# Learning paths

MVC: <https://app.pluralsight.com/paths/skills/mvc5>

Bootstrap:

jQuery: <https://app.pluralsight.com/paths/skill/jquery>

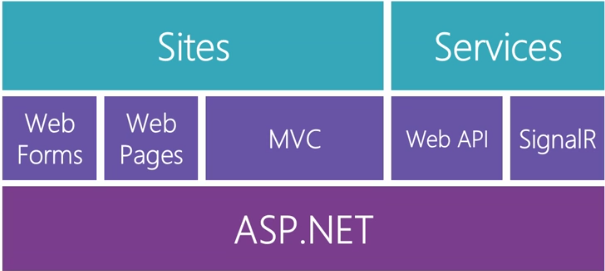
# A comparison of Microsoft web technologies

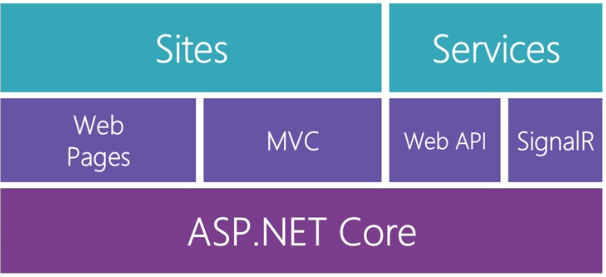
Presenter: Michael Palermo

URL: <https://app.pluralsight.com/library/courses/microsoft-web-technology-comparison>

## An introduction to the Microsoft web platform

### Microsoft web platform





### Demo – Visual Studio tooling

Visual Studio is solution and project orientated, whereas Visual Studio Code is file system orientated – all the text files in the targeted folder are effectively the VS Code workspace.

## Understanding how web forms work

### Describing web forms

With ASP.net web forms you can build dynamic web sites using a familiar drag-and-drop event driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

<http://www.asp.net/get-started/websites>

Based on classic ASP, a mixture of HTML and server-side code. Web forms was designed to be familiar to Windows Forms developers.

Supports separation of markup from logic.

Supports pre-compilation into a single assemble, or dynamic compilation (each page compiled at the time it’s viewed).

Visual Studio integration – design view, controls and data binding.

### Processing web forms

Client sends a HTTP request to the server. On the server the request is first processed by HTTP modules, which determine how the request should be routed and handled. Next the request is processed by a HTTP handler, in the case of web forms the handler will process the code for the requested page. More HTTP modules can then examine the request and response and determine whether any adjustments are needed before sending the response.

### Demo – create a web site

In Visual Studio 2015, file -> new -> project. Select Web in the navigation tree on the left and select the ASP.net web application project type. Another dialogue box opens to choose the template to use – select “web forms”. It’s also possible to add MVC or Web API to the project at this stage.

Key files added to the project by default:

.aspx files are the files which will be requested by the client. Opening one of these files, the developer can switch between design view and source view. Each .aspx file inherits from a namespace and class name, which is contained in the .aspx.cs file which is dependent on the .aspx file.

Site.Master is the master page for the project, containing content which is common to all pages.

Look and feel is managed using the .css files in the Content folder.

Global.asax allows developers to tap into the HTTP pipeline.

### Demo – post data

Add a TextBox, button and an empty label to the page. Double-click the button to create an event handler for it, and update the event handler like this:

protected void Button1\_Click(object sender, EventArgs e)

{

Label1.Text = TextBox1.Text;

}

### Demo – display a list of data

Select the App\_Data folder in the solution explorer, and then select the Project menu -> add new item. With Data selected in the navigation tree, select the SQL Server Database item type. Rename the item to db. Visual Studio creates db.mdf and db\_log.ldf in the solution explorer. Double-click the db.mdf file in the solution explorer, this opens the server explorer, with the db.mdf database visible. Expand the database to see its tables, views, stored procedures etc (it doesn’t have any yet). Add a new table, give it some columns and add some data (this could all be done in SSMS).

Open a .aspx file in the designer and drag the table onto it from the Server Explorer. Right-click, choose a format, and select the columns to display. And that’s it!

## Understanding how web pages work

### Describing web pages

ASP.net web pages and the new Razor syntax provide a fast, approachable and lightweight way to combine server code with HTML to create dynamic web content.

<http://www.asp.net/web-pages>

Also based on classic ASP, but not the first in the ASP.net family. Supports ASP.net core, whereas web forms don’t.

Blend of markup and server code. Code is distinguished from markup in the Razor (.cshtml) syntax using the @ symbol.

Uses dynamic compilation – there’s no assembly.

The closest to classic ASP of all the technologies covered in this course.

### Processing web pages

Client sends a request for a .cshtml file to the server, although the request doesn’t need to include the .cshtml extension. On the server, HTTP modules perform tasks such as routing the request. The HTTP handler then uses the .cshtml file to process the request. More HTTP modules can then examine the request and response in case any adjustments need to be made before sending the response to the client.

### Demo – create a web site

In Visual Studio 2015, File -> new -> web site. Select the ASP.net web site (Razor v3) template.

Key files added to the project by default:

.cshtml files are the pages that the client will request.

\_SiteLayout.cshtml contains content common to all pages, a bit like a master page in web forms. The page content from the individual .cshtml files will be rendered where the @RenderBody() method call is.

Look and feel is controlled using .css files in the Content folder.

\_AppStart.cshtml initialises the site on the first request to the site.

### Demo – post data

In a .cshtml file, add a form and a span to the <article> element:

<article>

<form method=”post” action=””>

<input type=”text” name=”userData” />

<input type=”submit” name=”postAction” value=”Post data” />

</form>

<span class=”message-success”>

@if (IsPost) { <text>@Request.Form[“userData”]</text> }

</span>

</article>

### Demo – display a list of data

This demo uses SQL Server Compact. The project already contains a StarterSite.sdf database file. In the SQL Server Compact pad, right-click “data connections” and select “Add connections from solution”. Expand the StarterSite database to see its tables (it already contains some to help with user authentication etc).

Add a new table, give it some columns and add some data to it.

In the .cshtml file, add new variables to the @{} block at the top of the file (this example assumes that the table is called TechItem and it has a column called Name):

@{

// stuff that’s already there

var db = Database.Open(“StarterSite”);

var items = db.Query(“SELECT Name from TechItem”);

}

Add some more markup to the <article> element:

<article>

<!-- stuff that’s already there -->

<h3>Tech Items</h3>

@foreach (var row in items)

{

<div>@row.Name</div>

}

</article>

## Understanding how MVC works

Model View Controller

### Describing MVC

ASP.net MVC gives you a powerful, patterns-based way to build dynamic web sites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development.

<http://www.asp.net/mvc>

MVC supports ASP.net core as well as ASP.net

Model = data

View = user interface

Controller = processing

### Processing MVC

Client sends a request to the server. If only the host name is supplied, this will route to the default action of the default controller. If the host name and controller but no action is supplied, this will route to the default action of the supplied controller. Or the request can specify the controller, action, and optionally a query string too.

HTTP modules process the request, typically for authentication etc. A MVC HTTP handler is called to pass the request to the appropriate action of the appropriate controller, and finally HTTP modules can examine the request and response to determine whether any adjustments are needed before sending the response to the client.

### Demo – create a web site

In Visual Studio 2015, create a new project. Select Web in the left hand navigation tree and choose the ASP.net web application template. In the next dialogue, select the MVC project type.

Key files added to the project added by default:

Global.asax contains the MvcAccplication class, which contains an Application\_Start method, which calls methods to register areas, filters, routes and bundles. Those methods are defined in classes in the App\_Start folder.

The Controllers folder contains controller classes, with names ending in “Controller”. Each controller class contains an action method, which typically returns an ActionResult instance, which is the View associated with the action method. Each action method responds to a HTTP request of the form /hostname/controllername/methodname (where the controller name is the name of the controller class without the “Controller” suffix).

The Models folder contains any classes used to hold data passed from a controller to a view.

The Views folder contains the .cshtml files which make up the UI. This contains one subfolder per controller, with the same name as the controller class without the “Controller” suffix, and within the subfolders one .cshtml file per action method, with the same name as the method.

### Demo – post data

In an action method, change the ViewBag.Message property to an empty string. In the corresponding view, add a form with a text box and submit button:

@using (Html.BeginForm())

{

<input type=”text” name=”userData” />

<input type=”submit” name=”postAction” value=”Post...” />

<h3>@ViewBag.Message</h3>

}

Return to the controller class and add a new action method for the view, but with a HttpPost attribute, which means the method will only respond to POST request, not GET requests. Give the method a userData parameter, the same name as the text box in the form, this will hold the text from the text box.

[HttpPost]

public ActionResult About(string userData)

{

ViewBag.Message = userData;

return View();

}

### Demo – display a list of data

Add a class to the Models folder called TechItem:

public class TechItem

{

public string Name { get; set; }

public static IEnumerable<TechItem> GetItems()

{

return new List<TecItem>

{

new TechItem{ Name = ”ASP”},

new TechItem{ Name = ”ASP.net”},

new TechItem{ Name = ”ASP.net core”}

};

}

}

Add a new action method to a controller:

public ActionResult List()

{

return View(TechItem.GetItems());

}

Create a view for the action method, using the template “List”

## Understanding how ASP.net core works

### Describing ASP.net core

ASP.net core is a lean framework for building web and cloud applications. ASP.net core is fully open source and available on GitHub.

<http://www.asp.net/vnext>

Originally called ASP.net 5, but renamed because it’s been rebuilt from the ground up, not developed from a previous version.

Can be used with web pages, MVC, web API or SignalR.

Can be used with .net core 1.0 for cross-platform applications.

Can host in IIS or in its own process.

### Processing ASP.net core

Client sends a HTTP request to the server. The HTTP processing pipeline is all in the Startup.cs file – no HTTP modules or handlers.

### Demo – tour of starting template

In Visual Studio 2015, create a new project. With Web selected in the left hand navigation tree, select the ASP.net web application template. In the next dialogue box, under the ASP.net 5 (or ASP.net core) templates section, select the web application project type.

Key files added to the project by default:

project.json lists the assemblies (middleware) upon which the project depends, rather than them being listed in the project file. Also, the .xproj project file doesn’t list the files (classes, views etc) which make up the project, instead the contents of the filesystem are the project files.

StartUp.cs is where the HTTP pipeline is controlled, and is where the services you want to use must be registered.

The Models, Views and Controllers folders work much the same as in MVC.

Views are .cshtml files using Razor syntax, same as in web pages and MVC, however there are lots of new HTML attributes with names beginning with asp-, called tag helpers.

The site can be hosted in a process other than IIS express. Open the project folder in Visual Studio Code and enter the command dnx web. This launches a web server running in a console window. This is possible because project.json contains a reference to the web command.

## A strategic comparison

### Comparing ASP.net to ASP.net core

|  |  |  |
| --- | --- | --- |
|  | ASP.net | ASP.net core |
| Server OS | Windows only | Cross-platform |
| Supported technologies | Web forms, web pages, MVC | Web pages, MVC |
| .net support | .net framework | .net core & .net framework |
|  |  | Unified syntax and controllers |
|  |  | Dependency injection supported out of the box (no need for Unity, NInject etc) |

### Web forms

|  |  |
| --- | --- |
| Strengths | Challenges |
| Established maturity | Bloated object model and events |
| Rapid application development | Viewstate – poor performance |
| Strong 3rd party support | Not supported in ASP.net core |

### Web pages

|  |  |
| --- | --- |
| Strengths | Challenges |
| Server code embedded in HTML | Server code embedded in HTML |
| Dynamic compilation | Smaller adoption |
| Supported in ASP.net core |  |

### MVC

|  |  |
| --- | --- |
| Strengths | Challenges |
| Emphasis on separation of concerns | Stronger developer skills needed |
| Supports ASP.net core |  |
| Strong adoption |  |

# ASP.NET MVC 4 fundamentals

Presenter: Scott Allen

URL: <https://app.pluralsight.com/library/courses/mvc4>

## Introduction to ASP.net MVC 4

### Introduction

Release notes for ASP.net MVC 4: <http://www.asp.net/whitepapers/mvc4-release-notes>

Web API is new in MVC 4, for building web services.

Asynchronous controllers allow you to keep the application responsive even with lots of long-running network calls.

Bundling and minification allow you to optimise style sheets and javascript files.

### Getting started

The demos use Visual Studio 2012, although 2010 could also be used. Create a new project, select Web in the left hand navigation tree and the select ASP.net MVC 4 web application template.

Alternatively, start with a blank solution and add the components manually. Add a Windows class library project to the solution. This is the domain project containing the business objects.

#### Domain objects

In this demo, the business object classes are Employee and Department. Make the properties virtual because we’ll be using Entity Framework and it can override these properties to provide things like change tracking.

public class Employee

{

public virtual int Id { get; set; }

public virtual string Name { get; set; }

}

public class Department

{

public virtual int Id { get; set; }

public virtual string Name { get; set; }

public virtual ICollection<Employee> Employees { get; set }

}

#### Interface representing the database

The domain project also needs an interface representing the application’s data source. This interface doesn’t determine anything about the implementation of the data source (e.g. whether it’s SQL server, in-memory or an imp with a notepad), instead it describes the operations that can be performed on the data source. This is important because it allows dependency injection in the controllers, making them unit testable. The properties are of type IQueryable because this demo uses Entity Framework and will use a lot of Linq queries.

public interface IDepartmentDataSource

{

IQueryable<Employee> Employees { get; }

IQueryable<Department> Departments { get; }

}

Add an ASP.net MVC 4 web application project to the solution. In the next dialogue box select the appropriate template:

* Empty – no content, just project references
* Internet application – includes default content and forms authentication
* Intranet application – includes default content and Windows authentication
* Mobile application
* Web API – for web services.

MVC 4 introduces nuget integration, and adds a packages.config file to the project listing the packages and version numbers used by the project.

### Models, views, controllers

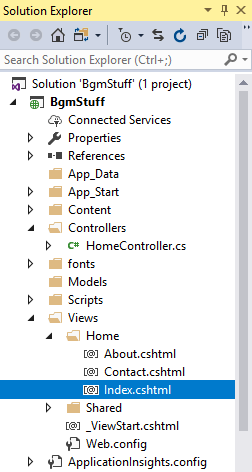
A model holds the data for a view to display. A controller handles a HTTP request, works out what to do with it, typically builds a model and then passes the model to a view. A view just displays the data held in the model.

#### Routing and bundling

Global.asax.cs contains the method which initialises the application, including registering routes and bundles, by calling methods on classes held in the App\_Start folder.

App\_Start\RouteConfig.cs contains the method which registers routes, i.e. how URLs are mapped to controllers and action methods, and default values for if the URL doesn’t specify an action or a controller. The default routing is to use URLs of the form hostname/controllername/actionname, where the controller name is the name of the controller class without the “Controller suffix” and the action name is the name of the action method in the controller class. If no action name is provided, “Index” is assumed, and if no controller name is provided, “Home” is assumed.

#### Controllers

Controllers are in the Controllers folder by default, but don’t need to be, they could be anywhere in the project. The controller class name must be suffixed with “Controller” and be derived from the Controller class. Action methods in controller classes typically have a return type of ActionResult, and build a model and then pass it to a view using the View() method. The View() method’s default behaviour is to return a view with the same name as the action method, from a folder with the same name as the controller (minus the “Controller” suffix), which in turn is within the Views folder. Alternatively, the action method can pass the name of the view to the View() method.

An action method can build an instance of a model to pass data to the view, or they can use properties of the ViewBag object to pass data to the view, or both.

##### An example controller class:

public class HomeController : Controller

{

// responds to hostname/Home/Index

public ActionResult Index()

{

ViewBag.Message = “Your message here”;

// will return the Views\Home\Index.cshtml view

return View();

}

// responds to hostname/Home/About

public ActionResult About()

{

var myModel = new SomeModelClass();

myModel.AProperty = “Hello world”;

// will return the Views\Home\About.cshtml view

return View(myModel);

}

// responds to hostname/Home/Contact

public ActionResult Contct()

{

// will return the Views\Home\ADifferentView.cshtml view

return View(“ADifferentView”);

}

}

#### Views

A view is a .cshtml file, using Razor syntax which is a mixture of HTML markup and C#. Any C# code is prefixed with the @ symbol. To display a literal @ symbol, escape it with another @, i.e. @@. A view has access to the properties of the ViewBag object and also to any properties of the model passed to it by the action method.

##### An example view

@{

ViewBag.Title = “Home page”

}

<h1>@ViewBag.Title</h1>

<div>@ViewBag.Message</div>

##### \_Layout.cshtml

Content which is common to all pages, such as site navigation, is typically contained in \_Layout.cshtml, in the Views\Shared folder. Any views in that folder are available to all controllers in the application. The layout view normally contains the document’s html, head, body etc tags, and a call somewhere to @RenderBody(), which invokes the specific view for the current action.

Views can also contain sections, which are rendered when they layout view calls @RenderSection().

@section mySection {

<div>This is my section</div>

}

And in the layout view:

@RenderSection(“mySection”, required: false)

You can define multiple layout views and select the one to use at runtime if you want.

##### The department manager view – starting point

@{

ViewBag.Title = “Home page”

}

<h1>Department manager</h1>

### Data access

This demo uses Entity Framework version 5 for data access. The web project needs a reference to the domain project in order to be able to use IDepartmentDataSource. For this demo, a folder called Infrastructure is added to the web project, which contains a class which is the concrete implementation of IDepartmentDataSource. Because it uses Entity Framework, it is derived from DbContext. It has properties of type DbSet because that’s what Entity Framework gives us, and has explicit implementation of the IDepartmentDataSource properties which simply return those DbSet properties.

public class DepartmentDb : DbContext, IDepartmentDataSource

{

public DbSet<Employee> Employees { get; set; }

public DbSet<Department> Departments { get; set; }

IQueryable<Employee> IDepartmentDataSource.Employees

{

get { return Employees; }

}

IQueryable<Department> IDepartmentDataSource.Departments

{

get { return Departments; }

}

}

Entity Framework can now use this to create and seed a database, but how to do this is out of scope for these notes – I’m focussing on MVC rather than Entity Framework.

### Showing data

To make the HomeController display a list of all the departments:

public class HomeController : Controller

{

private IDepartmentDataSource \_db;

public HomeController(IDepartmentDataSource db)

{

\_db = db;

}

public ActionResult Index()

{

var allDepartments = \_db.Departments;

return View(allDepartments);

}

// remainder of the class omitted for brevity

}

Note that the controller doesn’t use DepartmentDb anywhere, instead it uses IDepartmentDataSource. Later we’ll introduce another IDepartmentDataSource implementation which uses in-memory collections rather than a database, which can be used in unit testing to remove dependency on the database.

Running this as-is will cause a runtime error “No parameterless constructor defined for this object”. So we need to tell MVC what to pass to the constructor when it’s asking for an IDepartmentDataSource instance. This is called dependency injection.

#### Dependency injection

We now need an Inversion of Control container nuget package to handle dependency injection. There are a few out there, e.g. NInject and Unity. This demo uses StructureMap.MVC3 (which does work with MVC 4, despite the name). Add this package to the UI project. This will add a DependencyResolution folder to the project, containing a class called IoC.cs. Update this class, adding the highlighted line.

public static IContainer Initialize() {

ObjectFactory.Initiallize(x =>

{

x.Scan(scan =>

{

scan.TheCallingAssembly();

scan.WithDefaultConventions();

});

x.For<IDepartmentDataSource>().HttpContextScoped()

.Use<DepartmentDb>();

});

}

This tells the MVC runtime that whenever a controller’s constructor or action method has a parameter of type IDepartmentDataSource, instantiate and pass to it an instance of DepartmentDb, which is a new instance for each HTTP request.

#### Creating a view using scaffolding

Right-click within the Index action method and select “Create view”. Visual Studio can build a view to match the action method, using templates and scaffolding. In this case tick the “create a strongly typed view” box, and in the “model class”, enter “Department (eManager.Domain), and for “Scaffold template” select “List”. This creates the following view as Index.cshtm:

@model IEnumerable<Emanager.Domain.Department>

@{

ViewBag.Title = “Index”;

}

<h2>Index</h2>

<p>

@Html.ActionLink(“Create New”, “Create”)

</p>

<table>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.ActionLink(“Edit”, “Edit”, new { id=item.Id })

@Html.ActionLink(“Details”, “Details”, new { id=item.Id })

@Html.ActionLink(“Delete”, “Delete”, new { id=item.Id })

</td>

</tr>

}

<table>

i.e. a table containing a list of departments, with links to action methods for each one to edit, display details or delete it. Those links are broken at the moment because the HomeController doesn’t have Edit, Details or Delete action methods yet.

Note the @model (lower case m) declaration in the first line, this tells the view what the data type of its model is. We can then use the view’s Model (upper case M) property to access properties of the model. In this case the model’s type is IEnumerable<Department> but in real-world scenarios we’d implement our own model class, which could include a property of type IEnumerable<Department>.

#### Creating a view manually

We don’t have to use scaffolding to create views, we can create them manually, or edit the view created by scaffolding. A simple view for this demo, to display a simple unordered list of department names, could be:

@model IEnumerable<eManager.Domain.Department>

@{

ViewBag.Title = “All Departments”;

}

<h1>@ViewBag.Title</h1>

<ul>

@foreach(var department in Model)

{

<li>@department.Name</li>

}

</ul>

### Listing data

#### @Html.ActionLink method

Updating the highlighted line in the view gives us a hyperlink to a specific action method on a specific controller. The parameters in the overload used here are

* Text to display (department name)
* Action method name
* Controller name
* Parameter to pass to the method
* HTML attributes to give to the <a> element

@model IEnumerable<eManager.Domain.Department>

@{

ViewBag.Title = “All Departments”;

}

<h1>@ViewBag.Title</h1>

<ul>

@foreach(var department in Model)

{

<li>@Html.ActionLink(department.Name, “detail”, “department”, new { department.Id}, null)</li>

}

</ul>

An alternative way of doing this might be

<li><a href=”/detail/department/5 /></li>

But that’s not very flexible. Using the HTML helper lets us generate the URL from the routing information, in case the application’s routing configuration is changed later.

#### A Detail action method

We now need a new controller class in the Controllers folder, called DepartmentController, with an action method called Detail. It’s possible to scaffold a controller from a variety of templates, or we can write one from scratch.

public class DepartmentController : Controller

{

private readonly IDepartmentDataSource \_db;

public DepartmentController(IDepartmentDataSource db)

{

\_db = db;

}

public ActionResult Detail(int id)

{

var model = \_db.Departments.Single(d => d.Id == id);

return View(model);

}

}

The MVC runtime will attempt to supply the value of the id parameter, either from the URL’s query string, or if posting from a form, from the value of an element of the form with the same name as the parameter.

#### @Html.DisplayNameFor and @Html.DisplayFor

And we now need to create the view that this method returns, called Detail.cshtml in the Views\Department folder. This can be created using scaffolding or can be created by hand.

@model eManager.Domain.Department

<h2>Department Detail</h2>

<fieldset>

<legend>Department</legend>

<div class=”display-label”>

@Html.DisplayNameFor(model => model.Name)

</div>

<div class=”display-field”>

@Html.DisplayFor(model => model.Name);

</div>

</fieldset>

<table>

<tr>

<th>Name</th>

<th></th>

</tr>

@foreach (var employee in Model.Employees)

{

<tr>

<td>@employee.Name</td>

<td>

</td>

</tr>

}

</table>

This view will display the name of the department and a table of the employees in that department, along with an action link to add an employee to the department.

Two new HTML helpers here - @Html.DisplayNameFor will write out a label with the name of a property, and @Html.DisplayFor will write out an appropriate control for displaying the value of the property depending on its data type and metadata. See the MVC 3 course for examples of how to influence what those HTML helpers display.

### Creating data

We want to add an action link to the Detail view, to allow users to add an employee to the department:

</fieldset>

@Html.ActionLink(“Create an employee”, “Create”, “Employee”, new {DepartmentId = @Model.Id}, null)

<table>

First, let’s add a property to the Employee class (Entity Framework can update the database schema to reflect this but again that’s out of scope for these notes):

public class Employee

{

public virtual int Id { get; set; }

public virtual string Name { get; set; }

public virtual DateTime? HireDate { get; set; }

}

#### View model for accepting user input

Rather than using a domain object as the model for accepting user input, it’s safer to create a view model containing only the properties that you expect the user to be able to input. This reduces the risk of a malicious user being able to fiddle with properties that they shouldn’t be able to access.

It’s called a view model rather than a model because it’s a model which is specific to the view for creating employees.

public class CreateEmployeeViewModel

{

[HiddenInput(DisplayValue = false)]

public int DepartmentId { get; set; }

[Required]

public string Name { get; set; }

[Required]

[DataType(DataType.Date)]

public DateTime HireDate { get; set; }

}

The attributes decorating the properties are the metadata used by HTML helpers such as @Html.DisplayFor to determine what sort of element to render, and can also be used to validate the user input. So for example, HiddenInput means it’s a property which the user doesn’t input, instead it’s a hidden field on the form. Required means a value must be supplied (used in form validation). And DataType(DataType.Date) means we’re only interested in the date component, not the time.

#### Action method to add an employee

We now need an EmployeeController class with a Create method:

public class EmployeeController : Controller

{

public ActionResult Create(int departmentId)

{

var model = new CreateEmployeeViewModel();

model.DepartmentId = departmentId;

return View(model);

}

}

#### View to add an employee

Right-click within the Create method to scaffold a view for creating an employee. We need to make a few changes to the view created by the scaffolding.

@model eManager.Web.Models.CreateEmployeeViewModel

@{

ViewBag.Title = “Create”;

}

@using (Html.BeginForm()) {

@Html.ValidationSummary(true)

<fieldset>

<legend>CreateEmployeeViewModel</legend>

<div class=”editor-field”>

@Html.EditorFor(model => model.DepartmentId)

</div>

<div class=”editor-label”>

@Html.LabelFor(model => model.Name)

</div>

<div class=”editor-field”>

@Html.EditorFor(model => model.Name)

@Html.ValidationMessageFor(model => model.Name)

</div>

<div class=”editor-label”>

@Html.LabelFor(model => model.HireDate)

</div>

<div class=”editor-field”>

@Html.EditorFor(model => model.HireDate)

@Html.ValidationMessageFor(model => model.HireDate)

</div>

</fieldset>

<input type=”submit” action=”post” value=”Create” />

}

We have some more HTML helpers here.

#### @Html.BeginForm

Writes out an opening <form> tag. As it’s enclosed in a @using block, the closing </form> tag is written automatically when it goes out of scope.

#### @Html.EditorFor

Writes out a suitable control for entering the value of the property based on its data type and metadata.

#### @Html.ValidationMessageFor

Writes out a message which will be displayed if the value entered into the control fails validation.

#### @Html.ValidationSummary

Writes out a summary of all validation errors in the form.

#### Action method to handle posting the form

We need to add a [HttpGet] attribute to the existing Create method so that it only handles GET requests, i.e. when the user first navigates to the create employee page. We then need to add another version of the Create method with a [HttpPost] attribute, to handle the form data being posted to the server.

[HttpPost]

public ActionResult Create(CreateEmployeeViewModel viewModel)

{

if (ModelState.IsValid)

{

var department = db.Departments.Single(d => d.Id == viewModel.DepartmentId);

var employee = new Employee();

employee.Name = viewModel.Name;

employee.HireDate = viewModel.HireDate;

department.Employees.Add(employee);

\_db.Save();

return RedirectToAction(“detail”, “department”, new {id=viewModel.DepartmentId});

}

return View(viewModel);

}

Because the view’s model is a CreateEmployeeViewModel, the runtime will automatically create an instance of that view model and pass it as a parameter to the action method.

We only save the new employee to the database if the entered information is valid according to the metadata we defined on the view model. If it’s not valid then we return the user back to the form with the information they’ve already entered, to give them another chance to get it right.

Because we’re accessing the database, we also need a constructor which will obtain the data source for us and a private field to store it in.

private readonly IDepartmentDataSource \_db;

public EmployeeController(IDepartmentDataSource db)

{

\_db = db;

}

We now get a compilation error because we’re trying to call the Save() method of IDepartmentDataSource, when there’s no such method defined in the interface.

#### Saving changes to the database

So far our only data access has been reading from the database. To save changes to the database we need to add a Save() method to IDepartmentDataSource:

public interface IDepartmentDataSource

{

IQueryable<Employee> Employees { get; }

IQueryable<Department> Departments { get; }

void Save();

}

And this method also needs to be added to the DepartmentDb implementation of IDepartmentDataSource:

void IDepartmentDataSource.Save()

{

SaveChanges();

}

This just calls the Entity Framework SaveChanges method which DepartmentDb inherits from DbContext. How you implement this method will depend on what your data store is and how you interact with it (e.g. if you’re not using Entity Framework then it’ll be different).

### Authorisation

#### [Authorize] attribute

Authorize can be used to decorate a controller class or individual action methods within a controller. Authorize with no parameters just means that the user needs to be logged in to the site (when using the internet application template) or an authenticated Windows user (when using the intranet application template).

Authorize can also be used to restrict access to a comma-separated list of users or roles, e.g.

[Authorize(Roles = “Admin”)]

The rest of this section relates to authorisation using the internet application template, where users and roles are stored in the application database, rather than Windows authentication, so I’ve skipped it.

### Cross-site request forgery

Anti-forgery request token is used to ensure that when a user attempts to post form data to your server, they are posting it from a form which came from your server, rather than from something crafted by a malicious person. It does this by embedding some cryptographic information (the token) in a hidden field in the form, and on handling the HTTP post, compares this with the cookie served to the client when the user first authenticates themself.

Decorate your action method with the ValidateAntiForgeryToken attribute:

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(CreateEmployeeViewModel viewModel)

{

...

This will now fail with a runtime error “The required anti-forgery cookie “...” is not present. To resolve this, add @Html.AntiForgeryToken() to the form in the view.

### Deploying to the cloud

Not completed

## Web optimisation with ASP.net MVC

System.Web.Optimization namespace.

Bundling is concatenating multiple script or css files into a single file.

Minification is optimising these files for quicker download, e.g. by removing comments and whitespace.

### The scenario

A simple MVC application with a single controller and not much business logic, but it includes lots of infrastructure, such as Bootstrap, jQuery, modernizr etc.

#### Application root

“~/” represents the root path of the application.

#### Fiddler

Free tool for examining HTTP requests and responses on your server. Rules menu -> performance -> simulate modem speeds, and disable caching, to get more accurate readings on how long it takes for your pages to load on the client.

#### Internet Explorer developer tools

F12 to open the developer tools. Switch to the network tab and open your site to see how long everything took to load.

See the maximum number of concurrent connections to a server allowed by the browser by switching to the Console tab and entering window.maxConnectionsPerServer.

### Installing Microsoft.AspNet.Web.Optimization

Add the nuget package Microsoft.AspNet.Web.Optimization to your project. It’s already there if you used the ASP.net MVC 4 project template. This also adds a package called WebGrease, which does most of the hard work. WebGrease includes a console application wg.exe for minifying files.

### Configuring bundles

Script bundles and style bundles are registered in BundleConfig.cs in the App\_Start folder.

bundles.Add(new StyleBundle(“~/bundles/css”)

.Include(“~/Content/styles.css”)

.Include(“~/Content/bootstrap.css”));

bundles.Add(new ScriptBundle(“~/bundles/js”)

.Include(“~/Scripts/jquery-1.\*”));

The parameter for the ScriptBundle and StyleBundle constructors is the virtual folder that the files will appear to be served from; this doesn’t need to be a folder that actually exists in your project. The parameter for the Include method is the path to the file within your project. In both cases the tilde (~) represents the root of the web site.

Each bundle is served to the client as a single file, to reduce the number of files to download, because browsers typically limit the number of files that can be downloaded concurrently from a single domain. Wildcards are allowed. Bundling also minifies the file, but only when debug compilation is set to default in the web.config file:

<compilation debug="false" targetFramework="4.5" />

### Virtual paths and relative references

With virtual paths, be careful with style sheets. If styles use images, be sure that the relative path from the style sheet to the images in the virtual path is the same as the relative path from the style sheet to the images in the project file system.

### Rendering bundles

To include a bundle in a page, use the script or style helpers:

@Styles.Render(“~/bundles/css”)

@Scripts.Render(“~/bundles/js”)

### The results are in!

In the demo, bundling and minification reduced the page load time from 60 seconds to 20 seconds.

### Transforming CoffeeScript

There’s a separate course on CoffeeScript. If you use CoffeeScript in your project then you can bundle and minify it by adding the CoffeeBundler package to the project.

## The ASP.net WebAPI

Not completed – relates only to web services

## Async Await in MVC 4

Not completed

## MVC 4 and mobile development

Not completed

# Building applications with ASP.net MVC 4

Presenter: Scott Allen

URL: <https://app.pluralsight.com/library/courses/mvc4-building>

Source code: <http://odetofoodmvc4.codeplex.com>

## Introduction to ASP.net MVC 4

### Web platform installer

To install Visual Studio Express 2012, search the web for “Microsoft web platform installer”. Click the download button and run. Once initialised, search for “2012 web” and select “Visual Studio Express 2012 for web with Windows Azure SDK”. This includes SQL express and IIS express.

### New project

File -> new project -> ASP.net MVC 4 web application. Use the empty template if you want to add things from scratch, internet application for built-in forms authentication, intranet application for Windows authentication.

Nuget support is new in MVC 4. MVC 4 uses HTML 5 best practices.

The view template includes <meta name=”viewport” content=”width=device-width” />. This is important for mobile devices, without it the browser assumes that the page is at least 90 pixels wide and so zooms out to fit this width on the screen, making text too small to read. With this element, content can be rearranged to suit the screen size.

The view template also references modernizr, a javascript library to make HTML5 features work on older browsers which don’t support HTML 5.

### Models, views, controllers

MVC is a design pattern which has been in existence for several decades. ASP.net MVC is Microsoft’s implementation of the MVC pattern.

A HTTP request is handled by a controller, which builds a model containing the data to display to the user, which is passed to a view which is responsible for presenting it. A view doesn’t need to know how to access the database because the controller does that, and a controller doesn’t need to know what colour something should be because the view does that.

ASP.net MVC design goals:

* Embrace the web
* Run on ASP.net
* Extensible
* Testable

### Making changes

Controller classes live in the Controllers folder of the project. A request to /hostname/Home will be handled by the HomeController.cs class and a request to /hostname/Boing will be handled by the BoingController.cs class.

Views are .cshtml files which live in the Views folder of the project. The Views folder has a subfolder for each controller, named after the controller but without the “Controller” suffix, and each subfolder contains a view, typically one for each action method in the controller class and named after the action method, although action methods can specify a different view to return.

A view doesn’t contain all the page content, instead content which is common to all pages is held in a layout view – more about this later.

ViewBag is a dynamically typed object, you can add properties to it in the controller and they’ll be available to the view.

Model classes typically belong in the Models folder (although they can be anywhere, even in a separate project) and provide a strongly-typed alternative to ViewBag for passing information from a controller to a view.

When a view is passed a model, add a directive to the view to tell it what the Type of its model is, e.g (note lower case m).

@model MyModel

Once this directive is added to the view, the Model (upper case M) property and its properties become available to the view.

<div>Value of my property from my model: @Model.MyProperty</div>

### Unit testing

A goal of the MVC framework is to make controllers unit testable without needing a web server, database, network resources or lots of manual testing. Visual Studio 2012 express includes MSTest, which wasn’t available in previous free versions of Visual Studio. As controllers are fairly POCO, they should be easy to unit test in isolation.

### Javascript and CSS

The Scripts folder of the project contains javascript files, both those from nuget packages you’ve added to the project, and your own javascript files. The Content folder contains CSS files, images and other static content.

## Controllers in ASP.net MVC 4

### Routes and controllers

How does ASP.net know how to deliver a request to /hostname/Home/About to the HomeController class? Routing.

routes.MapRoute(

name: “Default”,

url: “{controller}/{action}/{id}”,

defaults: new { controller = “Home”,

action = “Index”,

id = UrlParameter.Optional

})

This statement defines a route with the name Default, which expects the URL to be in the form /hostname/controller/action/id, and sets default values for the controller name, action name and ID if none are present in the URL. This statement is found in the RouteConfig.cs file in the App\_Start folder, and is called from the Application\_Start method in the Global.asax.cs file. To inspect the routing information from a controller:

public ActionResult Index()

{

var controller = RouteData.Values[“controller”];

var action = RouteData.Values[“action”];

var id = RouteData.Values[“id”];

var message = string.Format(“{0}::{1} {2}”, controller, action, id);

ViewBag.Message = message;

return View();

}

Although we wouldn’t normally use RouteData in our own application code, MVC does that for us. The next section shows the normal way of accessing the ID from a URL.

### Actions and parameters

We can add a new route to RouteConfig.cs to match URLs like hostname/Cuisine/French, because we don’t really want to add an action method to the CuisineController for every possible cuisine type, instead we want to treat the cuisine type as an ID rather than an action. The ordering of the routes is important, this new route should be placed before the default route, because the default route will match pretty much any URL, and if more than one route matches the URL then the first one to be defined is the one which takes precedence.

routes.MapRoute(

“Cuisine”,

“cuisine/{name}”,

new { controller = “Cuisine”, action = “Search”, name = “” });

RouteConfig.cs also contains a line

routes.IgnoreRoute(“{resource}.axd/{pathInfo}”);

This tells MVC that requests for .axd files, which are virtual files served by ASP.net but which don’t exist in the filesystem, don’t need to be routed to a controller, because ASP.net itself takes care of those requests.

As of now, a request for /hostname/Cuisine/Swedish returns a 404 error because there’s no Cuisine controller yet, so let’s add one to the Controllers folder.

public class CuisineController : Controller

{

public ActionResult Search(string name)

{

var message = Server.HtmlEncode(name);

return Content(message);

}

}

An action is just a public method in a controller class. If the method has a parameter, MVC will look for the value of that parameter, in the URL, query string and posted form data.

The example above serves text content to the client, whereas normally an action method would return a View. Because this method is returning a Content, it needs to HtmlEncode the parameter it was passed, to guard against a malicious user trying to request a URL containing a script or other dodgy content. We don’t need to do this in action methods which return a View.

### Action results

The Controller base class has a number of methods for returning different types of ActionResult.

|  |  |  |
| --- | --- | --- |
| Name | Framework behaviour | Producing method |
| ContentResult | Returns a string literal | Content |
| EmptyResult | No response |  |
| FileContentResult / FilePathResult / FileStreamResult | Returns the contents of a file | File |
| HttpUnauthorizedResult | Returns a HTTP 403 status |  |
| JavaScriptResult | Returns a script to execute | JavaScript |
| JsonResult | Returns data in JSON format | Json |
| RedirectResult | Redirects the client to a new URL | Redirect |
| RedirectToRouteResult | Redirects to another action or another controller’s action | RedirectToRoute / RedirectToAction |
| ViewResult / PartialViewResult | Response is the responsibility of a view engine | View / PartialView |

### Action selectors

You can map an action to a method with a different name from the action using the ActionName attribute. You can also use HttpGet and HttpPost attributes to control the request types that an action method responds to.

[HttpPost]

[ActionName(“Modify”)]

public ActionResult Edit(int parameter)

{

// ...

}

### Action filters

|  |  |
| --- | --- |
| Name | Description |
| OutputCache | Cache the output of a controller |
| ValidateInput | Turn off request validation and allow dangerous input |
| Authorize | Restrict an action to authorized users or roles |
| ValidateAntiForgeryToken | Helps prevent cross site request forgeries |
| HandleError | Can specify a view to render in the event of an unhandled exception |

Action filters are attributes which can be placed on individual actions, or on controllers to apply to all actions in the controller.

Global filters are registered in the FilterConfig class in the App\_Start folder, e.g. to define a page to handle all unhandled exceptions.

filters.Add(new HandleErrorAttribute());

And turn customErrors on in web.config

<system.web>

<customErrors mode=”remoteOnly” />

There’s an Error.cshtml view in the Views\Shared folder of the project by default.

#### Building a custom action filter

Four methods of ActionFilterAttribute that you can override:

* OnActionExecuting – when the action is about to be executed
* OnActionExecuted – when the action has finished executing
* OnResultExecuting – when the result (e.g. the returned view( is about to be executed
* OnResultExecuted – when the result has finished executing

public class LogAttribute : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

}

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

}

public override void OnResultExecuting(ResultExecutingContext filterContext)

{

}

public override void OnResultExecuted(ResultExecutedContext filterContext)

{

}

}

## Razor views

### Code expressions

Use the @ symbol to introduce C# code in a Razor view, e.g.

<div>We are showing the latest @Model.Count() reviews</div>

Use the tilde (~) to represent the root path of the application.

#### HTML encoding

MVC will automatically HTML encode any property values written to the response stream using @. E.g. if a property value is <script>alert(‘xss’);</script>, the HTML which is written to the response stream is actually &lt;script&gt;alert(&#39;xss&#39;)&lt;/script&gt;, so the code of the attempted script is displayed in the browser rather than being executed.

HTML encoding can be suppressed using @Html.Raw(), if you really want to write HTML held in a model property to the response stream, but be careful with this, and don’t write out data entered by users like this without first validating it for malicious content.

#### Explicit code expressions

Consider the expression @item.Rating / 10. Does this mean display the Rating property followed by the literal “ / 10”? Or does it mean divide the rating by 10 and display the result? It is actually interpreted by Razor as the former. If you want to do the latter, use an explicit code expression by surrounding all the C# code with brackets: @(item.Rating / 10).

Now consider the expression R@item.Rating. The intent is to prefix the Rating property with an R, but Razor interprets this as an email address and displays the literal R@item.Rating. The desired affect can be achieved using R@(item.Rating).

What about displaying a Twitter handle, which starts with an @? Including it in the view as a literal, e.g @OdeToCode, will cause an error because Razor will go looking for a property or object called OdeToCode. In this case, escape the @ with another @, e.g. @@OdeToCode.

### Code blocks

To enter a series of C# expressions in a Razor view, surround them with curly brackets, e.g.

@{

ViewBag.Title = “Index”;

var firstReview = Model.First();

}

But remember, a lot of C# code in a view is probably a bad thing, because it means the view is doing something which should be done by the controller instead.

Curly brackets are also used for iterating through collections, e.g.

@foreach (var item in Model) {

<div>@item.Name</div>

}

Another edge case – if you want to display literal text at the start of a @foreach loop but not within a HTML element.

@foreach (var item in Model)

{

Review

<div> ... </div>

}

This will throw an error because Razor will try to treat the word Review as C# code rather than a literal. To treat it as a literal, escape it with @:

@foreach (var item in Model)

{

@:Review

<div> ... </div>

}

### Layout views

A layout view is similar to a master page in ASP.net web forms. It contains content common to all (or many) pages, such as the head tag and navigation elements. It uses inherited methods such as RenderBody and RenderSection to specify where content from individual pages should appear.

The layout view is typically called \_Layout.cshtml, in the Views\Shared folder of the project.

The layout view to use for the whole site is set in \_ViewStart.cshtml:

@{

Layout = “~Views/Shared/\_Layout.cshtml”;

}

Anything in the code block in \_ViewStart.cshtml will execute immediately before anything in any other view in the application. The main \_ViewStart.cshtml can be overridden by another file of the same name in a subfolder of the Views folder, which will be effective just within that subfolder.

You can also override the Layout property in an individual view file by specifying it in a code block at the top of that view file.

A layout view must contain exactly one call to the RenderBody() method, for Razor to know where to put the content of the specific view. It can contain zero or more calls to RenderSection(), which can render sections of individual views in different parts of the page to the main content. Each section can be defined as optional or required.

@RenderSection(“featured”, required: false)

To set the content of the section in an individual view:

@section featured {

<div>featured content</div>

}

### HTML helpers

Html is a property of the ViewPage base class, with helper methods to create controls, create links, create forms and more.

#### Html.EditorFor

@Html.EditorFor(model => model.FirstName)

Inspects the FirstName property of the model and emits a suitable control for editing it based on its data type, e.g. for a string it will emit a <input type=”text”>, and for a boolean it will emit a checkbox.

#### Html.ActionLink

@Html.ActionLink(“Edit this review”, “Edit”, new { id = item.Id })

Creates a hyperlink with the text “Edit this review”, which will be handled by the Edit action method of the current controller. The third parameter is an anonymously typed object with a property to match each of the parameters on the action method. This could be handled by an action method such as

public ActionResult Edit(int id)

{

var review = \_reviews.Single(r => r.Id == id);

return View(review);

}

#### Html.BeginForm

@using (Html.BeginForm()) {

… form content

}

Writes out an opening <form> tag. If enclosed within a using directive, the closing tag will be written out automatically when it goes out of scope. Can accept additional parameters such as the URL to post the form data to.

#### Html.HiddenFor

@Html.HiddenFor(model => model.Id)

Writes out a <input type=”hidden”> element to include some data item in the form data without displaying it to the user.

#### Html.LabelFor

@Html.LabelFor(model => model.Name)

Writes out a label to accompany an editor for the supplied property. This is good for accessibility, linking the label to the control that it labels.

#### TryUpdateModel – Handling a form post

[HttpPost]

public ActionResult Edit(int id, FormCollection collection)

{

var review = \_reviews.Single(r => r.Id == id);

if (TryUpdateModel(review))

{

// ... save to the database (although we don’t have one yet)

return RedirectToAction(“Index”);

}

return View(review);

}

The TryUpdateModel method attempts to update the model you give it (in this case the review) with information from the request routing data, URL query string and posted form data. It returns true if it succeeded, or false otherwise, in which case in this example we serve the same form back to the user with the data they entered, to let them try again to enter some suitable data. If the update succeeded then we redirect the user to a different action method, so that if they refresh the page it won’t submit the form data a second time.

#### More HTML helpers

There are many more HTML helper methods available than are listed here, see the @Html. intellisense. If you’re writing a lot of C# code in a Razor view, then it may mean you should implement your own custom HTML helper, to get logic out of the view and keep the view as simple as possible.

### Partial views

A partial view allows you to put Razor and C# code which you want to use in multiple places in your application into a single file.

The scenario is that we’ve got some Razor for displaying a restaurant review, which we want to use in a few different places, but we want to ensure that they’re displayed consistently everywhere without repeating code in multiple places.

#### Creating a partial view

Create a new view called \_Review.cshtml (the underscore is just convention, it’ll work without it), and if using scaffolding to create it, tick the “create as a partial view” box.

@model OdeToFood.Models.RestaurantReview

<div class=”review”>

<h4>@Model.Name</h4>

<span>@Model.Rating</span>

<p>@Model.City, @Model.Country</p>

<span class=”right”>

@Html.ActionLink(“Edit”, “Edit”, new { id = Model.Id})

</span>

</div>

#### Using a partial view - Html.Partial

And to use this partial view in a content view:

@foreach (var item in Model)

{

@Html.Partial(“\_Review”, item)

}

#### Accessibility of partial views

In the demo, this partial view was placed in the Views\Reviews folder of the project, making it accessible only to other views in that folder or their subfolders. Placing it instead in the Views\Shared folder would make it accessible to all views in the application.

#### Html.Action

A partial view can only display data which is in its model. Sometimes you want to display data which isn’t part of the model, especially in layout views. For example, to display the best restaurant review in any page whose model isn’t about restaurant reviews:

@Html.Action(“BestReview”, “Reviews”)

And the action method to respond to this could look like this, returning a PartialView.

[ChildActionOnly]

public class ReviewsController : Controller

{

public ActionResult BestReview()

{

var bestReview = from r in \_reviews orderby r.Raring descending select r;

return PartialView(“\_Review”, bestReview.First());

}

}

The ChildActionOnly attribute prevents browsers from accessing this action method directly.

## Working with data

### The entity framework

Framework to access a relational database with strongly-typed LINQ (Language Integrated Query) queries.

This section talks about the schema-first, model-first and code-first approaches, and the different database types supported by EF. Skipping as EF is out of scope for these notes.

### Building entities

This section follows the EF code-first approach and shows how to create the entity classes from which the database is created, retrieve all the instances of an entity from a table and display them on a page. Skipping as EF is out of scope for these notes.

### Database migrations

This section talks about how to influence from the code or configuration things like where the database is created, what it’s called, what the tables are called and so on. Skipping as EF is out of scope for these notes.

### Using LINQ

#### Two types of LINQ query

##### Comprehension Query Syntax

Similar to SQL syntax

var query = from r in \_db.Restaurants

where r.Country == “USA”

orderby r.Name

select r;

There are more keywords available, e.g. for grouping.

##### Extension method syntax

Offers more operations, e.g. Skip() and Take() which don’t have C# keywords.

var query = \_db.Restaurants

.Where(r => r.Country == “USA”)

.OrderBy(r => r.Name)

.Skip(10)

.Take(10);

See <http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b> for more examples, or download [LINQPAD](http://www.linqpad.net), a free utility for trying them out.

#### Projections

Ordering restaurants by their average review rating:

var model =

from r in \_db.Restaurants

orderby r.Reviews.Average(review => review.Rating)

select new RestaurantListViewModel

{

Id = r.Id,

Name = r.Name,

City = r.City,

Country = r.Country,

CountOfReviews = r.Reviews.Count()

};

This requires the creation of a RestaurantListViewModels class. The same can be done using extension methods:

var model =

\_db.Restaurants

.OrderByDescending(r => r.Reviews.Average(review.Rating))

.Select(r => select new RestaurantListViewModel

{

Id = r.Id,

Name = r.Name,

City = r.City,

Country = r.Country,

CountOfReviews = r.Reviews.Count()

});

#### AutoMapper

A nuget package which can map properties of one object to properties of another object: <http://automapper.org>

### A search filter

public ActionResult(string searchTerm = null)

{

var model =

\_db.Restaurants

.OrderByDescending(r => r.Reviews.Average(review.Rating))

.Where(r => searchTerm == null || r.Name.StartsWith(searchTerm))

.Select(r => select new RestaurantListViewModel

{

Id = r.Id,

Name = r.Name,

City = r.City,

Country = r.Country,

CountOfReviews = r.Reviews.Count()

});

}

This needs some markup in the view:

<form method=”get”>

<input type=”search” name=”searchTerm” />

<input type=”submit” value=”search by name” />

</form>

### A restaurant controller

This section demonstrates using scaffolding to create a default controller and views for a particular model. Skipping most of it apart from a few points.

#### HttpNotFound()

public ActionResult Details(int id = 0)

{

Restaurant restaurant = db.Restaurants.Find(id);

if (restaurant == null)

{

return HttpNotFound();

}

return View(restaurant);

}

If there’s no restaurant with the supplied ID in the database, the HttpNotFound() method will return a HTTP 404 status to the browser. Alternatively, you could return View(“NotFound”) to serve a view with with a message to say “sorry, we couldn’t find that restaurant”.

### Listing reviews

This section looks at displaying entities which have linked entities using EF.

#### Binding to a parameter with a different name

To use a query string value such as id=5 as the value of a method action parameter called restaurantId, add the Bind attribute to the parameter in the method signature:

public ActionResult Index([Bind(Prefix=”id”)] int restaurantId)

### Creating a review

This is covered elsewhere in these notes, but it’s worth reiterating, if the user’s doing a post and the information they’ve entered is invalid, redirect them back to the form they were on before posting, so that if they refresh their page, it won’t attempt to repeat the post action.

### Editing a review

More EF stuff in this section, which I’ve skipped.

### Mass assignment

The client could send any data items to your model which you don’t allow in your form, e.g. by editing the URL or by use of other tools. One defence against this is the [Bind(Exclude=)] attribute.

public ActionResult Edit([Bind(Exclude = “ReviewerName”)] RestaurantReview review)

More examples at <http://odetocode.com/blogs/scott/archive/2012/03/11/complete-guide-to-mass-assignment-in-asp-net->

Another defence is to create a class with only the properties that you expect the user to supply, called a view model, and use that, rather than a domain object, as a parameter to your action method.

### Validation annotations

Validation annotation attributes are in the System.ComponentModel.DataAnnotations namespace and can be used to decorate properties of a model class.

To restrict an integer property to values within a range:

[Range(1,10)]

To specify that a property must be supplied (but doesn’t make much sense on a value type, because it’s not nullable):

[Required]

To specify a maximum length for a string property:

[StringLength(1024)]

Other validation attributes are available, including getting the server to validate the value – see MSDN.

Validations are run on the client if javascript is enabled, but if it isn’t then they’ll be run on the server and will set the ModelState.IsValid to false if validation fails, so you should test this in action methods which handle post requests.

To give @Html.LabelFor a friendly name to display, decorate a property with

[Display(Name = “My field”)]

To control how the value is displayed in a control:

[DisplayFormat( /\* named attributes \*/)]

### Custom validations

#### Custom validation

This sort of validation only happens on the server, but you can make it happen on the client using Ajax.

public class MyCustomValidationAttribute : ValidationAttribute

{

private readonly int \_somethingImportant;

public MyCustomValidationAttribute(int somethingImportant)

: base(“{0} isn’t valid for whatever reason”)

{

\_somethingImportant = somethingImportant;

}

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

// value is the thing we’re validating, \_somethingImportant is one of the values we use to validate it

if (value != null)

{

var valueAsString = value.ToString();

if (valueAsString.Length > \_somethingImportant)

{

// passes DisplayName to the format string set in the constructor

var errorMessage = FormatErrorMessage(validationContext.DisplayName);

return new ValidationResult(errorMessage);

}

}

return ValidationResult.Success;

}

}

#### IValidatableObject interface

Implement the Validate method and it will have access to all the form controls, not just the one being validated:

public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)

{

if Rating < 2 && ReviewerName.ToLower().StartsWith(“scott”)

{

yield return new ValidationResult(“Sorry, you can’t do that”);

}

}

The results of this will show up in a @Html.ValidatonSummary element. @Html.ValidationSummary(true) will display only general validation errors like this, @Html.ValidationSummary(false) will display all validation errors, both those associated with a form element and those which are not.

## Ajax and ASP.net MVC

### The scripts

These are some of the scripts included in a MVC 4 project by default.

#### \_references.js

This file tells Visual Studio which other javascript files you commonly use, in order to help it provide intellisense for those files. If you commonly work with other files then you might want to add them to this file.

#### jquery-(version).js

This is the core of jQuery. There are 3 versions of the file.

jquery-(version).intellisense.js is never sent to the client, it just helps Visual Studio to provide intellisense for jQuery.

jquery-(version).js is the main javascript library for jQuery. It’s the readable version of the file, with whitespace and comments.

jquery-(version).min.js is the minified version of jquery-(version).js, with whitepace and comments removed and names shortened to make the file as small as possible, so that it loads more quickly.

There is a feature in MVC 4 for minifying files, more on this later.

#### jquery-ui-(version).js

This is an extension to jQuery which provides controls such as a date picker. It also has verbose and minified versions.

#### jquery.validate.js

Another extension to jQuery which provides client-side validation of forms, with both verbose and minified versions.

#### knockout-(version).js

Allows you to use a MVVM pattern in client side script, with features such as declarative data binding.

#### modernizr-(version).js

Enables HTML5 features in older browsers which don’t natively support them.

#### \*.unobtrusive.\*

These scripts are authored by the MVC team and act as a bridge between MVC and jQuery. You probably won’t need to use them directly.

### Managing scripts

Use the @Scripts.Render method to add a <script> element to a page. Put it in the layout view if it’s a script used by all the pages which use the layout, or put it in a specific view if it’s only used by that page.

Some scripts need to go at the top of the page, e.g. modernizr, which needs to load before the content of the page. Scripts which don’t need to be loaded before the page content should go at the end of the <body> element, so that they don’t delay loading of the visible page content.

@Scripts.Render and @Styles.Render will render <script> and <style> elements for ScriptBundes and StyleBundles defined in RegisterBundles.cs in the project’s App\_Start folder.

Each bundle has two parts, the virtual path and the content of the bundle, e.g.

bundles.Add(new ScriptBundle(“~/bundles/jqueryval”).Include(

“~/Scripts/jquery.unobtrusive\*”,

“~/Scripts/jquery.validate\*”));

The yellow highlighted part is the virtual path from which these files appear to be served. This doesn’t need to be a path which exists in the filesystem. The green highlighted parts are the paths in the filesystem to the files to be bundled.

The filesystem paths support tokens such as {version} and wildcards such as \*.

Render the above example bundle using @Scripts.Render(“~/bundles/jqueryval”).

Bundling will only bundle multiple files into a single file and minify it if the application is in release mode. In debug mode, each file is served individually and unminified. Set this in the web.config:

<system.web>

<compilation debug=”false” />

<system.web>

### Ajax helpers

The aim of this section is to be able to submit a search without losing our scroll position in the page.

#### The scripts we need

First, remove any @Scripts.Render(“~/bundles/jqueryval”) directives from individual views. Then add a new bundle to BundleConfig.cs: (NB: otf = Ode to Food)

bundles.Add(new ScriptBundle(“~/bundles/otf”).Include)

“~/Scripts/jquery-{version}.js”,

“~/Scripts/jquery-ui-{version}.js”,

“~/Scripts/jquery.unobtrusive\*”.

“@/Scripts/jquery.validate\*”));

And then at the end of the layout view, instead of rendering just jQuery, render the new otf bundle:

@Scripts.Render(“~/bundles/otf”)

#### The Ajax form

@using(Ajax.BeginForm(

new AjaxOptions(

HttpMethod=”get”,

InsertionMode=InsertionMode.Replace,

UpdateTargetId=”restaurantList”}))

{

<input type=”search” name=”searchTerm” />

<input type=”submit” value=”Search by term” />

}

Ajax.BeginForm works very similarly to Html.BeginForm except it makes an asynchronous request to the server using Ajax, rather than the synchronous request that a form normally makes. The UpdateTargetId property specifies the ID of the element which will be replaced with the response from the server. By default, this will replace the element with the entire page content, so to avoid this we need to create a partial view called \_Restaurants.cshtml containing the element which displays the search results

<div id=”restaurantsList”>

@foreach (var item in Model)

{

<div>

<h4>@item.Name</h4>

<div>@item.City, @item.Country</div>

<div>Reviews: @item.CountOfReviews</div>

<hr/>

<div>

}

</div>

And in the search page, we replace this element with a call to render the partial view:

@Html.Partial(“\_Restaurants”, Model)

And we need a change to the action method which handles the form posting:

public ActionResult Index(string searchTerm = null)

{

// ...

if (Request.IsAjaxRequest())

{

return PartialView(“\_Restaurants”, model);

}

return View(model);

}

### An async search

#### Unobtrusive javascript

Unobtrusive javascript is an approach which means we don’t have event handlers scattered throughout the HTML like

<a onclick=”handleClick();” ... />

Using unobtrusive javascript means that operations intended to be asynchronous will still work if javascript is disabled, but will be handled synchronously instead.

#### data- attributes

Using the Ajax.BeginForm renders attributes such as

data-ajax=”true” data-ajax-method=”get” data-ajax-mode=”replace”

data- attributes are part of the HTML5 specification, and you can make up whatever data- attributes you want.

#### Client side validation and data- attributes

Client side validation renders data- attributes too, e.g.

<input ... data-val=”true” data-val-number=”The field Id must be a number” data-val-required=”The Id field is required .... />

#### Implementing our own asynchronous form handling

We can implement our own asynchronous form submission using data- attributes. We’re using a data-otf- prefix on our own attributes to avoid any collisions between them and attributes defined by MVC.

<form method=”get” action=”@Url.Action(“Index”)” data-otf-ajax=”true” data-otf-target=”#restaurantList”>

<input type=”search” name=”searchTerm” />

<input type=”submit” value=”Search by term” />

</form>

And some javascript to interpret those data-otf- attributes

$(function() {

var ajaxFormSubmit = function() {

var $form = $(this);

var options = {

url: $form.attr(“action”),

type: $form.attr(“method”),

data: $form.serialize()

};

$.ajax(options).done(function (data) {

var $target = $($form.attr(“data-otf-target”));

$target.replaceWith(data);

});

return false;

};

$(“form[data-otf-ajax=’true’]”).submit(ajaxFormSubmit);

});

This will behave the same way as the earlier example using Ajax.BeginForm.

### Autocompletion

#### Using the jQuery UI autocomplete widget

First we need an action method. The parameter is called term because that’s what the jQuery UI widget sends to the controller. The Linq query returns a collection of anonymous objects with a property called label, being the restaurant name. Again, the property is called label because the jQuery UI widget expects objects with either a label property or a value property, or both.

public ActionResult Autocomplete(string term)

{

var model =

\_db.Restaurants

.Where(r => r.Name.StartsWith(term))

.Take(10)

.Select(r -> new { label = r.Name });

return Json(model, JsonRequestBehaviour.AllowGet);

}

If we call this method directly from the browser with a parameter of “1”, it will return JSON data similar to (although without the whitespace)

[

{“label”:”1”}

{“label”:”10”}

{“label”:”11”}

]

The view now needs something to tell it which element exhibits autocomplete behaviour and where the action method to implement that behaviour is:

<input type=”search” name=”searchTerm” data-otf-autocomplete=”@Url.Action(“Autocomplete”)” />

And we need some javascript to implement autocomplete behaviour for that element. The options object can specify all sorts of things, such as the number of characters the user should type before attempting to autocomplete, or how long to wait after the user stops typing before attempting to autocomplete, but the only mandatory parameter is source, which tells the widget where to get the information to action the autocomplete request.

var createAutocomplete = function() {

var $input = $(this);

var options = { source: $input.attr(“data-otf-autocomplete”) };

$input.autocomplete(options);

}

$(“input[data-otf-autcomplete]”).each(createAutocomplete);

#### Submitting the form automatically when an autocomplete suggestion is selected

This requires an additional parameter in the options object passed to the autocomplete method:

var options = {

source: $input.attr(“data-otf-autocomplete”),

select: submitAutocompleteForm

};

And we need to implement the submitAutocompleteForm function:

var submitAutocompleteForm = function(event, ui) {

var $input = $(this);

$input.val(ui.item.label);

var $form = $input.parents(“form:first”);

$form.submit();

};

#### Highlighting changed content using jQuery UI

Draw the user’s attention to an element which has changed as a result of an Ajax operation, with some new javascript in its done event handler:

$.ajax(options).done(function(data) {

var $target =$($form.attr(“data-otf-target”));

var $newHtml = $(data);

$target.replaceWith($newHtml);

$newHtml.effect(“highlight”);

});

### Paging results

This section uses a nuget package called PagedList.Mvc (which also installs PagedList).

#### Changes to the action method

In the action method, change the Linq query which retrieves the restaurants to display, removing the .Take(10) method call and instead appending a .ToPagedList() call to the end, passing it the page number to display and the page size. The page number will be passed to the action method as a parameter.

public ActionResult Index(string searchTerm = null, int page = 1)

{

var model =

\_db.Restaurants

.OrderByDescending(r => r.Reviews.Average(review => review.Rating))

.Where( r => searchTerm == null || r.Name.StartsWith(searchTerm))

.Select(r => new RestaurantListViewModel

{

// ...

}).ToPagedList(page, 10);

}

#### Default namespaces in web.config

The model for the view is now IPagedList<RestaurantListViewModel> rather than IEnumerable<RestaurantListViewModel>. In order to avoid explicitly specifying the namespace on all classes used in Razor views, you can set default namespaces in the web.config file in the Views folder of the project (not the main web.config file in the root folder of the project).

<system.web.webPages.razor>

<namespaces>

<add namespace=”PagedList” />

<add namespace=”PagedList.Mvc” />

<add namespace=”OdeToFood.Models” />

</namespaces>

</system.web.webPages.razor>

Changes to this web.config element often aren’t picked up until you restart Visual Studio.

#### Changes to the view

Now we need some new markup in the view:

<div id=”restaurantList”>

<div class=”pagedList” data-otf-target=”#restaurantList”>

@Html.PagedListPager(

Model,

page => Url.Action(“Index”, new { page }),

PagedListRenderOptions.MinimalWithItemCountText)

</div>

@foreach (var item in Model)

{

...

}

</div>

There are quite a few different members of the PagedListRenderOptions enumeration available, try them!

#### Making the pager control look pretty

The PagedList package installs a PagedList.css file into the Content folder of the project. Include this in a bundle which is referenced by the page. Now the pager control looks a lot better.

#### Making paging asynchronous

var getPage = function() {

var $a = $(this);

var options = {

url: $a.attr(“href”),

data: $(“form”).serialize(),

type: “get”

};

$.ajax(options).done(function(data) {

var target = $a.parents(“div.pagedList”).attr”data-otf-target”);

$(target).replaceWith~(data);

});

});

$(“main-content”).on(“click”), “.pagedList a”, getPage);

## Security and ASP.net MVC 4

### Windows authentication

#### Why am I prompted to log in to the site?

Internet Explorer only recognises you as being authenticated by Windows if you’re logged into the domain where the site is hosted, and it doesn’t consider localhost to be part of your domain. To resolve this, add the localhost DNS name to the local intranet zone.

@User.Identity.Name returns the user’s ID (this works in forms authentication too).

Windows authentication is set in the web.config:

<system.web>

<authentication mode=”Windows” />

</system.web>

### Forms authentication

<system.web>

<authentication mode=”Forms” />

</system.web>

Skipping the rest of this section.

### Taking control of membership

Skipping this section as it relates only to forms authentication.

### Forms authentication in action

Skipping this section as it relates only to forms authentication.

### Authorize

Decorate controllers or their action methods with the Authorize attribute to control who can access them.

Authorize with no parameters means that only authenticated users can access the action method.

If a class is decorated with Authorize, you can permit anonymous users to access individual action methods by decorating them with the AllowAnonymous attribute.

The Authorize attribute can take a Users parameter:

[Authorize(Users = “sallen, pall”)]

The Authorize attribute can also take a Roles parameter:

[Authorize(Roles = “administrators, sales”)]

Users and Roles work with both forms authentication and Windows authentication. In Windows authentication, Users means the active directory user and Roles means the active directory groups that the user is a member of.

### Seeding membership

Skipping this section as it relates only to forms authentication.

### Cross site request forgery

Make sure that the authenticated user is submitting the form that you gave them, and not a form that a malicious person has crafted.

To protect against this, decorate the action method which handles the form posting with a ValidateAntiForgeryToken attribute. To create the anti-forgery token in the view which contains the form, call the @Html.AntiForgeryToken() method from within the form.

### OpenID and OAuth

[OpenID](http://openid.net) and [OAuth](http://oauth.net) are both open standards for using a 3rd party (e.g. Microsoft or Twitter) to authenticate your users.

[DotNetOpenAuth](http://www.dotnetopenauth.net) is an open source module for using these standards in MVC applications.

Skipping this section.

## ASP.net MVC 4 infrastructure

### Caching

Decorate action methods which are called frequently with the OutputCache property to cache the return value of the method, so that subsequent calls can return the same value without executing the method again.

[OutputCache(Duration = 60, VaryByParam = “none”)]

public ActionResult Index()

{

// ...

}

The Duration parameter is how long the return value is cached for in seconds.

Return values from child actions can also be cached.

[ChildActionOnly]

[OutputCache(Duration = 60)]

public ActionResult SayHello()

{

return Content(“Hello”);

}

And the Razor markup to call it

@Html.Action(“SayHello”)

### Cache settings

#### OutputCache parameters

The following are all parameters you can pass to the OutputCache attribute.

##### VaryByParam

Vary by “\*” to cache for every permutation of all parameters.

Vary by “none” to always cache the same content.

Vary by “name” to cache for every value of the name parameter.

##### Location

Cache on the server, the client, or both

##### VaryByHeader

Vary based on a HTTP header, such as the client’s language settings.

##### VaryByCustom

Vary based on the result of a custom method in global.asax, using any information available to you from that class.

##### SqlDependency

Cache until data in a SQL server table changes.

#### Demo of caching

Let’s start by increasing the cache duration of the action method which searches for restaurants, from 60 seconds to 1 hour.

[OutputCache(Duration = 360)

public ActionResult Index(string searchTerm = null, int page = 1)

Navigating from one page of results to the next results in a call to the action method the first time a page number is requested, and another when another page number is requested, but those requests are cached so a request for a previously seen page number does not execute the method again.

Changing the search term also results in a new call to the action method, the results are cached, and searching for the same search term a second time returns the cached result instead of executing the action method.

However, navigating directly to a URL which includes a query string, e.g. <http://hostname/?searchTerm=20>, returns the cached response for those parameters, which contains only what the action method returned, i.e. page content but without the layout view. This is because this view returns search results using Ajax, rather than returning the entire page each time a search is performed. To resolve this, pass an extra parameter to the OutputCache attribute:

[OutputCache(Duration = 360, VaryByHeader = “X-Requested-With”, Location = OutputCacheLocation.Server)]

This is because the HTTP header for an Ajax request will contain this item, whereas a normal request won’t. This allows the action method to distinguish between an Ajax request and a normal request. The Location parameter forces the caching to be done on the server, because some browsers can’t tell the difference between Ajax and normal requests.

An alternative solution to this problem would be to have separate action methods for Ajax requests and normal requests.

When testing anything which uses caching, it’s a good idea to open the browser’s developer tools (F12) and clear the cache.

### Cache profiles

#### To cache or not to cache?

If you don’t know which action methods are being called most frequently, you risk either wasting memory caching things which are hardly ever used, or not caching things which are used all the time.

#### Cache profiles

Define cache profiles in web.config thus:

<system.web>

<caching>

<outputCacheSettings>

<outputCacheProfiles>

<add name=”Aggressive” duration=”300”>

<add name=”Mild” duration=”10”>

</outputCacheProfiles>

</outputCacheSettings>

</caching>

</system.web>

And reference them by name in the OutputCache attribute using the CacheProfile parameter

[OutputCache(CacheProfile = “Aggressive”)]

### Localisation

Thread.CurrentCulture impacts formatting, e.g. in what DateTime.Now.ToString() will return.

Thread.CurrentUICulture impacts resource loading.

ASP.net can set these properties for you based on the Accept-language HTTP header in the request, or you can set them yourself. To allow ASP.net to set these for you, you need to include a section in web.config:

<system.web>

<globalization culture=”auto” uiCulture=”auto” />

</system.web>

#### An experiment

@{

var amount = 10.0m;

var someDate = new DateTime(2002, 7, 9);

}

<div>

@amount.ToString(“c”) // formats as a currency

</div>

<div>

@someDate.ToShortDateString()

</div>

Running this on Scott’s machine, which is set up to use US English, the output is

$10.00

7/9/2002

Go into tools -> options -> appearance -> languages and change to a different preferred language. The page now shows:

10,00 €

09/07/2002

### Resources

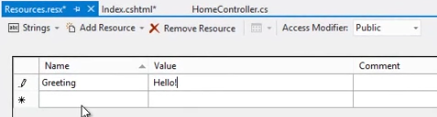
.resx files can store localised text and binary assets (such as images, although they’re not used in this section)

* In this demo,
  + Strings.resx will store default resources
  + Strings.es.resx will store resources for Spanish culture

In Razor, you can access the Greeting property in a Strings resource file using @Resources.Strings.Greetings.

#### Adding resources to the project

For this demo, we’re adding a resources file which is only used by the views for the HomeController, so we add a resources file to the Views\Home project folder, called Resources.resx. As we’re using the resource file from Razor views, we need to change the resource file’s access modifier from internal to public, because Razor views are compiled into a separate assembly from the rest of the MVC project.



This Resources.resx file has a dependent code file Resources.Designer.cs, which contains all the magic needed to serve up resources.

#### Using the resources in a Razor view

<div>

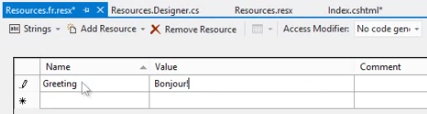
@OdeToFood.Views.Home.Resources.Greeting

</div>

At the moment, this will only display the greeting in English, regardless of the user’s CurrentUICulture, because the only resources we’ve got are in English.

#### Adding resources in another language

Add another resource file to the Views\Home folder, called Resources.fr.resx



This file doesn’t have a dependent .Designer.cs file, because all the code needed is in the base .Designer.resx file.

If any of your action methods use the VaryByHeader parameter of the OutputCache attribute (e.g. to distinguish between Ajax and non-Ajax requests, and the user changes their CurrentUICulture and refreshes the page, they won’t get the resources for the new culture unless you change the parameters passed to the OutputCache attribute:

[OutputCache(CacheProfile = “Long”, VaryByHeader = “X=Requested-With;Accept-Language”, Location = OutputCacheLocation.Server)]

#### Accessing resources from code

Accessing resources from a controller:

var greeting = OdeToFood.Views.Home.Resources.Greeting;

#### Accessing resources from data annotation attributes:

[Required(ErrorMessageResourceType = typeof(OdeToFood.Views.Home.Resources), ErrorMessageResourceName = “Greeting”)]

Although obviously, using the greeting as an error message doesn’t make sense, you’d need to add a resource string to the resource file for the error message you want.

### Diagnostics / health monitoring and ELMAH

#### Logging options

* ASP.net’s own health monitoring
* Log4net – open source library
* ELMAH – error logging handling and modules, can store results in a variety of destinations (XML, database, Twitter etc)

#### ASP.net health monitoring

This is controlled in the machine level web.config file for your .net version, e.g. C:\Windows\Microsoft.Net\Framework\v4.0.30319\Config\web.config, but individual settings can be overridden in your application’s web.config.

There are several important sections

##### <providers>

This section defines the destinations that you can publish events to, e.g. event log, SQL etc.

##### <eventMappings>

This section groups events into groups with friendly names

##### <rules>

This section determines which destination to publish events to, based on the group they are in as defined in the eventMappings element.

##### Using ELMAH

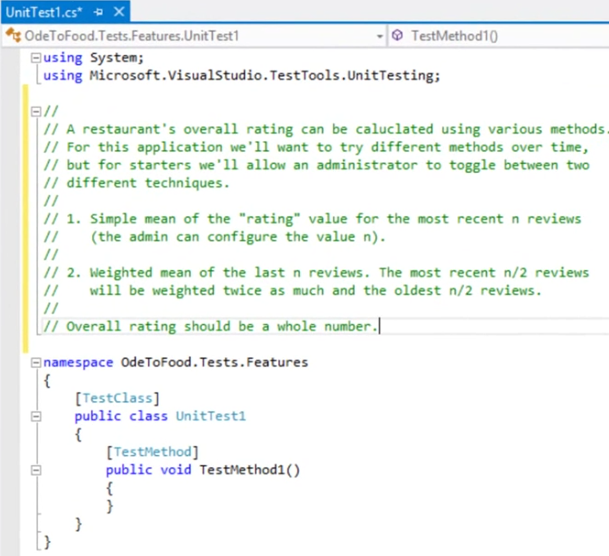
Skipping this section. See more at <http://code.google.com/p/elmah>

## Unit testing with ASP.net MVC 4

### Test Driven Development

This section and the next are about unit testing generally, rather than being specific to MVC. If you’re already familiar with unit testing, skip to the [Home controller tests](#_Home_controller_tests) section.

Add a unit test to the Features folder in your unit test project, and add some comments describing what the tests need to prove, or what the feature needs to do:

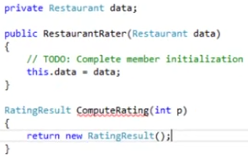


We know we’ll be working with the Restaurant class and its reviews, and it’ll need at least one review. We’ve decided that we need a RestaurantRater class to come up with a rating, and we’re going to tell it to work out the rating based on the last 10 reviews. Finally, we think that a restaurant with one review with a rating of 4 will result in an overall rating for the restaurant of 4.



Of course this won’t compile yet, because there’s no RestaurantRater class. Using Visual Studio you can right-click on the call to the RestaurantRater constructor and create a RestaurantRater class. Or you could create the class yourself. VS will create it in the unit test project, and eventually it needs to be somewhere in the application, but it’s OK to leave it in the unit test project for now.

The RestaurantRater class now needs a ComputeRating method, you can use VS to create this or you can create it yourself. Refactor it a bit so that the interface meets your expectations, and this is what it looks like



This still won’t compile because we need a RatingResult class, so let’s create that.



The project will now compile, so we run the unit tests, which fail, because we’re expecting a rating of 4 and we got a rating of 0.

[TestMethod]

public void TestMethod1()

{

var data = new Restaurant();

data.Reviews = new List<RestaurantReview>();

data.Reviews.Add(new RestaurantReview() { Rating = 4 });

var rater = new RestaurantRater(data);

var result = rater.ComputeRating(10);

Assert.AreEqual(4, result.Rating);

}

The simplest way to get this test to pass would be to change the ComputeRating method to return 4.

public RatingResult ComputeRating(int p)

{

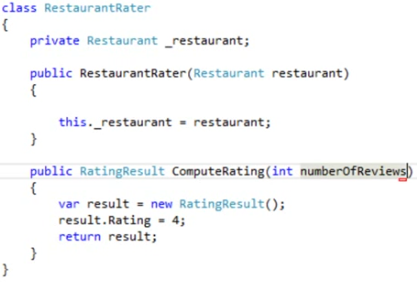
var result = new RatingResult();

result.Rating = 4;

return result;

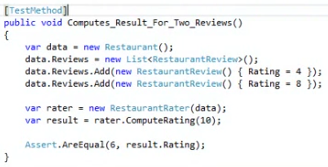
}

This seems a bit silly because the thing we’re testing clearly doesn’t do what it should do, it’s returning a hard-coded value, but we still have a valid test, so we can now start refactoring the code being tested, and we can use the test to check that the refactoring hasn’t broken anything.

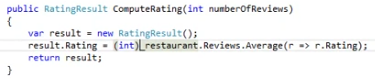


### Test Driven Design

#### A second test method



And a more sensible implementation of the ComputeRating class



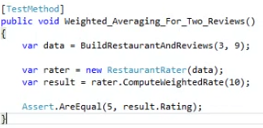
#### Refactoring out common setup code

We’ve now got two passing tests, however there’s quite a lot of common setup code in both, so we refactor this into a separate private method, which is called from both test methods.

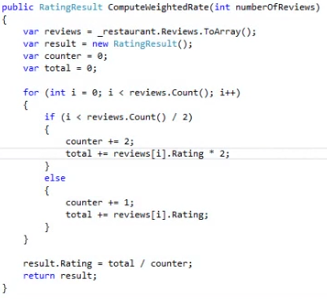


#### TDD of a second rating algorithm

We’ve implemented the first requirement, to calculate a rating just by averaging all the reviews, but we need to implement the second requirement, to calculate a rating by weighting the most recent n/2 reviews twice as much as the earlier reviews. We also need to bear in mind that there may be additional algorithms added in the future. So we start by writing a test method for the second rating algorithm.



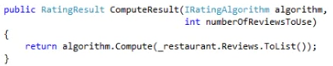
And then an implementation of the second rating algorithm.



At this point we start thinking about things such as the fact that the rating must be an integer, but the calculations can return a non-integer, so how should we convert that to an integer, i.e. should we always truncate, or when should we round up or down. If necessary this should prompt questions to the business to clarify those requirements, and then test methods can be written to ensure that the business logic does the correct thing in each of these scenarios.

#### Refactoring to make future change easier

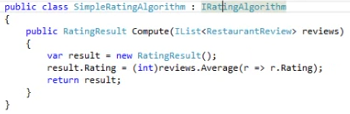
Rather than add a new rating method every time the business come up with a new algorithm, which could soon become difficult to manage, we can refactor things to make future additions easier. First we have a single rating method, with one of its parameters being the rating algorithm to use.

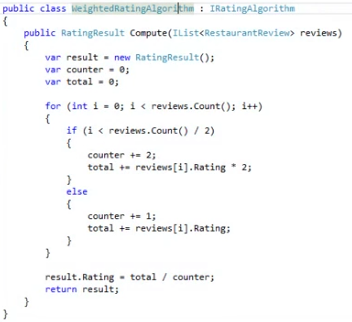


The IRatingAlgorithm interface looks like this

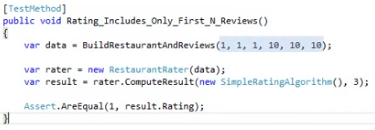


And its two (so far) implementations

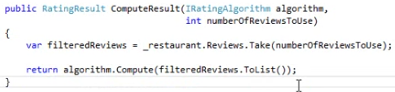




Now we add another test method to check whether the rater calculates the rating using the number of reviews it’s been told to use, creating a restaurant with 6 reviews but telling the rater to only use 3 of them.



This test will fail because the RestaurantRater uses all the restaurant’s ratings. We can fix this by filtering the reviews in the ComputeResult method.



This means that restricting the number of ratings to take into account is handled all in one place, so each individual algorithm doesn’t need to do it.

### Home controller tests

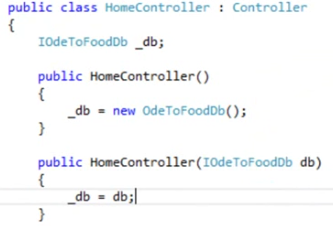
To unit test a MVC application it’s best to write tests for the controllers, and you need to think about isolating the controllers from any infrastructure they use (e.g. databases). You want the tests to run predictably, which means if your controllers are dependent on infrastructure then your setup code needs to put that infrastructure into a known state before each test, which can become unmanageable.

#### Isolating controllers from a database

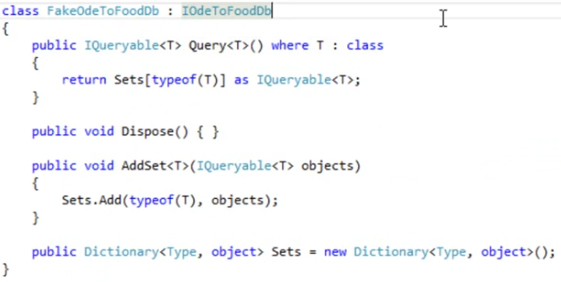
Rather than the controllers interacting with a database directly, they should interact with an interface representing all the logical operations which can be performed on the database, and the actual data access class will implement this interface.



The controller needs to work with the interface rather than the concrete implementation.



The second constructor is the one we’ll use in unit tests, to which we can pass a different implementation of IOdeToFoodDb, e.g. one which works with an in-memory set of data rather than a database.



To minimise setup code, there’s another class which returns an initialised set of restaurants.



The setup code in a test method could then look something like

[TestMethod]

public void Index()

{

var db = new FakeOdeToFoodDb();

db.AddSet(TestData.Restaurants);

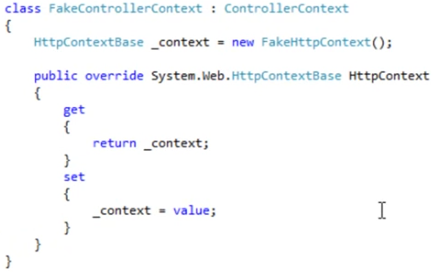
HomeController controller = new HomeController(db);

// ...

}

#### Isolating controllers from HTTP requests

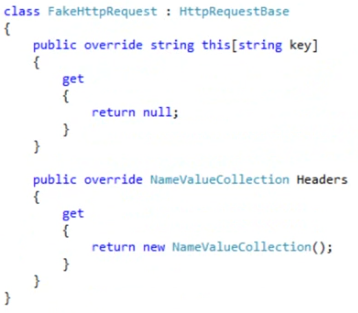
The test for this particular controller now fails because the Index action method tests the Request.IsAjaxRequest() method, and in the context of a unit test we have no HTTP request. To resolve this we need a FakeControllerContext class to give us something to use in place of the actual HttpContext.



And a FakeHttpContext class



And a FakeHttpRequst class



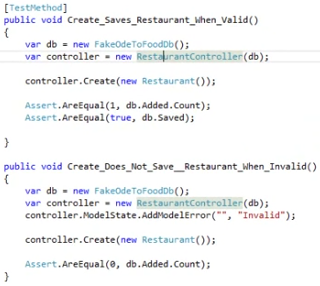
The test method can now set the controller’s ControllerContext to a FakeControllerContext.



The test still fails, but now it’s failing in the Assert call because we’ve changed the view which is returned, but haven’t changed what the Assert is testing for. In the case of the HomeController’s Index method we’re expecting it to display 10 restaurants out of the 1000 in the fake database, so we could assign the return value of the action method to a variable, cast its Model property to its actual runtime type, and assert that the model’s Count() property is 10.

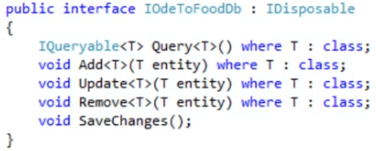
### Create action tests

To test an action method which creates something in the database, we want two test methods, one to test that a record is added when the model is valid, and one to test that nothing is added when the model is invalid.



These show how a fake should make things easier to test, e.g. by exposing as properties the number of records added and whether or not changes have been saved. In order to do this, the data store interface has some new properties.

#### Updated data store interface



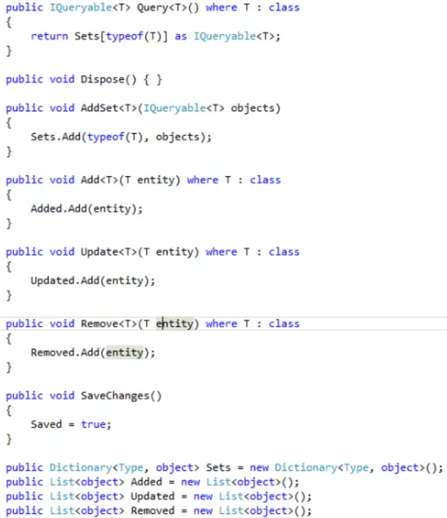
#### Concrete data store implementation

In the concrete implementation, we forward these calls to the appropriate Entity Framework methods.



#### Fake data store implementation

And in the fake implementation we expose properties so that the test methods can see what has been added, updated or removed.





## Deployment and configuration

### Configuration files

The configuration files are XML files which control environmental settings such as authentication, compilation, connection strings, cryptography and custom errors. The configuration is also extensible.

The easiest way to store custom values in the configuration is to add it to the appSettings section of the file.

<appSettings>

<add key=”MailServer” value=”mail.server.com” />

</appSettings>

And retrieve the value using the ConfigurationManager class

ViewBag.MailServer = ConfigurationManager.AppSettings[“MailServer”];

### Configuration hierarchy

Each application inherits settings from higher level config files

* Machine config
* Machine web.config
* Parent’s web.config
* Your web.config

Lower level settings override higher level settings.

The machine config file is C:\Windows\Microsoft.NET\Framework\[version]\Config\machine.config.

The machine web.config file is C:\Windows\Microsoft.NET\Framework\[version]\Config\web.config.

### Hosting ASP.net MVC

MVC projects build to a .dll file, which needs a host process to execute. IIS express is a development host which runs under your identity. IIS is a production web server, available on both server and personal versions of Windows, but switched off by default.

#### Installing IIS

The easiest way to install IIS is to use the Microsoft Web Platform Installer from the start menu. Search for IIS and select “IIS ASP.net 4.5” and “IIS management console”. Also search for and select “SQL Express” and “SQL Server Management Studio” for a local install because getting IIS to work with LocalDB is a bit of a pain. Once it’s all installed, verify it by navigating to <http://localhost>, which should display the IIS logo.

### Preparing for deployment

#### Tidying up the database migrations

In the Entity Framework Configurations constructor, set AutomaticMigrationsEnabled to false and delete the \_InitialCreate.cs file. This resets the migrations to start from a new database, as it doesn’t contain any production data yet. Delete the database too, to ensure you’re going to build a new database. In the package manager console, enter the command.

Add-Migration InitialCreate

### Deploying to IIS

Right click on your MVC project and select “Publish”. Create a new publishing profile and give it a name. Select “Web Deploy Package” as the publish method. Enter the path and filename for the .zip file which will contain the package. Enter “Default Web Site” as the site/application, to make this application the one which launches when you browse to <http://localhost>. On the next screen you can change the connection string and authentication method used to connect to the database.

Once the wizard is complete, open a command prompt as administrator, cd to the folder where the .zip file was created, and there’ll be a [package name].deploy.cmd file which you can run to deploy the package, e.g.

release.deploy.cmd /Y

Browsing to <http://localhost> now displays an error page because the identity the app pool is running as doesn’t have access to the database. We need to use SQL Server Management Studio to grant this identity access to the database with the db\_datareader, db\_datawriter and db\_ddladmin roles. The application now displays correctly at the <http://localhost> URL.

### A second deployment

Subsequent deployments are much easier. Say we want to remove the displaying of the MailServer configuration item from the home page and redeploy. Make the required change, right click on the MVC project and select “Publish”, and now the publishing profile has already been created, so we just select it and click “Publish”. We then run the deploy.cmd again, which will deploy the updated application to IIS.

### Deploying to Windows Azure

Skipping this section.

# ASP.NET MVC 5 fundamentals

Presenter: Scott Allen

URL: <https://app.pluralsight.com/library/courses/aspdotnet-mvc5-fundamentals>

New to MVC? Try [ASP.net MVC 4 Fundamentals](#_ASP.NET_MVC_4) first, as this course is mainly about the differences between versions 4 and 5.

## Introduction and prerequisites

This course concentrates on new features of MVC 5 so assumes the reader is familiar with MVC 4. The fundamentals of models, views and controllers are unchanged.

OWIN and Katana middleware are new in MVC 5, and identity and security are completely new in MVC 5.

Upgrading an application from MVC 4 to MVC 5 is covered in this introduction section. Note MVC 4 uses Bootstrap 2 and MVC 5 uses Bootstrap 3, which is a breaking change.

Need to target .net 4.5.0 as a minimum to use MVC 5.

MVC 5 uses Microsoft.AspNet.Razor version 3 and Microsoft.AspNet.WebPages version 3.

## OWIN and Katana

### Katana

The System.Web assembly is the core ofASP.net all in one assembly. Katana aims to provide the same features, but in a more modular way, so your site only uses the assemblies it needs. Katana is built on a standard called OWIN.

### OWIN – Open Web Server Interface for .net

From [owin.org](http://owin.org): OWIN defines a standard interface between .net web servers and web applications. The goal of the OWIN interface is to decouple server and application, encourage the development of simple modules for .net web development, and, by being an open standard, stimulate the open source ecosystem of .net web development tools.

Katana is Microsoft’s implementation of the OWIN standard.

### WebServer console

Creating a MVC ASP.net application project in Visual Studio 2013 will add references to various Microsoft.Owin\* assemblies, as well as an assembly called just Owin. (These references aren’t present in a Visual Studio 2017 MVC project – skipping the rest of this section because of this).

## Identity and security

### Authentication options

* No authentication – all of your site is open to everyone
* Individual user accounts – forms-based authentication, user profiles stored in SQL server, typically used for internet sites. Also allows users to sign in using their Facebook, Twitter, Google etc account.
* Organizational accounts – authenticate users with Active Directory, Azure Active Directory or Office 365. App needs to be registered with the Azure portal.
* Windows authentication – for intranet applications, the user is already authenticated to your Windows network.

### Registration

Skipped as this relates to individual user accounts, and I’m only interested in Windows authentication.

### [Authorize]

The Authorize attribute can be used to decorate action methods or controller classes, same as in MVC 4.

If a controller class is decorated with the Authorize attribute, anonymous access to an individual action method within the controller can be granted by decorating that action method with the AllowAnonymous attribute.

The Authorize attribute can accept some parameters

[Authorize(Users=”John, Paul, Ringo, George”)]

[Authorize(Roles=”admin, powerusers”)]

Find out about the current user using User.Identity

Find out if the current user is in a role using User.IsInRole

### The database

Skipping this section as it relates to individual user accounts.

### Core identity

Microsoft.AspNet.Identity.Core namespace.

IUser, IRole, IUserStore, IUserLoginStore, IUserPasswordStore.

Skipping this section as it relates to individual user accounts and people wanting to implement their own data store for authenticating users.

### Identity.EntityFramework

Microsoft.AspNet.Identity.EntityFramework namespace.

Contains concrete types which implement the interfaces from the previous section.

Skipping this section as it’s about the SQL server / Entity Framework implementation of individual user accounts.

### UserManager class

Skipping this section as it relates to individual user accounts

### Migrating and seeding

Skipping this section as it relates to Entity Framework

### External logins

This feature allows you to authenticate users using a 3rd party, e.g. Facebook, so you don’t need to manage users’ passwords etc, and your users don’t need to remember yet another password. Typically you need to register your application with the 3rd party and include a secret that they provide in your code in order for authentication to work.

Skipping this section.

### Third party claims

Skipping this section.

## Bootstrap 3

Sleek, intuitive and powerful mobile-first front-end framework for faster and easier web development.

Originally developed by Twitter but now open source.

Included in a MVC 5 project by default.

The single responsibility principle applied to CSS – see <http://tinyurl.com/7gkg6gl>

See the [Bootstrap 3 course](#_Bootstrap_3)

## WebAPI 2

For creating RESTful HTTP web services. Skipping this section.

## Entity Framework 6

Skipping this section as I’m not using Entity Framework.

## SignalR

### Web sockets

Rather than a request / response model, SignalR uses web sockets, a persistent bidirectional TCP connection which both client and server can use to send each other unsolicited messages.

var socket = new WebSocket(“ws://echo.websocket.org”);

// once the connection opens…

socket.send(“Hello there!”);

// server sent me a message…

socket.onmessage = function(event) {

alert(“I got data! “ + event.data);

}

SignalR allows you to use web socket functionality even in older browsers which don’t natively support web sockets.

### SignalR

Nuget package name: Microsoft.AspNet.SignalR

SignalR can try a number of different connection types to provide web socket-like functionality:

* Tries web sockets first as that’s the most efficient, but not supported by all browsers
* Server Sent Events (SSE) – a HTML5 API which allows a server to push data to a client over HTTP
* Forever Frame – some javascript on the client creates an iframe and maintains an open connection with the server using that iframe
* Long polling – client repeatedly polls the server

IIS 8 and above on Windows Server 2012 is required to support SignalR. IIS Express also supports it.

The server has a number of SignalR hubs and the client connects to a hub using a jQuery plugin, so jQuery is also required.

Example server code:

public class ChatHub : Hub

{

public void Send(string name, string message)

{

// call the broadcastMessage method to update clients

Clients.All.broadcastMessage(name, message);

}

}

Example client code:

$(‘sendmessage’).click(function() {

// call the sent method on the hub

chat.server.send($(‘#displayname’).val(), $(‘#message’).val());

// clear text box and reset focus for next comment

$(‘#message’).val(‘’).focus();

});

The following demo intends to stream performance counter data from the server and display it on the client.

### Setting up

Create a new MVC ASP.net web application called PerfSurf. Change the authentication to Windows authentication. Install the nuget package Microsoft.AspNet.SignalR. The readme file indicates that we need to create a Startup class with a Configuration method:

public class Startup

{

public void Configuration(IAppBuilder app)

{

app.MapSignalR();

}

}

Create a project folder called Hubs, and within it create a hub class (web -> SignalR in the left hand navigation tree, and select the SignalR Hub Class template) called PerfHub.cs.

public class PerfHub : Hub

{

public void Hello()

{

Clients.All.hello(“Hi!”);

}

}

The one method in this class will simply broadcast the supplied string to all connected clients and pass it to a function on the client called hello. We now need some client side markup to use this:

<h1>SignalR Demo</h1>

<div class=”row”>

<div class=”col-md-4”>

<input type=”text” placeholder=”Message” data-bind=”value:message” />

<button class=”btn btn-primary” data-bind=”click:sendMessage”>Send</button>

<div>

<div class=”col-md-4”>

<div data-bind=”foreach:messages”>

<div data-bind=”text: $data”></div>

</div>

</div>

</div>

@section scripts{

<script src=”~/Scripts/jquery.signal-2.0.0.js”></script>

<script src=”~/SignalR/hubs”></script>

<script src=”~/Scripts/knockout-2.3.0.js”></script>

<script src=”~/Scripts/perfSurf.js”></script>

}

We also need to add the nuget package knockout to the project. This provides features such as client side databinding.

And we need some javascript called perfSurf.js

### Being chatty

This demo is just a simple chat application, we start building the performance counter app in the next section.

Rather than sending a hard-coded string, we want clients to be able to send their own messages, so we want to update the Send method on the PerfHub class:

public void Send(string message)

{

Clients.All.newMessage(Context.User.Identity.Name + “ says “ + message);

}

When this is built, the URL ~/SignalR/hubs returns some generated javascript which includes proxies for each hub in the project, which exposes each public method as a function.

The perfSurf.js file:

(function() {

var perfHub = $.connection.perfHub;

$.connection.hub.logging = true;

$.connection.hub.start();

perHug.client.newMessage = function(message) {

model.addMessage(message);

}

var Model = function() {

var self = this;

self.message = ko.observable(“”);

self.message = ko.observableArray();

}

Model.prototype = {

sendMessage: function() {

var self = this;

perfHub.server.send(self.message());

self.message(“”);

},

addMessage: function(message) {

var self = this;

self.messages.push(message);

}

};

var model = new Model();

$(function() {

ko.applyBindings(model);

});

}());

### PerfServer

First, run perfmon. This shows that for each counter, we need the category, the counter name and an instance of the counter. Now we need some helper classes. Create a folder called Counters containing a class called PerfCounterWrapper:

public class PerfCounterWrapper

{

public PerfCounterWrapper(string name, string category, string counter, string instance)

{

\_counter = new PerformanceCounter(category, counter, instance, readOnly: true);

Name = name;

}

public string Name { get; set; }

public float Value { get { return \_counter.NextValue(); } }

PerformanceCounter \_counter;

}

Add a class called PerfCounterService

public class PerfCounterService

{

List<PerfCounterWrapper> \_counters;

public PerfCounterService()

{

\_counters = new List<PerfCounterWrapper>();

\_counters.Add(new PerfCounterWrapper(“Processor”, “Processor”, “% Processor Time”, “\_Total”));

\_counters.Add(new PerfCounterWrapper(“Paging”, “Memory”, “Pages/sec”));

\_counters.Add(new PerfCounterWrapper(“Disk”, “PhysicalDisk”, “% Disk Time”, “\_Total”));

}

public dynamic GetResults()

{

Return \_counters.Select(c => new { name = c.Name, value = c.Value });

}

}

And the PerfHub class needs some new code:

public PerfHub()

{

StartCounterCollection();

}

private void StartCounterCollection()

{

var task = Task.Factory.StartNew(async () =>

{

var perfService = new PerfCounterService();

while (true)

{

var results = perfService.GetResults();

Clients.All.newCounters(results);

await Task.Delay(2000);

}

}, TaskCreationOptions.LongRunning);

}

### PerfClient

The client now needs some javascript in order to consume this performance counter data. Update perfSurf.js:

(function() {

var perfHub = $.connection.perfHub;

$.connection.hub.logging = true;

$.connection.hub.start();

perHub.client.newMessage = function(message) {

model.addMessage(message);

}

perfHub.client.newCounters = function (counters) {

model.addCounters(counters);

}

var ChartEntry = function(name) {

var self = this;

self.name = name;

self.chart = new SmoothieChart({ millisPerPixel: 50, labels: { fontSize:15 } });

self.timeSeries = new TimeSeries();

self.chart.addTimeSeries(sed.timeSeries, { linewidth:3, strokeStype:’#00ff00 });

}

ChartEntry.prototype = {

addValue: function(value) {

self.timeSeries.append(new Date().getTime(), value);

},

start: function() {

var self = this;

self.canvas = document.getElementById(self.name);

self.chart.streamTo(self.canvas);

}

}

var Model = function() {

var self = this;

self.message = ko.observable(“”);

self.message = ko.observableArray();

self.counters = ko.observableArray();

}

Model.prototype = {

addCounters: function(updatedCounters) {

var self = this;

$.each(updatedCounters, function(index, updateCounter) {

var entry = ko.utils.arrayFirst(self.counters(), function(counter) {

return counter.name = updatedCounter.name;

});

if (!entry) {

entry = new ChartEntry(updateCounter.name);

self.counters.push(entry);

entry.start();

}

entry.addValue(updateCounter.value);

});

}

sendMessage: function() {

var self = this;

perfHub.server.send(self.message());

self.message(“”);

},

addMessage: function(message) {

var self = this;

self.messages.push(message);

}

};

var model = new Model();

$(function() {

ko.applyBindings(model);

});

}());

This also requires smoothie charts, a javascript file which isn’t available as a nuget package, so it needs to be added to the project manually, and referenced from the view:

<script src=”~/Scripts/smoothie.js”></script>

And the view also needs a canvas for each performance counter:

<div class=”row” data-bind=”foreach:counters”>

<div class=”col-md-12”>

<h2 data-bind=”text:name”></h2>

<canvas width=”800” height=”100” data-bind=”attr:{‘id’: name}”></canvas>

</div>

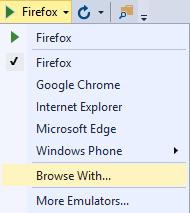
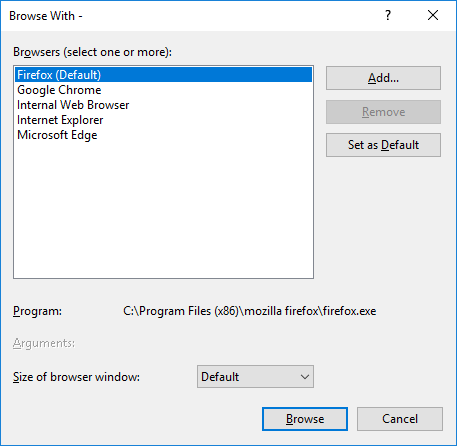
</div>

## Web developer tools and Visual Studio 2013

### Browser link

After running a web site without debugging (ctrl-F5), after making a change to the web project and rebuilding, you don’t need to switch back to the browser and refresh the page, instead you can use the Browser Link  button or ctrl-alt-enter to refresh the browser (or all linked browsers).

You can also launch multiple browsers when running the web site using the Browse With menu option:

Browser Link does this by making use of SignalR to set up a communications link between Visual Studio and the browser(s).

### Scaffolding

Scaffolded items are like file templates taken to the next level – after selecting a template, a wizard starts to prompt for additional information which is used to build some default code.

Adding new scaffolding is very difficult in 2013, but customising existing scaffolding can be done. Create a new folder in the project called CodeTemplates, copy in an existing template folder from Program Files\Microsoft Visual Studio 12.0\Common7\IDE\Extensions\Microsoft\Web\MVC\Scaffolding\Templates and modify it to suit your needs. Visual Studio will use any templates in your CodeTemplates folder before looking in its own templates folder.

The templates take the form of .t4 files, which are similar to Razor syntax.

### Web essentials overview

Web Essentials is an extension for Visual Studio.

Tools -> Extensions and updates -> Web Essentials 2013. Restart Visual Studio after installing.

After installing Web Essentials, there’ll be a Web Essentials node in the navigation tree of the application options dialogue.

This adds a new option into the Browser link dashboard – Design Mode. This effectively turns the browser into a design surface, so that you can edit its content, and the updates are automatically pushed back to the appropriate source file in your project.

Inspect mode allows you to hover over elements in the browser, and it highlights the corresponding lines in the source code in Visual Studio.

Application options allow you to turn on various errors and warnings for inefficient CSS or javascript code, or non-compliance with ECMA, and even suggest fixes for them using ctrl-full stop.

### Editor tricks

With Web Essentials installed, when editing HTML or Razor, put the cursor within an element and press alt-1 to select that element, and press alt-1 again to select its parent element and so on.

With an element selected (maybe using alt-1), press shift-alt-W to wrap that element in a new element.

Typing & opens an intellisense which lists all the available HTML entities (e.g. &amp;).

The first time you use an attribute (e.g. data-toggle), Visual Studio caches the intellisense for that attribute, making it available the next time you use that attribute in the same file.

In CSS files, with a colour value selected, ctrl-shift-up arrow or ctrl-shift-down arrow to make the colour lighter or darker.

### LESS

There’s a separate PluralSight course on LESS and SASS by Shaun Wildermuth.

New file template item for LESS stylesheets. LESS extends CSS.

#### Variables

LESS allows you to define variables, e.g.

@color: #f12345;

#header {

color: @color;

}

On saving this file, Visual Studio transforms it into a proper .css file dependent on your .less file, replacing any references to the variable with its value.

#### Mixins

Mixins allow you to embed one CSS class within another CSS class, and allows them to be parameterised. In this example, the .rounded-corners class is embedded within two other classes, with the radius as a parameter.

.rounded-corners (@radius: 5px) {

-webkit-border-radius: @radius;

-moz-border-radius: @radius;

-ms-border-radius: @radius;

-o-border-radius: @radius;

border-radius: @radius;

}

#header {

.rounded-corners;

}

#footer {

.rounded-corners(10px);

}

#### Nested selectors

Web Essentials allows selectors to be nested within .less files, e.g.

#header {

h1 {

font-size: 26px;

font-weight: bold;

}

p { font-size: 12px;

a { text-decoration: none;

&:hover {border-width: 1px }

}

}

}

LESS transforms these more terse definitions into the appropriate valid CSS definitions.

### Zen Coding

Zen Coding is a plugin for text editors for editing HTML. Web Essentials adds it to Visual Studio.

To get a <div id=”content”></div>, type div#content and then press tab.

To get a <div class=”content”></div>, type div.content and press tab.

To get a <div><nav></nav></div>, type div>nav and press tab.

And these shortcuts can be combined. Typing div#head>nav.menu and pressing tab gives you

<div> id=”head”>

<nav class=”menu”></nav>

</div>

Typing li\*5 and pressing tab gives you 5 empty <li></li> elements.

So you could do all of these things, and also give each list item its own unique ID, with a single line: div#head>nav.menu>ul>li#item$$\*5. (The number of $ indicates the number of digits to pad the number out to).

A final example, creating not only child elements but sibling elements: div.container>(header>nav>+(div.row>div.col-md-4\*3) gives you

<div class=”container”>

<header>

<nav></nav>

</header>

<div class=”row”>

<div class=”col-md-4”></div>

<div class=”col-md-4”></div>

<div class=”col-md-4”></div>

</div>

</div>

### Side Waffle

Side Waffle is another open source extension for Visual Studio. Get it from Tools -> Extensions -> Side Waffle Templates. Restart Visual Studio after installing.

#### Consumer of Side Waffle

Side Waffle adds a bunch of new item and project templates to Visual Studio, find them under Web -> Side Waffle. As new templates are released, Visual Studio will notify you of them.

#### Using Side Waffle to create new templates

First, ensure you have the Visual Studio 2013 SDK installed. Download the Side Waffle source code from github.com/LigerShark/side-waffle. This contains the TemplatePack project, which builds to the .vsix file which you install into Visual Studio as an extension.

To create a new template in the Side Waffle project, use the Extensibility -> SideWaffle -> SideWaffle Item Template template. This creates some .vstemplat- files in the project for different languages, rename the ones for languages you want to support, to .vstemplate.

### Azure tools

Azure tolls are already in Visual Studio 2013, no need to install any extensions. These allow you to work with any web sites, databases etc hosted in Azure.

# Bootstrap 3

Presenter: Shawn Wildermuth

URL: <https://app.pluralsight.com/library/courses/bootstrap-3>

## Getting started

### Adding bootstrap to a page

Add this to the head tag, before the site’s own CSS file, this allows the site’s CSS file to modify the styles in the bootstrap CSS file.

<link rel=”stylesheet” href=”css/bootstrap.min.css />

Add this to the end of the body tag. Scripts should be at the end of the file to allow page content to display without waiting for the scripts to download.

<script src=”js/jquery-2.0.3.min.js” />

<script src=”js/bootstrap.min.js” />

### Page container

Enclose the whole body in a div using the container class

<div class=”container”>

The container centres itself on the page with left and right margins for large displays, and on smaller displays resizes itself to fit the page width and takes up 100% of the screen width.

Alternatively if your page has a header, body and footer, you can enclose each of them in its own container.

### The grid system, col-xx-n and col-xx-offset-n classes

Bootstrap uses a grid which is 12 columns across. Use classes to control how many columns an element takes up. The class takes the form col-xx-n where xx is the size of screen (e.g. md for medium) and n is the number of columns to take up.

Offset an element by adding a col-xx-offset-n class, where xx is the size of display and n is the number of columns to offset the element by.

If a column’s size and offset would make it extend beyond the 12th column, it will wrap onto the next line.

### Display sizes and their abbreviations

|  |  |  |
| --- | --- | --- |
| Abbreviation | Size | Typical usage |
| lg | Large >= 1200px | Desktop PC |
| md | Medium >= 992px | Landscape tablet |
| sm | Small >= 768px | Portrait tablet |
| xs | Extra small < 768px | Phone |

Typically we’d only use the md class, unless we want to override the default bootstrap behaviour for a particular screen size, for example col-md-6 col-xs-12 means an element would take half the screen width except on the smallest screens where it would take the whole screen width.

### Rows

To force some content to appear above other content, enclose it in a <div class=”row”> element, because we can’t really nest a container inside another container.

### Images

To float text to the right of an image, give the image the pull-left class, and to float text to the left, give the image the pull-right class.

Give an image the img-thumbnail class to put a border and some padding between the image and any floated text. This will also make the image resize to fit the page width if it’s too wide to fit at its full size.

### Hide and show based on the display size

Use visible-xx and hidden-xx classes to control whether or not an element is displayed depending on the screen size. For example hidden-xs means the element will be displayed on all sizes but phone, and visible-md visible-lg means the element will be displayed on desktop and landscape tablets but not the smaller sizes.

### Your own theme

You can override bootstrap’s default fonts, colours, sizes etc in your own site’s CSS file, e.g. to change the background colour of a container

.container {

background-color: #f8f8f8

}

### Built-in and off-the-shelf themes

Use the navbar class for a navigation bar, with for example the navbar-default class to use the default theme for a navigation bar.

Use the btn class to style an element as a button.

Add a bootstrap theme CSS file to the page after the main bootstrap CSS file to apply that theme, e.g.

<link rel=”stylesheet” href=”css/bootstrap.min.css />

<link rel=”stylesheet” href=”css/bootstrap-theme.min.css />

Free themes for bootstrap are available from bootswatch.com, which can replace the bootstrap-theme.min.css in the example above.

### Customising bootstrap

At getbootstrap.com you can download a custom version of the bootstrap files by selecting only the features you want to use in your site, thereby reducing the size of CSS and JS files that the client needs to download. However you need to repeat this process each time a new version of bootstrap is released.

## Bootstrap 3 basics

### Typography classes

Use the lead class to emphasise a paragraph, e.g. the opening paragraph of a page.

Use the text-muted, text-primary, text-success, text-warning classes and more to change the colour of text. Other element types have similar classes, e.g. panel-warning, btn-primary.

### Buttons

Bootstrap won’t style an element based on just its type, you need to give it a class, e.g. to use bootstrap’s default styling for a button

<button class=”btn”>Some text</button>

And a further class to style the button a particular way, e.g.

<button class=”btn btn-primary”>Some text</button>

And a further class to change the size of the button, e.g.

<button class=”btn btn-primary btn-sm”>Some text</button>

The same size abbreviations used in the grid system are used here, i.e. lg, md, sm and xs.

You can also style an element to appear as a button even if its element type is not a button, e.g.

<a href=”#” class=”btn btn-success btn-sm>Some text</a>

This will appear and be styled as if it were a button but will behave like an anchor/hyperlink.

### Icons

The available icons are documented at getbootstrap.com/components/#glyphicons.

Icons can be included in your page using the glyphicon class followed by the class for the specific icon, e.g.

<button><span class=”glyphicon glyphicon-earphone”>Some text</span></button>

### Navs and navbars

Wrap your navigation bar in a div with the classes navbar navbar-default. The header of the navbar which belongs on the left should be wrapped in a div with the class navbar-header. Wrap navigation elements within the navbar in a ul with the nav and navbar-nav classes. Add the navbar-right class to push the elements to the right hand side of the page. Use the nav class on each of the list items within the list.

Add the active class to the nav class to indicate that a navigation element is active, i.e. it represents the page we’re currently on.

#### Collapsing navigation elements

Wrap the ul containing your navigation elements in a div with the navbar-collapse and collapse classes to not display them if the screen isn’t wide enough. Add a button to the navbar-header div with the classes btn btn-success, and with a data-toggle attribute of collapse and a data-target attribute of .navbar-collapse (the class of the div containing the unordered list of navigation elements). This can be used to show the navigation elements if they are collapsed because the screen is too small.

#### Make your page content start after the navigation bar

Because the navigation bar is floating over the page content rather than appearing above it, force the page content to start below the navigation bar by adding something like this to your site CSS file

body {

margin-top: 70px;

}

Adjust the margin size to suit the height of the navigation bar’s contents.

### Tables

Add the table class to a table for basic formatting, including making the table the width of its container.

Add the table-bordered class to add borders.

Add the table-striped class to set different coloured backgrounds for alternating rows.

Add the table-hover class to highlight the row currently under the mouse pointer.

Add the table-condensed class to reduce the padding of the cells in the table.

Wrap the table in a div with the table-responsive class to prevent the contents of the cells from wrapping if the screen is too narrow to display them, instead the table gains a horizontal scroll bar.

### Forms

#### Vertical form

Format an input element and its label together like this

<div class=”form-group”>

<label for=”nameInput”>Your name</label>

<input type=”text”

name=”nameInput”

class=”form-control”

placeholder=”e.g. your name” />

</div>

#### Inline form

Add the class form-inline to your form to position each of the input groups horizontally, only wrapping onto the next line when they run out of container width.

#### Horizontal form

Add the class form-horizontal to your form to and add classes such as col-xx-n to the labels and input elements to line the inputs up vertically.

## Components

### Page header and breadcrumbs

Important information about your page can be included in a div with the class page-header to make it stand out from the page content which follows it.

You can include breadcrumbs with markup similar to

<ol class=”breadcrumb”>

<li><a href=”/”>Home”</a></li>

<li class=”active>About</li>

</ol>

### Button groups

<div class=”btn-group” data-toggle=”buttons”>

<label class=”btn btn-success active”><input type=”radio”>one</label>

<label class=”btn btn-success”><input type=”radio”>two</label>

<label class=”btn btn-success”><input type=”radio”>three</label>

</div>

### Dropdowns

<div class=”dropdown”>

<button class=”btn” data-toggle=”dropdown”>show me</button>

<ul class=”dropdown-menu”>

<li><a href=”#” tabindex=”-1”>one</a></li>

<li><a href=”#” tabindex=”-1”>two</a></li>

<li class=”divider” />

<li class=”disabled”><a href=”#” tabindex=”-1”>three</a></li>

</ul>

</div>

### Button dropdowns

Or you can make the dropdown look like a button:

<div class=”btn-group”>

<button class=”bt btn-success” data-toggle=”dropdown”>

Other <span class=”caret”></span>

</button>

<ul class=”dropdown-menu”>

<li><a href=”#” tabindex=”-1”>one</a></li>

<li><a href=”#” tabindex=”-1”>two</a></li>

<li class=”divider” />

<li class=”disabled”><a href=”#” tabindex=”-1”>three</a></li>

</ul>

</div>

### Using a dropdown for a select

<div class=”dropdown”>

<button class=”btn btn-success” id=”pickButton”>

Pick one...

</button>

<button class=”btn btn-success” data-toggle=”dropdown”>

<span class=”caret”></span>

</button>

<ul class=”dropdown-menu” id=”reasonDropdown”>

<li><a href=”#” tabindex=”-1”>one</a></li>

<li><a href=”#” tabindex=”-1”>two</a></li>

<li class=”divider” />

<li class=”disabled”><a href=”#” tabindex=”-1”>three</a></li>

</ul>

</div>

To change the text of the button to the selected item, we also need some javascript:

(function () {

“use strict”;

var $pickButton = $(“#pickButton”);

$(“#reasonDropdown li a”).on(“click”, function () {

var reason = $(this).text();

$pickButton.text(reason);

});

})();

### Input groups

<div class=”input-group”>

<span class=”input-group-addon”>@</span>

<input type=”email” name=”emailInput” class=”form-control” placeholder=”your email address” />

</div>

Input-group-addon spans can be added before or after the input element, or both.

### Pagination

Only provides navigation elements, doesn’t actually split data into pages.

<ul class=”pager”>

<li class=”previous”><a href=”#”>&larr; Previous</a></li>

<li class=”next”><a href=”#”>Next &rarr;</a></li>

</ul>

<ul class=”pagination”>

<li class=”disabled”><a href=”#”>&laquo;</a></li>

<li class=”active”><a href=”#”>1</a></li>

<li><a href=”#”>2</a></li>

<li><a href=”#”>3</a></li>

<li><a href=”#”>&raquo;</a></li>

</ul>

### Thumbnails

<div class="col-md-4 col-sm-6">

<div class=”thumbnail”>

<a href=”#”>

<img src=”/images/1.jpg” class=”image-responsive” alt=”1”/>

</a>

<div class=”caption”>caption text here</div>

</div>

</div>

### Panels

<div class=”panel panel-default”>

<div class=”panel-heading”>

<h2 class=”panel-title”>Panel heading goes here</h2>

</div>

<div class=”panel-body”>Panel content goes here</div>

<div class=”panel-footer”>Panel footer goes here</div>

</div>

### Wells

<div class-“well well-sm”>

Content here

</div>

## Bootstrap plugins

### Collapse

<ul class=”nav nav-pills navbar-default>

<li><a href=”#firstThingToCollapse” data-toggle=”collapse”>One</a></li>

<li><a href=”#secondThingToCollapse” data-toggle=”collapse”>Two</a></li>

</ul>

…

<div id=”firstThingToCollapse” class=”collapse in”>

<!—this section will be shown initially because of the in class 🡪

Collapsible content here.

</div>

…

<div id=”secondThingToCollapse” class=”collapse”>

Collapsible content here.

</div>

### Accordion

<div id=”accordion” class=”panel-group”>

<div class=”panel panel-info”>

<div class=”panel-heading>

<div class=”panel-title”>

<a href=”#first” data-toggle=”collapse” data-parent=”#accordion”>

First

</a>

</div>

</div>

<div class=”panel-collapse collapse” id=”first”>

<div class=”panel-body”>

Content goes here.

</div>

</div>

</div>

</div>

### Modal

#### Modal markup

<div class=”modal fade” id=”sentDialog” tabindex=”-1”>

<div class=”modal-dialog”>

<div class=”modal-content”>

<div class=”modal-header”>

<a href=”#” class=”close” data=dismiss=”modal”>&times;</a>

<h4>Thanks for clicking</h4>

</div>

<div class=”modal-body”>

<p>This form doesn’t really work</p>

</div>

<div class=”modal-footer”>

<button class=”btn btn-success” data-dismiss=”modal”>Close</button>

</div>

</div>

</div>

</div>

All the above won’t be visible on the page until it’s invoked, with something like

<a href=”#sentDialog” class=”btn btn-info” data-toggle=”modal”>

Show dialog

</a>

#### Modal events

To invoke a modal form from a form post, give the form a name, e.g.

<form id=”contactForm”>

And then we need some javascript

var $sentDialog = $(“#sentDialog”);

$(“#contactForm”).on(“submit”, function() {

$sentDialog.modal(‘show’);

return false; // don’t submit the form

});

$sentDialog.on(“hidden.bs.modal”, function() {

Alert(“close”);

});

### Tab

Tab buttons:

<ul class=”nav nav-tabs nav-justified”>

<li><a href-“#formTab” data-toggle=”tab” active>Contact form</a></li>

<li><a href-“#addressTab data-toggle=”tab””>Address</a></li>

</ul>

Tab contents:

<div class=”tab-content”>

<div class=”well tab-pane active” id=”formTab>

Content here

</div>

<div class=”well tab-pane” id=”addressTab”>

Content here

</div>

</div>

### Tooltip

<input type=”submit” value=”Submit” class=”btn btn-success” data-toggle=”tooltip” title=”Press here” data-placement=”right” data-delay=”500” />

Can also add a data-html=”true” attribute to expect the title attribute’s value to be HTML markup.

And some javascript to enable tooltips for that element:

$(“#contactForm input[type=submit]”).toolip();

### Alert

<div class=”alert alert-warning collapse”>

<a href=”#” data-dismiss=”alert” class=”close”>&times;</a>

<p>This doesn’t work, it’s not real!</p>

</div>

And some javascript:

var $sentAlert = $(“#sentAlert”);

$sentDialog.on(“hidden.bs.modal”, function() {

$(“#sentAlert”).show();

});

// to only hide the alert when it’s dismissed, rather than removing it from

// the DOM entirely

$sentAlert.on(“close.br.alert”, function() {

$sentAlert.hide();

return false;

});

### Carousel

#### Basic carousel

<div>

<div class=”row visible-md visible-lg”>

<div class=”col-md-8 col-md-offset-2”> <!-- to size it-->

<div class=”carousel slide” id=”theCarousel” data-interval=”2000”>

<div class=” carousel-inner”>

<div class=”item active”>

<img src=”1.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”2.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”3.jpg” class=”img-responsive” />

</div>

</div>

</div>

</div>

</div>

</div>

And some javascript:

$(“#theCarousel”).carousel();

#### Carousel indicators

<div>

<div class=”row visible-md visible-lg”>

<div class=”col-md-8 col-md-offset-2”> <!-- to size it-->

<div class=”carousel slide” id=”theCarousel” data-interval=”2000”>

<ol class=”carousel-indicators”>

<li data-target=”theCarousel”

data-slide-to=”0”

class=”active”></li>

<li data-target=”theCarousel” data-slide-to=”1”></li>

<li data-target=”theCarousel” data-slide-to=”2”></li>

<li data-target=”theCarousel” data-slide-to=”3”></li>

</ol>

<div class=” carousel-inner”>

<div class=”item active”>

<img src=”1.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”2.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”3.jpg” class=”img-responsive” />

</div>

</div>

</div>

</div>

</div>

</div>

#### Carousel navigation

<div>

<div class=”row visible-md visible-lg”>

<div class=”col-md-8 col-md-offset-2”> <!-- to size it-->

<div class=”carousel slide” id=”theCarousel” data-interval=”2000”>

<ol class=”carousel-indicators”>

<li data-target=”theCarousel”

data-slide-to=”0”

class=”active”></li>

<li data-target=”theCarousel” data-slide-to=”1”></li>

<li data-target=”theCarousel” data-slide-to=”2”></li>

<li data-target=”theCarousel” data-slide-to=”3”></li>

</ol>

<div class=” carousel-inner”>

<div class=”item active”>

<img src=”1.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”2.jpg” class=”img-responsive” />

</div>

<div class=”item”>

<img src=”3.jpg” class=”img-responsive” />

</div>

</div>

<a href=”#theCarousel”

class=”carousel-control left”

data-slide=”prev”>

<span class=”icon-prev></span>

</a>

<a href=”#theCarousel”

class=”carousel-control right”

data-slide=”next”>

<span class=”icon-next></span>

</a>

</div>

</div>

</div>

</div>

#### Carousel captions

<div>

<div class=”row visible-md visible-lg”>

<div class=”col-md-8 col-md-offset-2”> <!-- to size it-->

<div class=”carousel slide” id=”theCarousel” data-interval=”2000”>

<ol class=”carousel-indicators”>

<li data-target=”theCarousel”

data-slide-to=”0”

class=”active”></li>

<li data-target=”theCarousel” data-slide-to=”1”></li>

<li data-target=”theCarousel” data-slide-to=”2”></li>

<li data-target=”theCarousel” data-slide-to=”3”></li>

</ol>

<div class=” carousel-inner”>

<div class=”item active”>

<img src=”1.jpg” class=”img-responsive” />

<div class=”carousel-caption”>

<h4>Bowling</h4>

<p>Bowling is fun</p>

</div>

</div>

<div class=”item”>

<img src=”2.jpg” class=”img-responsive” />

<div class=”carousel-caption”>

<h4>The Dude</h4>

<p>Bowling is fun</p>

</div>

</div>

<div class=”item”>

<img src=”3.jpg” class=”img-responsive” />

<div class=”carousel-caption”>

<h4>Awards</h4>

<p>Bowling is fun</p>

</div>

</div>

</div>

<a href=”#theCarousel”

class=”carousel-control left”

data-slide=”prev”>

<span class=”icon-prev></span>

</a>

<a href=”#theCarousel”

class=”carousel-control right”

data-slide=”next”>

<span class=”icon-next></span>

</a>

</div>

</div>

</div>

</div>

# jQuery Fundamentals

Presenter: Dan Wahlin

URL: <https://app.pluralsight.com/library/courses/jquery-fundamentals>

## jQuery Fundamentals

### Why use jQuery?

Single javascript file, cross browser, Selectors – selecting a DOM element and manipulating it, Events, Ajax, Plugins.

Locate elements with a specific class, apply styles to multiple elements, handle events in a cross-browser manner

### Getting started with jQuery

Download the jQuery script from <http://jquery.com>

jQuery 1.x to support IE6 to 8, jQuery 2.x if you don’t need to support them

Reference the script:

<head>

<script type=”text/javascript” src=”jquery.js”></script>

</head>

### Using a CDN

You can reference the jquery file from a CDN instead of hosting it locally. Delivers caching benefits because the same script may be referenced from the same CDN from multiple domains. Also allows the script to be downloaded in parallel with content from your own domain.

To fallback to a local copy of the script if the CDN can’t be reached:

<script type=”text/javascript” src=”http:mycdn.org/whatever/jquery.js”></script>

<script>

window.jQuery ||document.write(‘<script src=”jquery.js”><\/script>’)

</script>

### Detecting when a page has loaded - Using the jQuery ready function

$ is an alias for the word jQuery.

Use $(document).ready() to detect when the DOM of a page (but not necessarily its images etc) have loaded.

<script type=”text/javascript”>

$(document).ready(function(){

// perform action here

});

</script>

Example of detecting when the page has loaded, locating an element with a specific ID and replacing the content of that element:

<html>

<head>

</head>

<body>

Hello jquery

<div id="myContent">

My jQuery enabled page!

</div>

<script type="text/javascript" src="jquery-3.2.1.js"></script>

<script type="text/javascript">

// using javascript to tell whether the page (excluding images etc) has been loaded

window.onload = function() {

alert('Window loaded');

}

// using jQuery to tell if the DOM has loaded

$(document).ready(function() {

// alert('DOM loaded');

// the # means do a document.getElementById

// the .html replaces the content with different HTML

$('#myContent').html('Hello world from .html');

});

</script>

</body>

</html>

### jQuery documentation

jquery.com -> API documentation

## Using jQuery selectors

$(selectorExpression) is the same as jQuery(selectorExpression)

### Selecting nodes by tag name

$(‘p’) selects all <p> elements

$(‘a’) selects all <a> elements

$(‘p,a,span’) selects all <p>, <a> and <span> elements

$(‘table tr’) selects all <tr> elements that are descendants of a <table> element. Descendants are children, grandchildren etc.

And to manipulate the selected elements, something like $(‘div’).css(‘background-color’, ‘Green’);, or

$(‘div’).each(function() {

alert($(this).html());

});

If the selector returns multiple elements, jQuery automatically iterates through all of them.

### Selecting nodes by ID

$(‘#myID’) selects <p id=”myID”> element.

Most efficient, doesn’t need to scan the whole DOM.

### Selecting nodes by class name

$(‘.myClass’) selects <p class=”myClass”> element. Less efficient, scans the whole DOM to find elements with the supplied class name.

$(‘.BlueDiv,.RedDiv’) selects all elements with the class BlueDiv or RedDiv.

$(‘a.myClass’) selects all <a> elements with the class myClass. More efficient than just looking for the class because it scans only the elements of the supplied type, not the whole DOM.

### Selecting nodes by attribute value

$(‘a[title]’) selects all <a> elements that have a title attribute.

$(‘a[title=”ProgrammingInfo”]’) selects all <a> elements that have a “ProgrammingInfo” title attribute value. This is case-sensitive.

### Selecting input nodes

$(‘:input’) selects all input elements including input, select, textarea, button, image, radio and more.

$(‘:input[type=”radio”]’) selects all radio buttons on the page, but is it the most efficient way? No, $(‘input[type=”radio”]’) gives the same result but only needs to scan the input elements, not the whole DOM.

To get the value of an input element:

var inputs = $(‘:input’);

alert($(inputs[1]).val());

The array is 1-based.

### Contains, starts with, odd, even etc

$(‘div:contains(“pluralsight”)’) selects <div> elements that contain the text pluralsight (case=sensitive).

$(‘tr:odd’) and $(‘tr:even’) select odd or even rows from tables. Index is 0 based.

$(‘*element*:first-child’) selects the first child of every element group, e.g. $(‘span:first-child’) would select the highlighted elements in

<div>

<span>First Child, first group</span>

<span>Second Child, first group</span>

</div>

<div>

<span>First Child, second group</span>

<span>Second Child, second group</span>

</div>

[*attribute*^=”*value*”] selects all elements with an attribute whose value begins with the supplied value, e.g. $(‘input[value^=”Events”]’).

[*attribute*$=”*value*”] selects all elements with an attribute whose value ends with the supplied value, e.g. $(‘input[value$=”Events”]’).

[attribute\*=”value”] selects all elements with an attribute whose value contains the supplied value, e.g. $(‘input[value\*=”Events”]’).

There are more, see jQuery documentation.

<http://codylindley.com/jqueryselectors> is good for testing selectors.

## Interacting with the DOM

### Iterating through nodes

.each(function(index, Element)) is used to iterate through jQuery objects, e.g. to iterate through each <div> and return its index number and text:

$(‘div’).each(function(index) {

alert(index + “=” + $(this).text());

});

Or to achieve the same, but accepting the current element of the iteration as an extra parameter of the function (not a common way of doing it as it’s more typing):

$(‘div’).each(function(index, elem) {

alert(index + “=” + $(elem).text());

});

For efficiency, if you reference the same element every time through a loop, avoid using the selector within the loop, instead cache the jQuery object returned by the selector into a variable before starting the loop.

### Modifying DOM object properties

The this.*propertyName* statement can be used to modify a DOM object’s properties directly, e.g. to iterate through each <div> and modify its title attribute, adding the attribute if it doesn’t exist already:

$(‘div’.each(function(i) {

this.title=”My Index = “ + i;

});

### Modifying jQuery object attributes

#### Modify a single attribute

Get attributes of jQuery objects using attr(*attributeName*), e.g. to retrieve the value of the title attribute:

var val=$(‘#CustomerDiv.attr(‘title);

Modify attributes of jQuery objects using attr(*attributeName*, *newValue*), e.g. to change the title attribute of images:

$(‘img’).attr(‘title’, ‘My image title);

#### Modify multiple attributes using JSON

Modify multiple attributes using a JSON object containing name/value pairs:

$(‘img’).attr({

title: ‘My image title’,

style: ‘border: 2px solid black;’

});

A more complex example of JSON with a nested object (complex property), not directly related to jQuery:

{

FirstName: ‘John’,

LastName: ‘Doe’,

Address: {

Street: ‘1234 Anywhere St.’,

City: ‘Phoenix’,

State: ‘AZ’,

ZipCode: 85249

}

}

#### Modify multiple attributes using chaining

Because functions like attr and css return the modified jQuery object you can chain calls to them together fluent API style, e.g.:

$(‘div.BlueDiv, div.RedDiv’)

.attr(‘title’, ‘Some title’)

.css(‘background-color’: ‘yellow’)

.css(‘color’: ‘black’)

.css(‘font-size’, ‘20pt’);

Line breaks aren’t required but can make the code easier to read.

### Adding and removing nodes

Remember, if you manipulate the same element more than once, select it only once and cache the returned jQuery object for efficiency.

For more functions, see docs.jquery.com and select “manipulation” from the left hand menu.

#### .append() and .appendTo()

Adds a child node as the final child of selected elements. The following examples both add a <span> with the inner text “ (office)” to any elements with the class name officePhone.

$(‘<span> (office)</span>’).appendTo(‘.officePhone’);

$(‘.officePhone’).append(‘<span> (office)</span>’);

#### .prepend() and .prependTo()

Adds a child node as the first child of selected elements. The following examples both add a <span> with the inner text “Phone: ” to any elements with the class name officePhone.

$(‘<span>Phone: </span>’).prependTo(‘.officePhone’);

$(‘.officePhone’).prepend(‘<span>Phone: </span>’);

#### .wrap()

<div class=”state”>Arizona</div>

$(‘.state’).wrap(‘<div class=”US\_State” />

Results in:

<div class=”US\_State”>

<div class=”state”>Arizona</div>

</div>

Note that the element passed to the wrap method must be closed, either explicitly or self-closing.

#### .remove()

To remove elements with the class names phone or location:

$(‘.phone, .location’).remove();

Other selectors can be used, not just selecting by class.

### Modifying styles

Use the .css() function to change an object’s style, e.g.

$(‘div’).css(‘color’, ‘red’);

The .css() function can also accept JSON objects, e.g.

$(‘div’).css({

‘color: ‘#ccc’,

‘font-weight’: ‘bold’

});

### Modifying classes

#### .addClass()

.addClass() adds one or more class names to the class attribute of elements, e.g.

$(‘p’).addClass(‘classOne’);

$(‘p’).addClass(‘classOne classTwo’);

#### .hasClass()

.hasClass() returns true if the selected element has a matching class that is specified, e.g. to do something if there is any <p> element with the class name “className”:

if($(’p’).hasClass(‘className’)) {

// do something

}

#### .removeClass()

.removeClass() removes one or more classes or all classes, e.g. to remove two classes:

$(‘p’).removeClass(‘classOne classTwo’);

Or to remove all class attributes:

$(‘p’).removeClass();

#### .toggleClass()

.toggleClass() alternates adding or removing a class based on the current presence or absence of the class, e.g.

$(‘#phoneDetails’).toggleClass(‘highlight’);

To highlight an input when it receives the focus and un-highlight it when it loses focus:

<input id=”something” onfocus=”FocusBlur(this)” onblur=”FocusBlur(this) />

And some javascript:

function FocusBlur(tb) {

$(tb).toggleClass(‘Highlight’);

}

NB this is dependent on the class being present or absent correctly on the element depending on whether it has the focus at page load.

## Handling events

See http://api.jquery.com/category/events

### Handling events

#### Anonymous function

How to handle a click event with an anonymous function:

$(‘#myID’).click(function() {

alert(‘The element myID was clicked’);

});

#### Raising an event

How to raise a click event on another object:

$(‘#otherID’).click(function() {

$(‘#myID’).click();

});

#### Named event handler

How to handle a click event with a named function, the old fashioned way:

<input type=”button” onclick=”SubmitButton\_Click()” />

function SubmitButton\_Click() {

alert(‘Clicked Button’);

}

#### Register event handlers in document ready

How to register a click event handler in document.ready. This may be preferable to the old fashioned way because all the event handlers are set in one place.

$(document).ready(function() {

WireEvents();

}

function WireEvents() {

$(‘#SubmitButton’).click(function() {

alert(‘Clicked Button’);

});

}

#### .change()

Handling a drop-down list value being selected. Also works for text boxes and text areas:

$(‘#StatesSelect’).change(function() {

alert($(this).val());

});

#### .mouseenter() and .mouseleave()

Handling the mouse entering and leaving a control:

$(‘#MyDiv’).mouseenter(function() {

Toggle(this);

$(div).css(‘cursor’,’ ‘pointer’);

})

.mouseleave(function() {

Toggle(this);

});

function Toggle(div) {

$(div).toggleClass(‘Highlight’);

}

#### Passing the event to the handler

Passing the event to an event handler and getting the target element of the event (note the target is a raw DOM object, not a jQuery object):

$(‘#MyDiv’).mouseup(function(e) {

alert(e.target); // gets the element which raised the event

$(this).text(‘X: ‘ + e.pageX + ‘ Y: ‘ + e.pageY);

});

### Binding to events

.on() and .off() are newer versions of .bind() and .unbind(), which are deprecated since 1.7.

#### .on()

.on(*eventType*, *handler(eventObject)*) attaches a handler to an event for the selected element(s).

$(‘#myDiv’).on(‘click’, function() {

// do something

});

Binding to multiple events with .on(). Event names to bind are separated with a space.

$(‘#myDiv’).on(‘mouseenter mouseleave’, function() {

$(this).toggleClass(‘entered’);

});

#### .off()

.off(*event*) removes a handler previously attached to an element.

Removing all event handlers from an element:

$(‘#test’).off();

Removing handlers for a specific event:

$(‘#test’).off(‘click’);

### .live(), .delegate() and .on()

Allow new DOM elements to automatically be “attached” to an event handler, which is actually attached to a parent element, to avoid having many individual event handler registrations. .live(), .die(), .delegate() and .undelegate() are all deprecated – use .on() and .off() instead.

#### .live() and .die()

.live() and .die() were removed in jQuery 1.9.

Registering event handlers using .live(). Events are handled at the document level by default.

$(‘.someClass’).live(‘click’, someFunction);

.die() removes event handlers registered by .live().

$(‘.someClass’).die(‘click’, someFunction);

#### .delegate() and .undelegate()

Replacement for .live() and .die(), introduced in jQuery 1.4. A context object (#Divs in the sample below) handles events by default rather than the document object. Works even when new objects are added into the DOM.

$(‘#Divs’).delegate(‘div’, ‘click’, someFunction);

Remove .delegate() event handlers using .undelegate().

$(‘#Divs’).undelegate();

#### .on()

.on() is a new replacement for .bind(), .delegate() and .live(). In the following example the event handler is registered on the <tbody> but for the <tr>s within it, so it’s a single registration.

$(‘#myTable tbody’).on(‘click’, ‘tr’, function(event) {

alert(‘Row was clicked and event bubbled up’);

});

#### Using .on() with a map

$(‘#myTable tr’).on({

mouseenter: function() {

$(this).addClass((‘over’);

},

mouseleave: function() {

$(this).removeClass(‘out’);

}

});

#### Event handlers acting on elements added at runtime

var tbody = “(‘#customers tbody’);

$(‘#addRow’).on(‘click’, function() {

tbody.append(‘<tr><td>Jane</td><td>Doe</td></tr>’);

}):

tbody.on(‘click’, ‘td’, function() {

alert($(this).text));

}):

### .hover() and .toggle()

#### .hover(handlerIn, handlerOut)

handlerIn is equivalent to mouseenter and handlerOut is equivalent to mouseleave.

$(‘#target’).hover(

function() {

$(this).css(‘background-color’, ‘#00ff99’);

},

function() {

$(this).css(‘background-color’, ‘#ffffff’);

}

);

#### .hover(handlerInOut)

This overload can be used with toggle methods, e.g.

$(‘#target’).hover(function() {

$(this).toggleClass(‘over’);

});

#### .toggle()

Similar to .hover() but it acts on mouse clicks. This can accept as many parameters as you want –the first click is handled by the first function, the second click by the second function, the third by the third and so on until we run out of functions, at which point it goes back to the first function.

$(‘#myTable tr’).toggle(

function() {

$(this).css(‘background-color’, ‘#efefef’);

},

function() {

$(this).css(‘background-color’, ‘#fff’);

},

function() {

$(this).css(‘background-color’, ‘Yellow’);

}

);

## Working with Ajax features

### jQuery Ajax functions

$(*selector*).load() loads HTML data from the server.

$.get() and $.post() gets/posts raw data from the server.

$.getJSON() gets/posts JSON data.

$.ajax() provides core functionality.

### Loading HTML content from the server

$(*selector*).load(*url*, *data*, *callback*) allows HTML content to be loaded from a server and added into a DOM object.

#### .load(url)

Put the content of helpDetails.html into the div with ID myDiv:

$(document).ready(function() {

$(‘#helpButton’).click(function() {

$(‘#myDiv’).load(‘helpDetails.html’);

}):

}):

Put the content of the element with the ID mainTOC from helpDetails.html into the element in the current document with the ID myDiv.

$(‘#myDiv’).load(‘helpDetails.html #mainTOC’);

#### .load(url, data)

The data parameter is a JSON object to pass to the page being loaded.

$(‘#myDiv’).load(‘GetCustomers.aspx’, {PageSize: 25});

#### .load(url, callback)

The callback is a function to call when a response is received from the server, which accepts as parameters the HTTP response, status text and the XML HTTP request.

$(‘#outputDiv’).load(‘NotFound.html’, function (response, status, xhr) {

if (status == “error”) {

alert(xhr.statusText);

}

}):

### Making GET requests

$.get(*url*, *data*, *callback*, *datatype*) retrieves data from a server. .load() usually makes a GET request unless you pass it data, in which case it makes a POST request, so .get() is a way of passing in data but still making a GET request. .get() can also return data of other types than HTML.

#### $.get(url, callback)

$.get(‘helpDetails.html’, function (data) {

$(‘#OutputDiv’).html(data);

});

#### $.get(url, data, callback, datatype)

$.get(‘CustomerJson.aspx, { id: 5 }, function(data) {

alert(data.FirstName);

}, ‘json’);

#### $.getJSON()

$.getJSON(*url*, *data*, *callback*) can retrieve JSON data from a server.

$.getJSON(‘CustomerJson.aspx’, {id: 1}, function(data) {

alert(data.FirstName + ‘ ‘ + data.LastName);

});

### Making POST requests

$.post(*url*, *data*, *callback*, *datatype*) can post data to a server and retrieve results/

$.post(‘GetCustomers.aspx’, {PageSize: 15}, function(data) {

$(‘#outputDiv’).html(data);

});

.post() can also be used to interact with an Ajax-enabled WCF service:

$.post(‘CustomerService.svc/GetCustomers’, null, function(data) {

var cust = data.d[0];

alert(cust.FirstName + ‘ ‘ + cust.LastName);

}, ‘json’);

When calling a WCF service, the data returned is in the data.d[0] array element rather than in the data variable.

#### Fiddler

Fiddler is a HTTP proxy which logs HTTP traffic to and from your computer and can be used to debug GET and POST operations if they’re not behaving as expected. Get it from fiddler2.com.

When debugging a locally hosted site, Fiddler doesn’t intercept URLs like localhost:12345, instead, add a . after localhost, e.g. localhost.:12345.

### Introduction to the ajax() function

All the functions we’ve looked at so far in this module use the ajax() function under the hood. Using it directly gives you a lot more control over things like

* contentType
* data
* datatype
* error
* success
* type (GET or POST)

The .ajax() function takes one parameter, which is a JSON object containing any or all of the above properties.

$.ajax({

url: ‘../CustomerService.svc/InsertCustomer’,

data: customer,

datatype: ‘json’,

success: function (data, status, xhr) {

alert(“Insert status: “ + data.d.Status ‘\n’ + data.d.Message);

},

error: function (xhr, status, error) {

Alert(‘Error occurred: ‘ + status);

}

});

There are more properties available, see <http://api.jquery.com/jQuery.ajax>

#### Demo

Get json2.js from [www.json.org](http://www.json.org) to make the JSON object available to convert JSON objects to strings.

<html>

<head>

<!-- omitted the bits to bring in jQuery etc -->

<script type=”text/javascript”>

$(document).ready(function() {

$(‘#MyButton’).click(function() {

var customer = ‘cust=’ +

JSON.stringify({

FirstName: $(‘#FirstNameTB’).val(),

LastName: $(‘#LastNameRB’).val()

});

$.ajax({

url: ‘../CustomerService.svc/InsertCustomer’,

data: customer,

datatype: ‘json’,

success: function (data, status, xhr) {

$(‘#OutputDiv’).html(‘insert status: ‘

+ data.d.Status + ‘<br/>’

+ data.d.Message);

},

error: function (xhr, status, error) {

alert(‘error occurred: ‘ + status);

}

});

});

});

</script>

</head>

<body>

First name: <input type=”text” id=”FirstNameTB” value=”John” />

<br/>

Last name: <input type=”text” id=”LastNameTB” value=”Doe” />

<br/>

<br/>

<button id=”MyButton”>Submit</button>

<div id=”OutputDiv”></div>

</body>

</html>

# jQuery in Depth

Presenter: Karl Swedberg

URL: <https://app.pluralsight.com/library/courses/jquery-in-depth>

## Events in jQuery

### Review of previous workshop

#### Loading events

$(document).ready() – when DOM has loaded but not referenced images

.load() – typically used with $(window), helpful with images but be careful, e.g. different browsers behave differently e.g. if an image isn’t found, or if it’s already cached

#### Multiple “ready” handlers

All are executed in the order defined. Can be defined after document is ready.

### Event binding and unbinding

#### Low level methods

.bind() adds an event listener, for a native event like click, or a custom event triggered by other code.

$(‘button’.bind(‘click’, function(event) {

//. Do something

});

#### Method context

The “this” keyword

$(‘button’).bind(‘click’, function(event) {

// “this” is the <button> DOM element

// this.className += ‘ clicked’; // adds the class regardless of whether it’s already on the element

// $(this} returns jQuery object with 1 element

// $(this).addClass(‘clicked’); // adds the class only if it’s not already there

});

### Triggering events

#### Low level methods

.trigger(‘*eventname*’) raises an event from code, works for native and custom events.

.click() can also raise events but .trigger() is probably clearer and is restricted to native events.

### Event propagation

AKA event bubbling. An event will be handled by the innermost element which is registered to handle it, and won’t be handled by any parent element even if registered to handle it.

#### Two event tutorials

How to ensure that new elements added to the DOM have events bound to them:

Using event delegation: <http://tinyurl.com/eventdelegation>

Using re-binding: <http://tinyurl.com/eveentrebinding>

#### Event delegation

Scales much better, makes it trivial to add and remove elements.

Uses event.target rather than this, doesn’t ask every element for events, asks a parent instead

Effort is moved from binding to handling.

### .live() and .die() methods

.live() and .die() were the original jQuery event binding methods. .live() binds to the document object rather than individual elements, so events are automatically handled for elements added to the document at runtime. Expensive because it requires every event to be checked for event handlers all the way up the DOM hierarchy, so don’t use them anymore, deprecated in jQuery 1.7.

### Event delegation

#### .delegate() and .undelegate() methods

.delegate() binds an event handler to jQuery set, triggered when element indicated in first argument is acted upon or one of its ancestors matches the first argument. More optimised delegation than.live().

.undelegate() unbinds an event handler.

$(‘#wrapper’).delegate(‘button’, ‘click’, function(event) {

// do something

});

// unbind click event for button

$(‘#wrapper’).undelegate(‘button’, ‘click);

Registers the event handler on the element with ID wrapper, as a delegate for all its child elements. The wrapper needs to be in the DOM when .delegate() is called, but event handling will still happen for elements added at runtime as descendants of the wrapper. Event handling won’t happen for elements added at runtime somewhere in the DOM outside the wrapper.

Use .delegate() and .undelegate() for event delegation in jQuery before 1.7.

#### .on() and .off() methods

New in jQuery 1.7, use these instead of .bind(), .live() and .delegate(). It’s better because it can be used to bind an event handler to a single element or a container to support event delegation.

To bind directly, $(‘whatever’).on(‘click’, clickListener);

For event delegation, $(‘#wrapper’).on(‘click’, ‘button’, clickListener);

### Event object

Normalised by jQuery to provide consistent information across browsers.

* event.target to access the element that triggered the event
* event.pageX/Y for the mouse position
* event.preventDefault() to prevent an event’s default action
* event.stopPropagation() to prevent an event from bubbling
* event.which to identify which key is pressed
* More event properties at <http://api.jquery.com/category/events/event-object>

Example: to prevent the default action from happening and to stop the event’s propagation up the DOM:

$(‘a.toggler’).on(‘click’, function(event) {

event.preventDefault();

event.stopPropagation();

// do something else with the event

});

We could instead return false to prevent the default action and stop propagation, but if a runtime error happened in our own handling of the event, the browser wouldn’t reach the return false statement and so the default action would still happen and the event would still be propagated.

### Event object properties

#### Example

identify the type of event being triggered in a handler which handles multiple event types

$(‘button’).on(‘mouseenter mouseleave’, function(event) {

var isEntered = event.type == ‘mouseenter’;

$(‘#something’).toggleClass(‘active-related’, isEntered);

if (isEntered) {

// do one thing

) else {

// do another

}

});

#### Key events

* event.which – key code noramlised by jQuery
* event.metaKey – command key on Mac, Ctrl on Windows
* event.altKey – Alt key on Windows, option key on Macc
* event.ctrlKey – Ctrl key everywhere
* event.ShiftKey

The event object also has a timestamp.

#### originalEvent property

The originalEvent property of the normalised event object returns the un-normalised, native event object which is wrapped in the normalised event object, complete with any properties or methods which the original event exposes but which aren’t included in the normalised event object. Example for touch events, which don’t exist in all browsers:

// An array of anonymous functions accessed through the sideSwipe variable

var sideSwipe = {

touchstart: function(event, firstTouch) {},

touchmove: function(event, firstTouch) {},

toughend: function(event) {}

};

if (document.createTouch) {

// then the browser supports touch events

$(‘body’).bind(‘touchstart touchmove touchend’, function(event) {

// use the original event

event = event.originalEvent;

// pass the first touch object as a second argument

var touch = event.targetTouches[0];

sideSwipe[event.type](event, touch);

});

})

#### Event mapping

A better way of implementing sideSwipe:

var sideSwipe = {

‘touchstart touchmove’: function(event) {

event = event.originalEvent;

var firstTouch = event.targetTouches[0];

if (event.type == ‘touchstart’) {

startcoords = {x: firstTouch.pageX, y: firstTouch.pageY};

return;

}

if (event.targetTouches.length === 1) {

event.preventDefault();

endcoords = {x: firstTouch.pageX, y: firstTouch.pageY);

} else {

endcoords = startcoords;

}

},

touchend: function(event) {

//

}

};

Registering the event is now simpler:

if (document.createTouch) {

$(document.body).on(sideSwipe);

}

### Advanced event handling tips

#### Namespacing

var $foo = $(‘#foo’);

var karClicks = function() { /\* do something \*/ };

$foo.on(‘mouseenter.karl’, function() { ‘/\* do something \*/’ };

$foo.on(‘click.karl’, karlClicks);

$foo.on(‘click’, function() { /\* do something \*/ };

$foo.off(‘.karl’); // stop doing .karl things

#### Custom events

$(‘#foo’).off(‘click.karl’).on(‘click.karl’, myFunction)

Useful if you’re writing a plugin and you don’t want external code accidentally binding your event to something and firing it when you’re not expecting it.

$(document).on(‘start.game’, myGameStartFn);

$(document).off(‘stop.game’, myGameStopFn);

$(document).on(‘updateScore.game’, function(event, data) {

$.ajax(‘/gameserver/’, data);

$(‘#scoreboard .score’).html(data.score);

});

$(document.on(updateBoard.game’, function() {

$(this).fadeOut(400, function() {

$(this).remove();

});

});

$(‘div.alien’).on(‘click’, function(event) {

score += 100;

killed: this.className,

name: $(this).data(‘name’)

};

$(event.target)

.trigger(‘updateBoard.game’)

.trigger(‘updateScore.game, (data));

if (score > 1000000) {

$(document).trigger(‘stop.game’);

)

## Dom manipulation

### Creating elements

$(‘<div></div>’) returns a new div element but doesn’t add it to the DOM. $(‘<div/>’) does the same. The parameter should always be well-formed HTML.

$(‘<div/>’, {

‘class’: ‘test’,

html: ‘<b>Click me!</b>’,

click: function() {

$(this).toggleClass(‘test’);

}):

}):

The second parameter is a JSON object representing the properties of the new element. The parameter name class needs to be surrounded by quotes because it’s a future reserved word in javascript however it’s being used as a property name here.

To add the returned element to the DOM, call one of its methods, e.g. .appendTo(‘body’);

This would also work but is less readable and depends on the developer to open and close all the elements correctly:

$(‘<div class=”test”><b>Click me!</b></div>’).on(‘click’, function() { /\* do something \*/ }.appendTo(‘body’);

You could also set the element’s inner HTML like this:

$(‘<div/>’, {

‘class’: ‘test’,

click: function() {

$(this).toggleClass(‘test’);

}):

}).html(‘<b></b>’).find(‘b’).text(‘Click me!’).end().appendTo(‘body’);

If using a loop to add multiple elements dynamically, either build a string of HTML or an array of HTML strings and create the DOM elements once from that string or array. For performance, don’t create a DOM element each time through the loop. This example shows the array way of doing it.

var mylis = [];

for (i=0; I < someLimit; i++)

{

mylis[i] = ‘<li>item ‘ + I + ‘</li>;’

}

$(‘<ul></ul>’, {

html: mylis.join(‘’);

}).appendTo(‘something’);

### Performance tips and inserting elements

Performance test your javascript code at <http://jsperf.com>

#### Cloning elements

Clone existing elements using $(‘#myId’).clone(). Use .clone(true) to copy events and data as well.

#### Inserting or moving elements

Inside other elements using .append(), .appendTo(), .prepend(), .prependTo().

Before and after other elements using .after(), .insertAfter(). .before(), .insertBefore().

Around other elements using .wrap(), .wrapAll() and .wrapInner().

In place of other elements using .replaceWith() and .replaceAll().

.*action*To() performs the action on the object it is being called on, using its parameter, whereas .*action*() performs the action on its parameter using the object it is being called on. E.g. $(*a*).appendTo(*b*) will append *a* to *b* whereas $(*a*).append(*b*) will append *b* to *a*.

In jQuery 1.4 and above, .append(), .prepend(), .after(), .before(), .wrap() and .wrapInner() can take a function as a parameter, which returns the content to add.

#### Removing elements

Removing elements with .remove() destroys the element entirely, including their data and events. Do this if you’re not going to need the element again later, to minimise memory usage in browsers which don’t do garbage collection very well.

Removing elements with .detach() just detaches the element from the DOM but doesn’t destroy it and leaves its data and events alone. Only do this if you might want to insert the element back into the DOM later.

.empty() removes data and events from the element but doesn’t remove it from the DOM.

### Content, attributes and properties

#### Content

.html() gets and sets HTML content and executes embedded javascript when setting.

.text() gets and sets text content. Any HTML entities such as <, > or & must be escaped.

.val() gets and sets form element values.

#### Attributes and properties

.attr() is used before jQuery 1.6.

Get and set any property with .prop()

Remove properties with .removeProp()

Get and set any attribute with .attr()

Remove attributes with .removeAttr()

Attributes are the attributes of elements that you see in the HTML, i.e. strings. Properties are properties of DOM elements, i.e. of objects. Some properties don’t have attribute equivalents. Some do have attribute equivalents but with different names because the attribute name is a reserved word in javascript, e.g.

|  |  |
| --- | --- |
| **prop** | **attr** |
| selectedIndex, tagName, nodeName, nodeType, ownerDocument | None |
| htmlFor | for |
| className | class |

Some attributes are defined in W3C as Boolean, these should only be get and set using .prop().

<input type=”checkbox” checked=”checked”>...</input>

<script>

// DOM property

// Will change with checkbox state

elem.checked == true;

// Will change with checkbox state

$(elem).prop(“checked”) == true;

</script>

<script>

// DOM method

// Initial state of the checkbox, does not change

elem.getAttribute(“checked”) == checked;

// (1.6) Initial state of the checkbox, does not change

$(elem).attr(“checked”) == “checked”;

// (1/6/1+) Will change with checkbox state

$(elem).attr(“checked”) == “checked”;

// (pre-1.6) Will change with checkbox state

$(elem).attr(“checked”) == true;

</script>

This inconsistent behaviour of .attr()shows why it’s better to use .prop()

Creating a JSON object to set the attributes of a new element uses .attr() behind the scenes, so the field names should be the attribute names, e.g. class, rather than the property names, e.g. className.

#### Getting and setting attributes

Get an attribute value using $(*something*).attr(‘*name*’);

Set attribute values one of two ways:

$(‘something).attr(‘title’, ‘this is my title’);

Or using a JSON object:

$(something).attr({

title: ‘this is my title’,

name: ‘some\_name’

});

Attributes can also be set using functions

$(something).attr(‘title’, function() {

return ‘go to ‘ + this.href;

});

Or using a JSON object

$(something).attr({

title: function(index, value) {

return ‘I do believe that ‘ + value;

},

name: ‘some\_name’

});

#### Getting and setting properties

Getting properties:

$(something).prop(‘id’);

$(something).prop(‘href’);

The href property / attribute can be confusing. The href *property* should be the fully qualified URL and the href *attribute* should be whatever text is in the HTML source. However Internet Explorer usually returns the fully qualified URL as the href attribute.

Setting properties is very similar to setting attributes:

$(something).prop(‘title’, ‘this is my title’);

and

$(something.prop({

title: ‘this is my title’,

name: ‘some\_name’

});

### CSS styles and dimensions

#### Styles and classes

Use .css() to modify the style property.

Use .addClass(), .removeClass() and .toggleClass() to modify the class attribute. This is preferable to .css() because it’s faster and also it gives better separation of concerns between logic and presentation.

Syntax for getting and setting styles and classes is the same as when using .attr() and .prop().

var col = $(something).css(‘color’);

$(something).css{‘backgroundColor’, ‘red’);

$(something).css({

backgroundColor: ‘red’,

‘margin-top’: ‘10px’

});

When using.css(), you can use either DOM properties (e.g. backgroundColor) or CSS syntax (e.g. margin-top). CSS identifiers which include a hyphen need to be surrounded by quotes because a hyphen isn’t valid in a javascript variable name.

.css() can also use functions:

$(something).css(‘fontSize’, function() {

return parseInt($(this).css(‘fontSize’), 10) + 2 + ’px’;

});

$(something).css({

backgroundColor: ‘red’,

marginTop: function(index, value) {

return parseInt(value, 10) \* 1.4;

}

});

This is a good example of where you’d use .css() over .addClass() etc, when you need to calculate the value of a style based on what it currently is.

#### Dimensions

Get and set element height and width using .height() and .width(). These return a number of pixels, without the px suffix.

Get and set the element offset relative to the document using .offset(). This returns an object, e.g. {top: 34, left: 50}.

Get the element offset relative to its offset parent using .position().

### Data

Arbitrary data can be associated with elements. Avoid storing data directly on DOM elements as this can cause memory leaks in browsers where the DOM engine and the javascript engine don’t really understand each other, resulting in circular references. Instead, associate data with jQuery objects.

// set data

$(something).data(‘personalInfo’, {firstname: ‘karl’});

// get data

var data = $(something).data(‘personalInfo’);

// remove data

$(something).removeData(‘personalInfo);

Unlike .prop(), .css() etc, these functions can’t use functions.

jQuery 1.4.4 onwards can read HTML5 data-\* attributes. See <http://learningjquery.com/2011/09/using-jquerys-data-apis> for more.

<div data-img=’{“alt”:”pic”,”src”:”path/file.jpg”}’></div>

var data = $(something).data(‘img’);

## Ajax

### $.ajax options and shortcut methods

#### url

var response = $.ajax({

url: ‘/url/to/serverResource’

});

var response = $.ajax(‘/url/to/serverResource’);

#### data

To send information to the server. On a GET request this will be included in the URL as a query string, on a POST request it will be included in the POST body.

var response = $.ajax({

url: ‘url/to/serverResource’,

data: {

key1: ‘value1’,

key2: ‘value2’

}

}):

When submitting a form, use .serialize().

$(‘#myform’).submit(function(event) {

event.preventDefault();

var formUrl = $(this).attr(‘action’),

formData = $(this).serialize();

var response = $.ajax({

url: formUrl,

data: formData

});

});

#### datatype

html for HTML to insert into document.

xml for XML documents, e.g. web services.

json for compact alternative to XML – easier to parse in javascript than XML is.

jsonp for remote resources. P stands for padding. Security prevents Ajax requests from one domain to another, so instead of returning JSON, the remote resource returns a function which returns the JSON object. This could be used e.g. to put a twitter feed on your site.

script for javascript files. Evaluates the response as javascript and returns the result as plain text.

var response = $.ajax({

url: ‘/url/to/serverResource’,

datatype: ‘json’

});

#### Other ajax options

See <http://api.jquery.com/jQuery.ajax/>

#### $.ajaxSetup()

Use this to set global defaults for ajax options.

### $.ajax responses

Before jQuery 1.5 these were handled by 3 more options:

var response = $.ajax({

success: function(response, status, xhr) {

// do something after successful request

},

error: function(xhr, status, errorThrown) {

// handle failed request

},

complete: function() {

// do something whether success or error, like a finally{} in C#

}

});

#### Ajax error handling in jQuery 1.5+

Explicit testing for ajax errors is important. Network failures don’t occur in local development environment. Parsing failures can happen if JSON is expected and the response isn’t valid JSON.

$.ajax implements the Promise interface

* Returns a jqXHR object (superset of XML HTTP request), a Promise
* Promise objects are derived from the Deferred object – see <http://api.jquery.com/category/deferred-object/>
* .done() and .fail() methods in jQuery 1.5+
* .always() method in jQuery 1.6+
* .done(), .fail() and .always() can be called multiple times to add more than one handler

Can store the result of an ajax request in a variable and attach handlers later for more manageable code structure. Handlers are invoked immediately if the ajax request has already responded

Ajax requests can be cancelled in an elegant way.

var myOptions = {

url: ‘http://api.jquery.com/jsonp/’,

datatype: ‘jsonp’,

data: {

title: search

}

};

$.ajax(myOptions).done(successfn).fail(errorFn).always(completeFn);

### Multiple handlers and promises

Response handling can be implemented using multiple function arguments, an array of functions or by chaining function calls:

request.done(successFn1, successFn2, successFn3);

request.done([successFn1, successFn2, successFn3]);

request.done(successFn1).done(successFn2).done(successFn3);

This opens up lots of possibilities for creating and reusing functions to pass to .done(), .fail() and .always().

#### Caching ajax responses

This is a simple approach, for a more generic abstracted approach see “script junkie: creating responsive applications using jQuery Deferred and Promises” <http://bit.ly/tph6F6>

(function() {

var api = {}, $response = $(‘#response’);

$(‘#ajaxForm’).bind(‘submit’, function(event) {

event.preventDefault();

var search = $(‘#title’).val();

$response.empty().addClass(‘loading’);

if (!api[search]) {

api[search] = $.ajax({

url: ‘http://api.jquery.com/jsonp/’,

datatype: ‘jsonp’,

data: {

title: search

},

timeout: 15000

});

}

api[search].done(successful).fail(errorFn).always(completeFn);

});

})();

## Effects and animations

### Basic methods and callbacks

#### Fades

$(‘.tooltip’).fadeIn();

$(‘.tooltip’).fadeOut(‘slow’);

$(‘.tooltip’).fadeToggle(250);

$(‘.tooltip’).fadeTo(200, 0.5);

#### Slides

$(‘div.panel’).wrapAll(‘<div class=”panel-wrapper”></div>

$(‘div.panel-heading a’).click(function() {

$(this).parent().next(‘.panel-wrapper’).slideToggle();

return false;

});

#### Callback

Function executed when the animation ends. Called once for each animated element. Consider using .promise() instead if using jQuery 1.6+.

$(‘div.toRemove’).slideUp(‘slow’, function() {

$(this).remove();

});

$(‘div.move’).slideDown(250).promise().done(function() {

alert(‘finished’);

});

#### Animation order

By default, multiple animations occur

* In sequence for the same element
* Simultaneously for different elements

#### Custom animations

$(‘div.toMove’).animate(propsObject, duration, easing, callbackFn);

$(‘div.toMove’).animate(propsObject, optionsObject);

### Animating properties and easing

Properties can be animated by number, percent etc, or relatively (e.g. +=200px, -=20%).

#### Custom animations

Example, slowly moving an element 300px to the right:

$(‘.toMove’).animate({

left: ‘=300px’

}, 800);

$(‘.toMove’).animate({

left: ‘+=300px’

}, {

duration: 800

});

Note that elements can only be moved if their position style is not static.

Options object allows for fine tuning:

* duration – a string or number determining how long the animation will run for
* easing – a string indicating which easing function to use (linear, swing etc). More available with plugins.
* complete – callback function to call once the animation is complete
* step – a function to be called after each step of the animation
* queue – Boolean indicating whether to place the animation in the effects queue. If false then the animation begins immediately.
* specialEasing – a map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions (added in jQuery 1.4)

#### Easing

Easing changes the velocity at different points in the animation. A number of standard equations first created by Robert Penner. Available with jQuery UI, or stand-alone at <http://gsgd.co.uk/sandbox/jquery/easing/>

Queues such as the effects queue can be converted to a Promise in jQuery 1.6+:

$(‘button’).bind(‘click’, function() {

$(‘div.pcb’).slideToggle(400).promise().done(function() {

// do something once after all div.pcb elements have finished sliding

});

});

Example of a custom animation with easing:

$(‘something).animate({height: ‘toggle’}, {

duration: 600,

easing: ‘linear’

});

Per-property easing available in jQuery 1.4+:

$(something).click(function() {

$(somethingElse).animate({

width: [‘toggle’, ‘swing’],

height: [‘toggle’, ‘swing’],

opacity: ‘toggle’,

}, 500, ‘linear’, function() {

$(this).after(‘<div>complete!</div>’);

});

});

Another multi-property easing example:

$(‘div.demodiv’).animate({

left: ‘300px’,

top: ‘245px’,

width: 0,

height, 0,

marginTop: ‘55px’

}, {

duration: 5000,

easing: ‘linear’,

specialEasing: {

left: ‘easeOutQuad’,

top: ‘easeOutBounce’

}

});

More examples at <http://pres.learnjquery.com>

Selector to identify animated elements

$(‘.toMove’).click(function() {

if (!$(this).is(:animated’)) {

$(this).animate({

left: ‘+=300px’

}, ‘slow’);

}

});

### Stop/delay and global settings

#### .stop()

Stops current animations from continuing. Two arguments: clearQueue and gotoEnd, both Boolean.

$(something).hover(function() {

$(this).stop(true, true).animate({bottom: ‘30px’}, 200);

}, function() {

$(this).stop(true, false).animate({bottom: ‘8px’}, 500;

});

#### .delay()

Delays any further items in the queue from executing for a specified period of time.

$(something).fadeIn(600).delay(4000).fadeOut();

#### Global duration

Globally change the duration that all animations will use unless a duration is specified.

// duration is set to 400 by default

$(something).slideToggle();

// modify default globally for all future effects

$.fx.speeds.\_default = 250;

$(something).slideToggle();

#### Global off

Globally prevent all animations from happening. This could be useful for mobile devices which don’t do javascript animations very well.

$.fx.off = true;