web.UI.WebResource.axd
  - WebResource.axd
  - dnn.js
  - initWidgets.js
- Stylesheets
  - ea55dfaef08cac608e278f4009257675.8.css

DNN 6.1 w/ Client Dependency

- Home page, clean install
  - **Logged in as Host**
    - Debug
      - 9 CSS Files
      - 20 JS Files
      - **29 Total**
    - Release
      - 2 CSS Files
      - 7 JS Files
      - **9 Total**
- **20 Fewer Requests**

debug="true"

- Scripts
  - ControlPanel.debug.js
  - Search.js
  - Telerik.Web.UI.WebResource.axd
  - WebResource.axd
  - dnn.controls.dnnlabeledit.js
  - dnn.controls.js
  - dnn.dom.positioning.js
  - dnn.jquery.js
  - dnn.js
  - dnn.modalpopup.js
  - dnn.xmlhttp.js
  - dnn.xmlhttp.jsxmlhttprequest.js
  - dnnactions.debug.js
  - dnncore.js
  - initWidgets.js
  - jquery-ui.min.js
  - jquery.cycle.min.js
  - jquery.dnnmega.debug.js
  - jquery.hoverIntent.min.js
  - jquery.min.js
- Stylesheets
  - ComboBox.Default.css
  - Telerik.Web.UI.WebResource.axd
  - container.css
  - default.css
  - dnnmega.css
  - module.css
  - module.css
  - portal.css
  - skin.css

debug="false"

- Scripts
  - 221f9bd5d9c2b2e7ca1376dfbf108881.8.js
  - 5af2e6604d72229c207549b4dd6e5572.8.js
  - 99c132b2674a7dc4914a158447ec2b43.8.js
  - Telerik.Web.UI.WebResource.axd
  - WebResource.axd
  - dnn.js
  - initWidgets.js
- Stylesheets
  - Telerik.Web.UI.WebResource.axd
  - ea55dfaef08cac608e278f4009257675.8.css

Into the wild

- DNN Core Strength: custom & third party components
- But, usage means resource requests often grow
- Consider these (unauthenticated. As of 8/16/2011)
  - R2integrated.com: 30+ JS files and 5 CSS files
  - DataSprings.com: 18 JS files and 11 CSS files
  - DotNetNuke.com: 16 JS files and 12 CSS files
  - EngageSoftware.com: 23 JS files and 9 CSS files
  - Mybrantford.ca: 17 JS files and 9 CSS files
  - Dreamslider.net: 16 JS files and 9 CSS files

Implementation Details

- Reference Assembly
- Additional web.config section
- Composite files stored in App_Data/ClientDependency
- DNN wrapper API methods
  - RegisterStyleSheet already exists
  - RegisterScript?
  - Wrapper control for user in skins and other controls
- WebUtility and WebControls assemblies need updating
- CDN integration
- Load ordering scheme for both JS & CSS

The New API

- DotNetNuke.Web.Client Assembly
  - RegisterStyleSheet methods
  - RegisterScript methods
  - DnnCssInclude
  - DnnJsInclude

File Combination

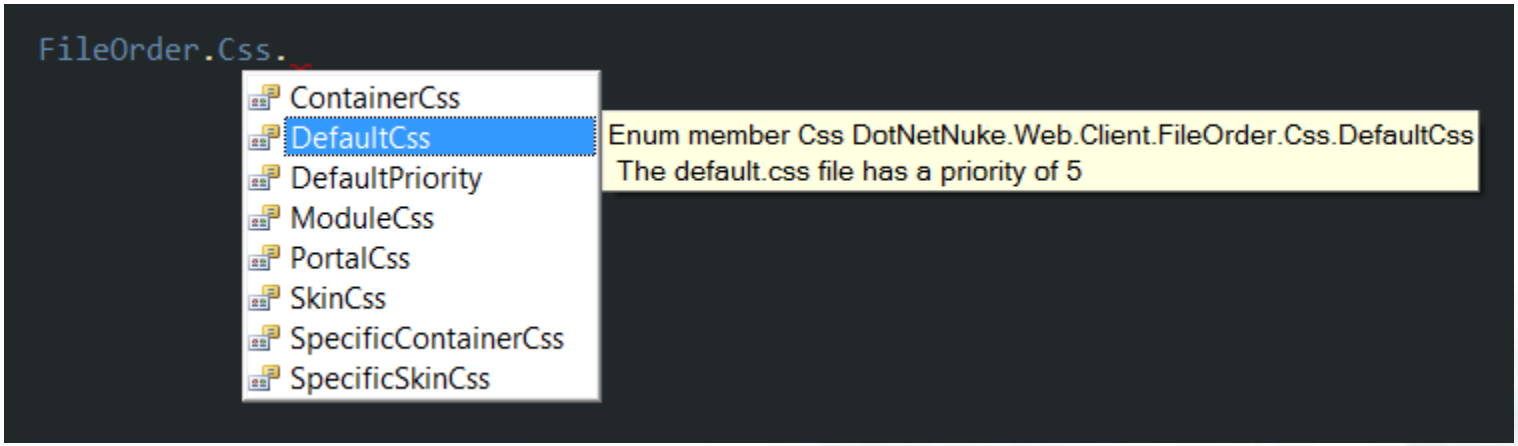
- Duplicates removed based on path/filename
- Combined into one file
- Absolute external URLS (JS & CSS) such as CDN requests are requested separately
- An xml file map is kept on the server
- The dynamic URL is a hash of those file path/names

Location in the Document

- Provider model
- Provider dictates where it is rendered
- Out of the box:
  - LoaderControlProvider
  - PageHeaderProvider
  - LazyLoadProvider
- DNN Provides:
  - DnnPageHeaderProvider (6.1.1)
  - DnnBodyRenderProvider
  - DnnFormBottomRenderProvider

File Ordering

- Integer based relative priority
- DotNetNuke core file order enumeration (spaced by 5)



Caching and Persistence

- ASP.NET Output Caching
  - MSDN: "*On subsequent requests, the page or user control code is not executed; the cached output is used to satisfy the request.*"
- Stored on disk for persistence across application restarts
  - Pulled from disk (not rebuilt) and put in cache

Versioning

- Integer based version number
- Stored in web.config
- Forces a fresh rebuild of the files
- A variety of ways to increment
  - Install an extension
  - Clear the cache
  - Save Portal.css
  - Perform an upgrade

Step 2: A New Development Approach

- Freed up to structure as necessary
  - No longer shove all styles into one module.css file
  - Can break it out into separate files and request as needed
    - CssInclude(‘base.css’)
    - CssInclude(‘ui-widgets.css’)
    - CssInclude(‘gallery.css’)
  - Same with JS files

2011
DotNetNuke™ WORLD
ORLANDO

Resources

General Module Development

- DotNetNuke Wiki (Really!)
  - jQuery Plugins
  - Manifest Files
- Razor's Edge User Map
- IWeb
- DnnSimpleArticle
- Customer Feedback
- C# WAP Module Development Templates
- Updating the Contest Module UI for DNN 6

Simple Made Easy

- Rich Hickey, Oct 20, 2011 @ Strange Loop Conference in St. Louis MO. Please watch!!!

- http://www.infoq.com/presentations/Simple-Made-Easy

WebFormsMVP Resources

- WebFormsMVP.com
- DnnSimpleArticle
- BeerCollectionMVP
- Upcoming: core forums module

Unit Testing Resources

- Books
  - Roy Osherov's "The Art of Unit Testing"
  - Michael Feathers "Working Effectively with Legacy Code"

Client Resource Management Resources

- ClientDependency Framework on Codeplex
- Client Resource Admin Module
- DNN Client Resource Logger Library
- **Blog posts**
  - Enhancements for working with JavaScript and CSS Files in DNN 6.1
  - DNN 6.1 JS/CSS File Combination Potential Gotchas
- **Documentation**
  - DNN Client Resource Management API
  - Client Dependency Framework

2011
DotNetNuke™ WORLD
ORLANDO

Bonus!

Client Capability API

- ClientCapabilityProvider
  - IsMobile
  - IsTablet
  - IsTouchScreen
  - SupportsFlash
  - + more
- Additional Resources
  - http://www.dotnetnuke.com/Resources/Blogs/EntryId/3194/Mobile-APIs-in-6-1.aspx
  - http://www.dotnetnuke.com/Resources/Blogs/EntryId/3208/Mobile-Device-Detection-and-Redirection-ndash-Under-the-Hood.aspx

Templating

- Ask me about templates!

```
<ul id="products">
  {% for product in products %}
    <li>
      <h2>{{ product.title }}</h2>
      Only {{ product.price | format_as_money }}

      <p>{{ product.description | prettyprint | truncate: 200  }}</p>

    </li>
  {% endfor %}
</ul>
```