

# Foundations

Epic 2016

---

*Web-Basics Programming*

Epic Systems Corporation  
1979 Milky Way • Verona, WI 53593 • Voice: (608) 271-9000 • Fax: (608) 271-7237  
[www.epic.com](http://www.epic.com)  
[documentation@epic.com](mailto:documentation@epic.com)  
Last Revision: October 14, 2016

# Web-Basics Programming

<b>Web Basics .....</b>	<b>1•3</b>
<b>Creating Content .....</b>	<b>2•5</b>
<b>Setting Layout.....</b>	<b>3•3</b>
<b>Designing Client Behavior .....</b>	<b>4•5</b>
<b>Implementing Business Logic .....</b>	<b>5•3</b>



# Lesson 1: Web Basics

<b>The Big Picture.....</b>	<b>1•3</b>
Philosophy of HyperspaceWeb .....	1•3
By the End of This Lesson, You Will Be Able To... .....	1•3
<b>Old and New.....</b>	<b>1•5</b>
Advantages of Moving to Web.....	1•6
Disadvantages of Moving to Web.....	1•6
<b>Transitional System Structure .....</b>	<b>1•8</b>
<b>How the Web Works.....</b>	<b>1•10</b>
Traditional Flow.....	1•10
Flow used by HyperspaceWeb.....	1•11



# Web Basics

## The Big Picture

This lesson covers why Epic is making the transition from VB to the web, as well as how various web technologies will be used differently by Epic (compared to standard industry usage).

### Philosophy of HyperspaceWeb

- Simple and **powerful**
- Constrained and **capable**
- **Modern** and functional
- **Less** is more
- **Perform** and be **manageable**
- **Predictable** and **supportable**
- Preserve and **grow**

*A platform for the next  
10+ years.*

As you can see, when the web transition team began designing the new web framework, they had many varied and sometimes conflicting goals in mind. They want it to be simple but powerful, modern and so on.

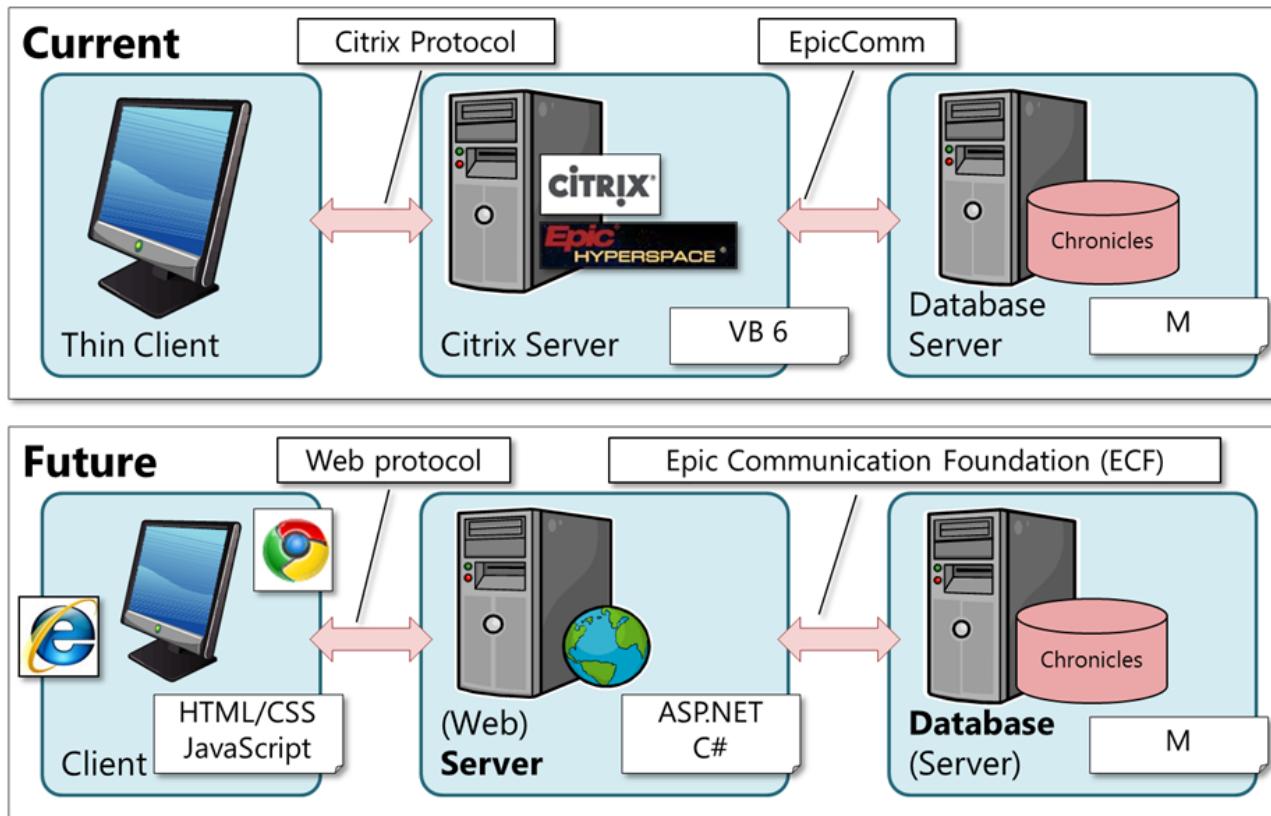
The overarching goal was to create something that works and will stand the test of time, like VB before it. If the new framework lasts for at least 10 years after the transition is complete, then it will be considered a success.

Trying to make something that would last forever is impossible, and would lead to an overdesigned, slow and unusable framework.

**By the End of This Lesson, You Will Be Able To...**

- Describe the difference between the tiers of VB Hyperspace and HyperspaceWeb
- Describe the transitional system structure
- List the key benefits and challenges of moving to the web
- Describe how the request flow used by HyperspaceWeb is different from traditional web applications

# Old and New



In our current architecture, Hyperspace runs on a Citrix server, which end users connect to through thin client machines. Hyperspace then communicates with the Database using EpicComm or EpicComm over ECF.

In the new system, the Citrix server is replaced with a web server running Microsoft's Internet Information Services (IIS), with C# as the code-behind language. Clients connect to the web server with a supported web browser, and view the ASP.NET pages, which are rendered to HTML, CSS and JavaScript when sent to the client. The server communicates with the database using full ECF.

ASP.NET	Active Server Page with .NET Framework <ul style="list-style-type: none"> <li>Markup of page content on the web server</li> <li>C# can be used to modify content at runtime</li> </ul>
HTML	HyperText Markup Language <ul style="list-style-type: none"> <li>Content of the page on the client</li> </ul>
CSS	Cascading Style Sheet <ul style="list-style-type: none"> <li>Determines look of page, colors, fonts, etc.</li> </ul>

JavaScript	<p>Provides dynamic behavior on client</p> <ul style="list-style-type: none"> <li>• Hide and unhide elements</li> <li>• Handle user events</li> <li>• Validate user input</li> <li>• Interacts with elements on the page (tables, controls, etc.)</li> <li>• Send requests for data to the server.</li> </ul>
------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Advantages of Moving to Web

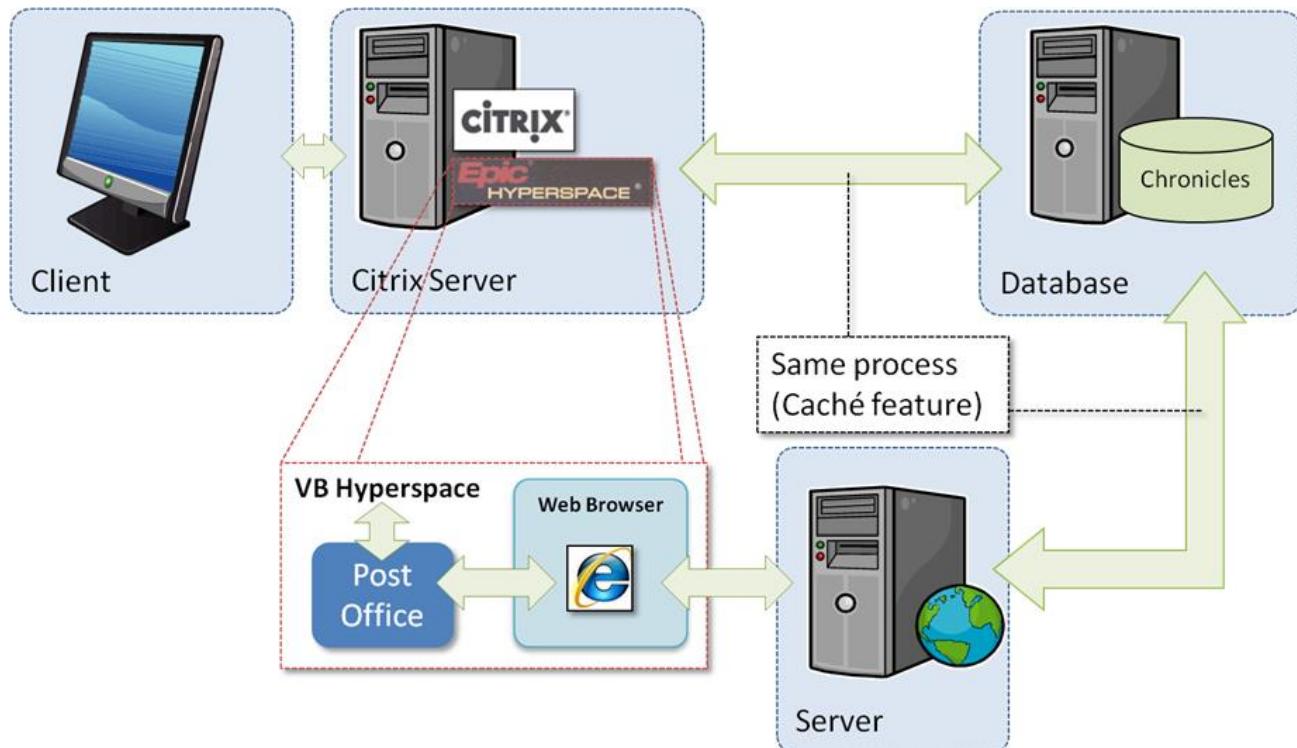
Flexible presentation	Web natively supports Cascading Style Sheets, which can be swapped out to easily change Hyperspace Themes or to account for layout differences on different locales.
Meets customer expectations	Mainstream support for VB6 ended in 2005 and extended support ended in 2008. Therefore moving off VB6 as a platform is critical for Epic.
Uses industry standards	Web is mainstream and well understood in the industry.
Avoids Citrix issues	Citrix is a great product, but it is expensive and slow when compared to a well-written web application.

## Disadvantages of Moving to Web

Security of the client browser	<p>It is very easy to gain access to the content and code of a web application once it is sent to the client, and anyone with a bit of programming knowledge can view and alter this code. It is also easy to observe network traffic as it moves between the client and the web server.</p> <p>Therefore extra precautions must be taken on the web server to ensure the incoming data has not been tampered with.</p>
--------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5 new languages	Developing in the web often requires knowledge of many different technologies. In the case of HyperspaceWeb, this includes HTML, CSS, JavaScript, ASP.NET and C#.
Steeper development learning curve when compared to programming in VB	<p>Now there are more choices when deciding where to place a piece of functionality.</p> <p>In VB, there were only two choices when deciding where to place code: in Hyperspace (written in VB) or in the database (written in M).</p> <p>In web, you can choose between the client (HTML/CSS/JavaScript), the web server (ASP.NET/C#) or the database (M).</p> <p>There are times, such as when writing validation logic, when the correct answer is to duplicate the code both on the client and the web server.</p>
Transition will take time	Epic has millions of lines of VB code and thousands of activities that need to be re-written. The transition will take many years to complete.

# Transitional System Structure



*Transitional System Structure: HyperspaceWeb Hosted in VB*

During the transition, the old architecture remains in place, but Hyperspace is slightly modified; a special activity is embedded in Hyperspace to host web activities. Rather than communicating directly with the database, like other activities, the web browser communicates with the web server.

Notice that now there are two separate paths to the database. Under normal circumstances, this means that there are two separate database processes associated with the same user.

Think about the scenario where a web activity and a VB activity need to use the same record. What problem could this cause? One activity could lock the other out of the record, even though it is the same user working in both activities at the same workstation.

InterSystems Caché allows both connections to coexist in the same process. For GT.M environments, Epic has a special module called the Shared Connection Provider (SCP) that provides a similar feature.

Another integration point is the post office. Using JavaScript, Web activities can send and receive messages to/from VB through the post office. This allows Web activities to integrate tightly with VB. As the transition progresses, the system will maintain consistent communication with both the VB and web versions of the same activity, which will listen for the same post office messages.

One key limitation is that only strings can be sent, not complete objects.



Can the server communicate directly with VB?

-

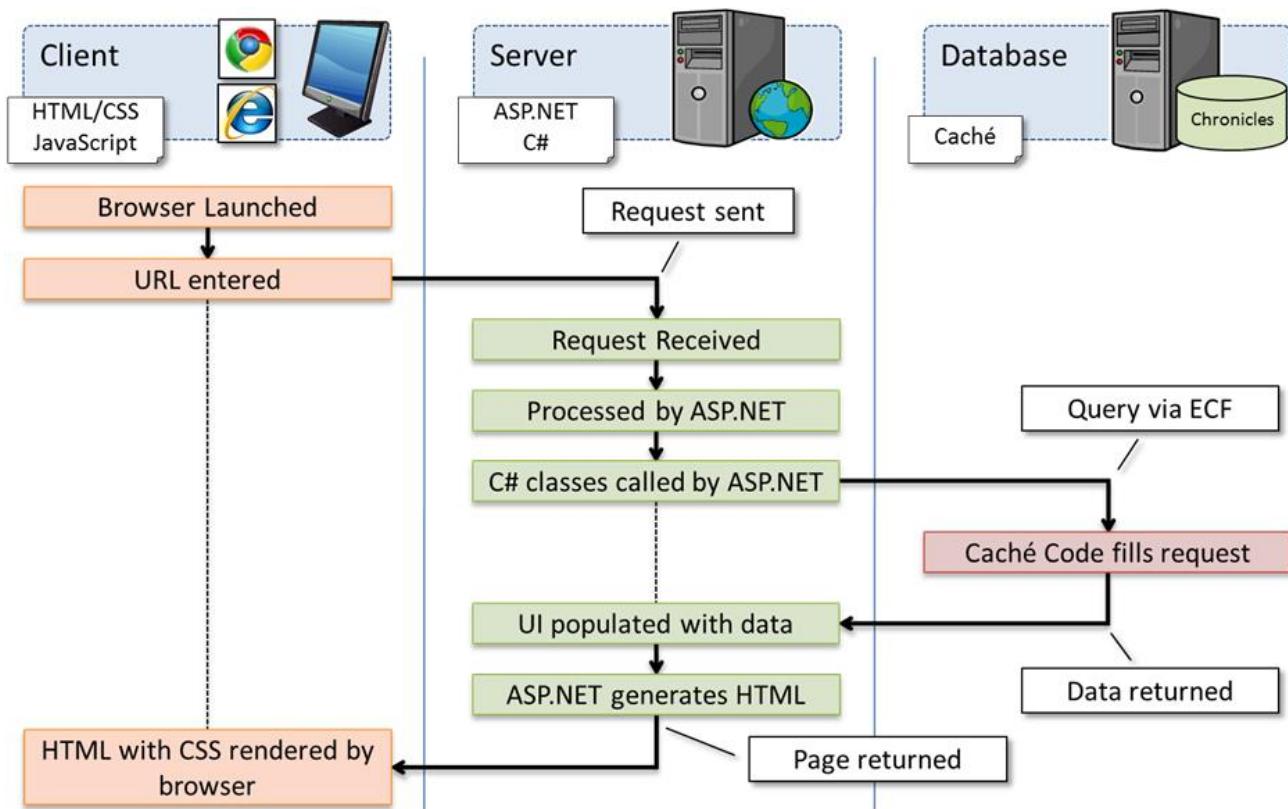
# How the Web Works

A common concern among developers with web experience is that traditional web applications do not perform as well as desktop applications. The perception is that Hyperspace will be slower in web than it was in VB.

The developers of the HyperspaceWeb framework were very aware of this concern and have created communication patterns that avoid this problem.

## Traditional Flow

The following diagram illustrates how a traditional web application functions:



The first thing that happens is the user launches a browser and then enters a Universal Resource Locator, or URL.

Once the URL is entered, a request is sent to the server. Upon receiving the request, the server will process the request using ASP.NET.

This involves instantiating all C# classes required to fill the request. If data from the database is required, the C# classes may query data from Chronicles using an ECF command. Once the data is returned, the UI of the page is populated with the returned data using the C# code behind.

Web browsers don't understand ASP.NET content, so before sending the response back to the client, all ASP.NET content is rendered to HTML, which browsers do understand. Finally, the page is returned and rendered on the browser.

URL

Uniform Resource Locator

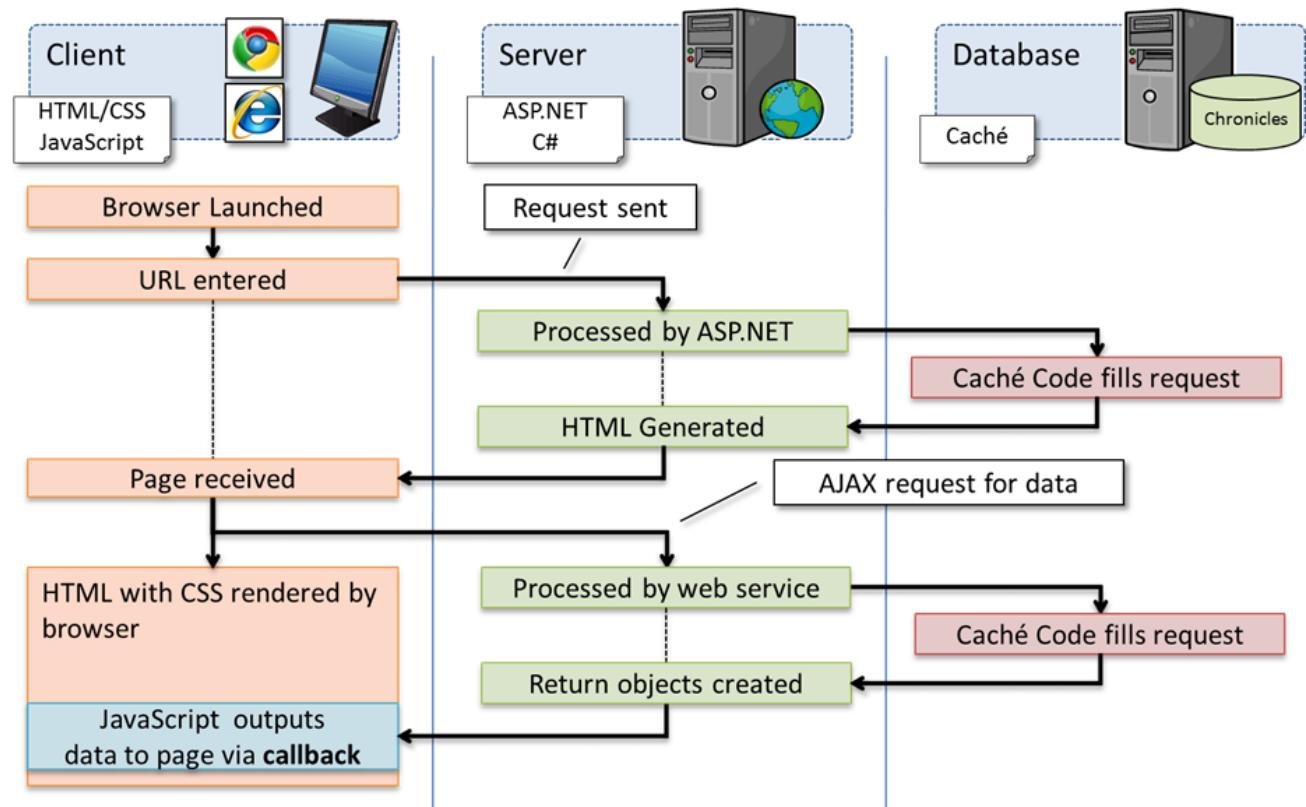
**Example:** `http://www.domain.com/CustService/Default.aspx`



Is this flow synchronous or asynchronous?

## Flow used by HyperspaceWeb

The following diagram illustrates how HyperspaceWeb functions:



Initially things look similar to a traditional web application, with one major difference: the request to the database is very small or is omitted completely. This allows a shell of the page to be quickly returned to the client so the render process can begin, giving the appearance of a more responsive system.

As the page is rendered, an AJAX request is sent to the server to obtain all the information from the database that wasn't returned with the initial page request.

Once the remaining data is returned to the client, a JavaScript callback is called to insert the data into the controls of the page. At this point the rendering process is complete.

The key here is that the page should start loading as quickly as possible so that the render process on the client and the second load-data call to the server can happen at the same time. The result is a faster and more responsive system.

## AJAX

### Asynchronous JavaScript And XML

- Communicate with server
- Permit user to continue working
- Page refresh not required

Although AJAX messages are traditionally in XML, this format is not required. HyperspaceWeb communicates using JavaScript Object Notation (JSON), which is more concise, resulting in less network traffic.

# Lesson 2: Creating Content

<b>The Big Picture.....</b>	<b>2•5</b>
Flavors of ASP.NET .....	2•5
Scenario.....	2•6
<b>Structuring Code.....</b>	<b>2•7</b>
General Principle.....	2•7
Categorizing Code .....	2•7
Logically Separating Code.....	2•9
Physically Separating Code.....	2•12
Exercise 1: Create a Web Application Shell.....	2•15
Part 1: Prepare Visual Studio.....	2•15
Part 2: Create the Folder Structure and Project .....	2•16
Part 3: Add additional files and test the site.....	2•19
Part 4: Running on other browsers.....	2•20
<b>Creating Basic Client Content.....</b>	<b>2•22</b>
Customer Service Page Content.....	2•22
Markup Languages.....	2•22
Why use markup? .....	2•23
Extensible Markup Language (XML).....	2•23
HTML is similar to XML.....	2•24
Epic's HTML Conventions .....	2•25
HTML DTDs .....	2•25
Page Tree .....	2•25
Exercise 2: Create a Simple HTML Page .....	2•26
Part 1: Overview.....	2•26
Part 2: Dividing the Page Vertically.....	2•27
Part 3: Headings and Paragraphs.....	2•28
Part 4: Images .....	2•30
Part 5: Lists and Links .....	2•32
Part 6: Special Characters.....	2•35
Wrap Up .....	2•36
<b>Displaying Consistent Content.....</b>	<b>2•39</b>
Share Content Using a Master Page .....	2•40
Exercise 3: Share Content Between Pages .....	2•40
Part 1: Creating a Master Page .....	2•41

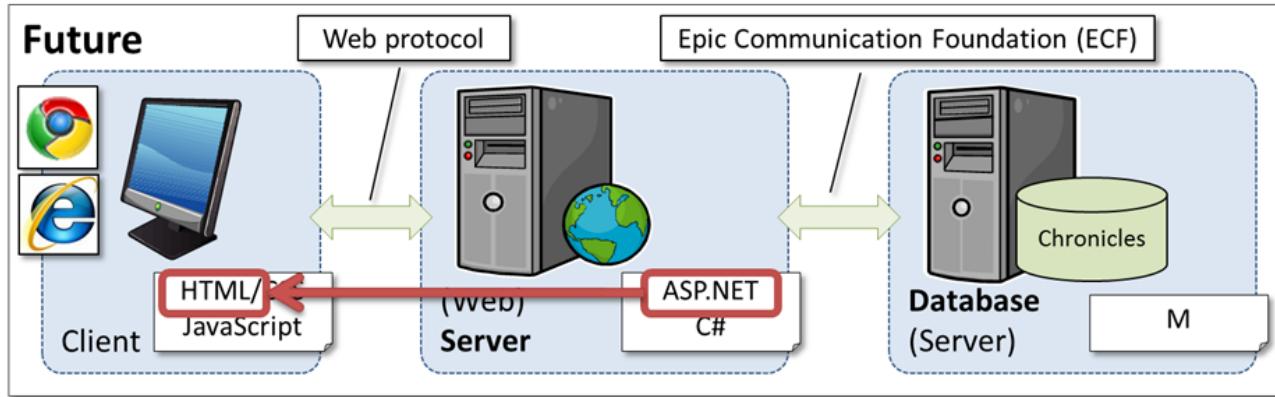
Part 2: Creating the Home Page.....	2•45
Wrap Up.....	2•46
If you have time.....	2•46
<b>Including Tables and Forms.....</b>	<b>2•48</b>
Scenario .....	2•48
Creating Tables.....	2•48
Creating Forms.....	2•49
Exercise 4: Creating Tables and Forms.....	2•51
Part 1: Tables.....	2•51
Part 3: Forms.....	2•52
Wrap Up.....	2•54
If you have time.....	2•56
<b>Creating Server Content .....</b>	<b>2•57</b>
Why create content in ASP.NET vs. HTML (or vice versa)? .....	2•57
Exercise 5: Creating Server Content .....	2•58
Part 1: Customer Service Page.....	2•58
Part 2: Inventory Page.....	2•61
Part 3: Order Page.....	2•64
Wrap Up.....	2•65
<b>Creating Custom Controls .....</b>	<b>2•66</b>
Examples.....	2•66
Credit Card Entry.....	2•66
Clock.....	2•66
Kinds of ASP.NET Custom Controls.....	2•67
User Controls.....	2•67
Web Controls.....	2•67
Why choose one kind of control over the other?.....	2•67
Exercise 6: Creating Custom Controls.....	2•68
Part 1: Create a Billing-info User Control.....	2•68
Part 2: Use the Billing-info User Control .....	2•68
Part 3: Create a Custom Attribute.....	2•69
Part 4: Clock WebControl Overview .....	2•71
Part 5: Add a new Project for the clock control .....	2•72
Part 6: Create the Control.....	2•73
Part 7: Use the Control.....	2•76
Part 8: Add an Attribute.....	2•79
Wrap Up.....	2•79





# Creating Content

## The Big Picture



As you learned during the previous lesson, the new architecture is broken down into three tiers: the client, server and database. Static content in the system will either be written in HTML or in ASP.NET. In either case, the content is sent from the server to the client when requested.

The big difference between HTML and ASP.NET is that ASP.NET is not natively recognized by browsers, so it needs to be rendered into HTML as part of the request. This process provides the opportunity to change the content dynamically before it reaches the client.

## Flavors of ASP.NET

Microsoft provides three different methods of creating ASP.NET content. It's important that you understand the difference between these so when you look for resources online, you don't use techniques that are incompatible with HyperspaceWeb.

### Web Forms

**This is the flavor of ASP.NET used in HyperspaceWeb.**

The Web Forms framework targets developers who prefer declarative and control-based programming.

Web Forms provide the following features:

1. An event model similar to desktop client application frameworks, like .NET Windows Forms.
2. Server controls that render HTML for you and that you can customize by setting properties and styles.
3. A rich assortment of controls for data access and data display.

	<p>4. Automatic preservation of state (data) between HTTP requests, which makes it easy for a programmer who is accustomed to client applications to learn how to create applications for the stateless web.</p> <p>Note that HyperspaceWeb relies very little on 3 and 4 above because Epic has created custom controls and state preservation techniques.</p>
Web Pages	<p>This is a simplified version of Web Forms. It is not used by HyperspaceWeb.</p>
MVC	<p>This framework encourages separating the business logic layer of a web application from its presentation layer.</p> <p>HyperspaceWeb accomplishes the same task using the MVVM design pattern on top of Web Forms.</p> <p><b>You should not use ASP.NET MVC in HyperspaceWeb.</b></p>



[Consult MSDN for more information on the flavors of ASP.NET.](#)

## Scenario

The scenario we will use is the Big Mouse Cheese Factory, an international distributor of gourmet cheeses.

You will be building a standalone site for this company that includes a home page, customer service page, orders page and a simple inventory management system.

However, before you write even one line of code, we must carefully consider how the code will be structured.

# Structuring Code

Understanding how the code is structured will make it easier for you to integrate your development into the overall system. It will also help you find shared components that you need quickly as well as effectively track down the source of bugs in the system.

## General Principle

The overarching principle behind how code will be structured is the separation of the user interface from the business logic. The question is, why is this so important?



Why separate the user interface from the business-logic?

In order to maintain flexibility in our code so it can be reused with multiple platforms, such as text, web and Windows Forms, it is important to separate all non-interface-related tasks (that is, business logic) from the interface we present to the users.

As you will see shortly, there are several more ways to categorize and separate code to promote reuse.

## Categorizing Code

In order to maximize the reusability of code, we need to know precisely when code should and should not be shared. To facilitate this, code is categorized and divided into different projects and namespaces.

Understanding the criteria used to categorize code is crucial both when deciding where to place new code and when tracking down existing code that you want to reuse.

There are five categories of code used to divide the source into reusable chunks. In each category, there is a special option that indicates the code is not tied to any particular option within that category and could be shared among all of them (in other words, it is independent of that category).

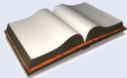
Category	Meaning
Product	The product is a collection of applications that are released within the same integrated framework.

Category	Meaning
	<p><b>Examples:</b></p> <ul style="list-style-type: none"> <li>• HyperspaceWeb</li> <li>• EMC2</li> <li>• If the code is product-independent, then the product category is <b>Code</b>.</li> </ul>
Owner	<p>The owner is a broad category that spans multiple related applications.</p> <p><b>Examples:</b></p> <ul style="list-style-type: none"> <li>• Access, Billing, Clinical, Reporting, Revenue</li> <li>• If the code is owned by Foundations (making it available across all other owners) then the owner is <b>Core</b>.</li> </ul>
Application	<p>An individually branded program, feature or module</p> <p><b>Examples:</b></p> <ul style="list-style-type: none"> <li>• <b>Clinical Applications:</b> Ambulatory, Emergency, Inpatient, Lab, OR, Orders, Pharmacy</li> <li>• <b>Access Applications:</b> ADT, HIM, Scheduling</li> <li>• If the code is shared across applications (for example, features owned by ApplCore), the application is <b>Common</b>.</li> </ul> <p>Common can also appear at the owner level if it is shared across multiple owners but is still not considered part of Foundations.</p>
Functional Area	<p>Sub-modules within the same application</p> <p><b>Examples:</b></p> <ul style="list-style-type: none"> <li>• <b>Clinical &gt; Orders:</b> Administration, Data, Services</li> <li>• <b>Access &gt; ADT:</b> Administration, BedPlanning</li> <li>• If the code is shared by all parts of the application, the functional area is <b>Base</b></li> </ul> <p>Sub areas can also be applied at this level. For example, the Chrongrid replacement for the web framework is in the <b>Controls.Grid</b> functional area (Product: Code and Owner: Core).</p>

Category	Meaning
Platform	<p>The user-interface framework.</p> <p><b>Examples:</b></p> <ul style="list-style-type: none"> <li>Text, Web, WinForms, WPF</li> <li>If the code is not specific to any one platform, then the platform is listed as <b>Model</b>.</li> </ul>

In general, if two C# classes share the same value for each of the five criteria listed above, then they belong in the same project. If they are in the same project, then they will end up in the same Assembly.

How widely code within an assembly can be shared depends on how it is categorized.



For example, code owned by Foundations (Core) that is platform independent (Model) can be used by anyone. Examples in this category include data-type extensions to .NET classes such as Boolean and DateTime.

On the other hand, if code is specific to the web platform, then it should not be included in code that applies to another platform, especially Model.

The parts of code are used to determine the logical division of code (namespaces) as well as the physical division (folder structure).



### Why categorize code?

- 
- 
- 
- 

## Logically Separating Code

Recall that namespaces are used to logically divide code in the .NET framework. Different assemblies (physical locations on disk) can contribute to the same logical collection of code (namespace).

At Epic, namespaces consist of the parts of code discussed earlier. The namespace structure is slightly different if the code in question is product independent or product dependent. The namespaces of product-independent code are made up of the Epic prefix (to distinguish it from Microsoft code), the owner, application, one or more functional areas and finally the platform.

Prefix	Owner	Application	[Area.SubArea]	[Platform]
Epic.	Core.	Database		
Epic.	Common.	BreakTheGlass.	Configuration	
Epic.	Billing.	Professional		
Epic.	Clinical.	Orders.	Administration.	Web

The functional area and platform are optional, meaning that if the code is functional-area independent (Base) or platform-independent (Model) then that portion is omitted from the namespace.



Do you remember what Core and Common mean?

- Core:

---

- Common:

---



## Why are the Area and Platform parts of the namespace optional?

Think of the namespace as a tree structure. If you are using code defined in the root namespace, then you automatically have access to that code from all branches extending from the root. However, if you need to include something from a different branch of the namespace tree, then you need to explicitly add using at the top of your C# code.

Suppose you are a developer in PB. If the area and platform were not optional, then all shared models in your application would live here:

`Epic.Billing.Professional.Base.Model`

Further suppose you are creating new pages in the Administration-area project:

`Epic.Billing.Professional.Administration.Web.Pages`

The problem you would face is that you would need to explicitly include the `Base.Model` namespace in every page that you write:

`using Epic.Billing.Professional.Base.Model`

However, by making the area and platform optional when they are shared and platform independent, then you will have access to that code without the using statement, because:

`Epic.Billing.Professional`

is automatically available to:

`Epic.Billing.Professional.Administration.Web.Pages`

This works because your namespace is an extension of the shared PB namespace.

The following namespaces are examples from the training exercises:

Prefix	Owner	Application	[Area.SubArea]	[Platform]
Epic.	Training.	Core.	Controls.	Web
Epic.	Training.	Core.	Controls.Time.	Web



### Why is Platform listed last?

It is important to be able to recognize when references that you include in your projects have a platform dependence.

If your project contains a reference to a \*.Web assembly, then your assembly should also end in Web. If it does not end with Web and does reference a \*.Web assembly, then you are misrepresenting your project as platform independent.

The product-dependent namespaces are tied to the platform used by the product. UI code at the activity level is usually product dependent.

Examples:

Prefix	Owner	Application	[Area.SubArea]	Platform	Project Type
Epic.	Common.	BreakTheGlass.	Configuration.	Web.	Views
Epic.	Billing.	Claims.		Web.	Views

Notice that there is an additional part of the namespace, which is the project type. In HyperspaceWeb, the project type is either Views or Assets. Views are UI controls representing activities, while Assets are projects that contain shared web resources, such as scripts, images, icons and style sheets.

In this part of training we use Pages as our UI element instead of Views since we don't have access to the HyperspaceWeb framework yet.

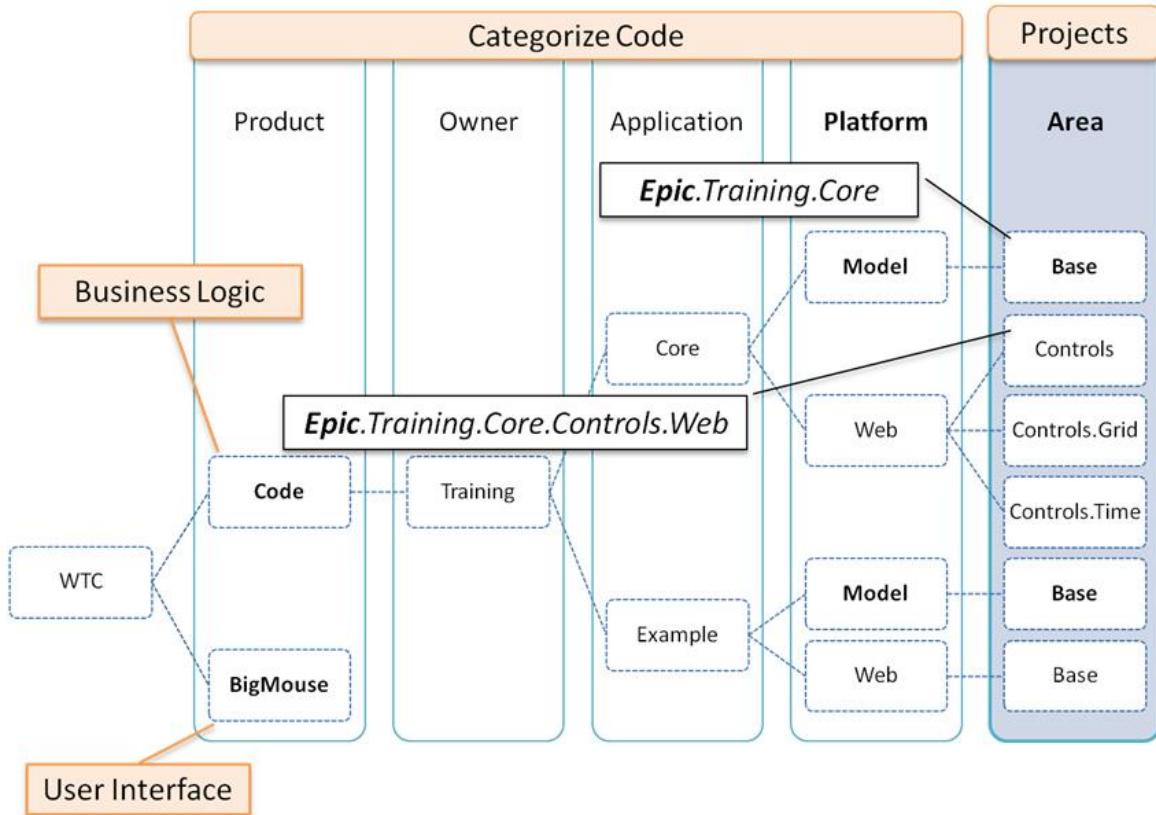
Examples:

Prefix	Owner	Application	[Area.SubArea]	Platform	Project Type
Epic.	Training.	Example.		Web.	Pages

Because it's impossible to define the UI of a product without specifying a platform, the platform is always required for product-specific code.

## Physically Separating Code

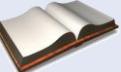
You will use the following folder structure with the in-class example to separate the different kinds of code. HyperspaceWeb uses a similar structure.



### *Training Folder Structure*

The root folder will be located in C:\EpicSource\WTC. From there, the different levels correspond to the different code parts. The Product, Owner, Application and Platform levels only contain sub folders, while the actual projects and source files are located in the Area level. Since the separation of UI and business logic is one of the key goals of the folder structure, this separation happens right away in the Product level.

Notice how the namespaces are similar to the folder structure, but are slightly different.

 For example,

- The project stored in WTC\Code\Training\Core\Model\Base has the namespace Epic.Training.Core
- The project stored in WTC\Code\Training\Core\Web\Controls has the namespace Epic.Training.Core.Controls.Web.



What is different between the namespace and folder structure conventions?

See if you can find all of the differences.

1.

---

2.

---

3.

---

4.

---

Read on to verify your answer.

Difference	Reason
Folders don't have the Epic prefix, namespaces do	Only code that we write will live in the folder structure, so a prefix isn't necessary. However, our namespaces must co-exist with Microsoft and customer namespaces, so the Epic prefix is used to distinguish the two.
Folders split between business-logic (Code folder) and UI (product folder) right away.  Namespaces optionally specify a UI type at the end of the namespace	The physical separation of UI and business logic in the folder structure ensures that the two will never mingle in the same assembly. This enhances the reusability of the code.  For namespaces, the UI-type is listed last to make it clearer that the code is or is not product dependent.
The folder structure swaps the platform and the area, when compared to the namespace.	This makes it easier to navigate the folder structure because you will only see projects that work with the platform that you selected at the higher folder level.  The platform is listed last in the product-independent namespaces to make it easy to recognize platform dependencies.

Difference	Reason
Namespaces omit unused code parts, while folders use a default name: Model for platform and Base for functional area.	The unused parts are dropped from namespaces to keep them concise and limit the number of using statements required in code. The default names are used in the folder structure to maintain a consistent height for the folder tree. This makes it easier to specify relative paths (which are critical to the proper function of the framework).

Another convention is that individual files are not shared between projects. Instead, assemblies are shared.

This is different than VB bas modules, where one file can potentially get compiled into multiple assemblies. Changing the module could result in changes to many different projects, which is not desirable.



A file should only ever be included in a single project.

## Exercise 1: Create a Web Application Shell

In this exercise you will create a web-application shell that logically and physically separates code according to Epic convention. This project will be used throughout part 1 of Web Tech Camp.

### Part 1: Prepare Visual Studio

1. Start Visual Studio
2. Reset your settings so that it is easier to follow the in-class instructions
  - a. Tools > Import and Export Settings > Reset All Settings
  - b. Save your current settings if you will want them later
  - c. Choose **General Development Settings** if using Visual Studio 2013, or just **General** if using Visual Studio 2015.
  - d. Click **Finish**
3. Import the code format settings
  - a. Tools > Import and Export Settings > Import selected environment setting

- b. Choose: **No, just import new settings, overwriting my current settings**
  - c. Click **Browse**
    - i. **If using Visual Studio 2013:**  
M:\fnd\FndGUI\HyperspaceWeb\VS2013\CodeFormatting.VS2013.vssettings
    - ii. **If using Visual Studio 2015:**  
M:\fnd\FndGUI\HyperspaceWeb\VS2015\TeamSettings.VS2015.vssettings
  - d. Click **Next**
  - e. Ignore any warnings and click **Finish**
4. Close Visual Studio
  5. Ensure you have the Epic templates:
    - a. If using Visual Studio 2013:  
M:\fnd\FndGUI\HyperspaceWeb\Templates\install.lnk
    - b. If using Visual Studio 2015:  
M:\fnd\FndGUI\HyperspaceWeb\VS2015\Templates\install.lnk



Templates and snippets simplify the work of creating a new project and adding new items and code by automating several tasks, including:

- Referencing common assemblies
- Adding basic files required for the project to compile & run
- Generating an example UI

You now have Epic templates on your computer

## Part 2: Create the Folder Structure and Project

1. Create a new folder for your in-class exercises on your local machine in the following location:
  - C:\EpicSource\WTC



Paths to web applications should not use special characters, such as the pound sign in "C#". This will cause problems with ASP.NET.

2. Create another folder:
  - C:\EpicSource\WTC\Solutions



You will create several solutions in this class. This will be a central location where you can keep them.

3. Start Visual Studio
4. Create a new web application:
  - File > New > Project > Visual C# > WTC > WTC: Web Application
5. **Name:** BigMouse
6. **Location:** C:\EpicSource\WTC\
7. Uncheck **Create directory for solution**
  - A folder matching *Name* will automatically be created within *Location*



A web application is preferred over a web site, because a web application compiles its projects to assemblies prior to deployment, while web sites store the source code on the server and compile them on demand.

8. In the solution explorer, select the solution (root of the tree):



*Solution Explorer*

Main Menu > View > Solution Explorer (Ctrl + Alt + L)



If you don't see the solution, check the following setting:

- Main Menu > Tools > Options > Projects and Solutions > Always Show Solutions

9. Save the solution to the Solutions folder:
  - Main Menu > File > Save BigMouse.sln As
  - File Name: C:\EpicSource\WTC\Solutions\BigMouse.sln



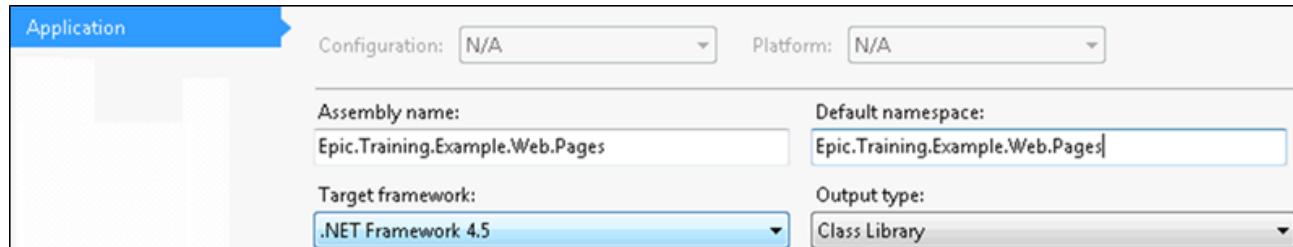
Visual Studio will automatically create a sln file and other related files within C:\EpicSource\WTC\BigMouse\ upon new project creation.

You should remove these files after relocating the solution.

10. Browse to C:\EpicSource\WTC\BigMouse
11. Delete all solution-related files from this folder:

- BigMouse.sln
  - BigMouse.v12.suo, if it exists
  - the .vs folder, if it exists
12. Read about project naming on the [HyperspaceWeb Wiki](http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting_Started/Namespaces_and_Code_Structure#Project_names)  
([http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting\\_Started/Namespaces\\_and\\_Code\\_Structure#Project\\_names](http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting_Started/Namespaces_and_Code_Structure#Project_names))
- In training we are using the application name as the project name
13. Rename the project (not the solution)
- Solution Explorer > BigMouse (project, not solution) > Right Click > Rename > *Training.Example.Web.Pages*
14. Set the default namespace and the assembly name for this project. This will ensure that all new pages use the correct namespace, and will also set the name of the binary file generated when you compile the project.
- Solution Explorer > Epic.Training.Example.Web.Pages > Right Click > Properties
15. Change Assembly name and Default namespace so they match the project:
- *Epic.Training.Example.Web.Pages*

#### Path: Project Properties > Application



The application properties of the *Training.Example.Web.Pages* project



It is important that you complete the previous step before adding additional content that includes .NET classes, otherwise **those classes will not use the correct namespace**.

16. Click the **Web** tab, and then click the **Enable Edit and Continue** checkbox if it is not already checked.
- This option is not available if you are using Visual Studio 2015
17. Switch to the **Build Events** tab.
18. In the **Post-build event command line**, add the following command:

```
rundll32.exe InetCpl.cpl, ClearMyTracksByProcess 8
```



This command line clears IE's cache whenever the web application is built. This ensures that stale versions of files are not cached in IE, preventing you from seeing your changes.

This shouldn't be a problem in the HS Web Framework because steps are taken to prevent page caching, but you should consider adding this line in other web applications.

Here are other codes you can use to delete IE information (8 is used in the example above):

- 1 (Deletes History Only)
- 2 (Deletes Cookies Only)
- 8 (Deletes Temporary Internet Files Only)
- 16 (Deletes Form Data Only)
- 32 (Deletes Password History Only)
- 255 (Deletes ALL History)

19. Save the changes and close the properties tab.

### Part 3: Add additional files and test the site

---

1. Create a folder for images

- Solution Explorer > Training.Example.Web.Pages > Right Click > Add > New Folder > images

2. Add files to the images folder:

- images > Right Click > Add > Existing Item
- **File Name:** I:\Internal\Advanced\Web Tech Camp\Big-Mouse\images

3. Select all the images

4. Click Add

5. Add a blank html page. This will be the customer service page:

- Solution Explorer > Training.Example.Web.Pages > Right Click > Add > New Item > Web > HTML Page > Customer.htm

6. Set Customer.htm as the startup page:

- Solution Explorer > Customer.htm > Right Click > Set as Start Page

7. Add some text so that you have something to see on the page.

- The text **Hello World!** is a good choice. Place it inside the body element, as shown below:

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title></title>
</head>
<body>
    Hello World!
</body>
</html>
```



We will discuss the content of this file in more detail in the following section.

8. Run your site (press F5).
9. You should see a blank page in your default browser. You will also notice that ASP.NET has started IIS Express because an additional icon will appear in the Windows system tray.



Closing the browser will end the debugging session. It will also stop the development server, if you checked "Enable Edit and Continue" on the web tab of your project's properties.

If you did not enable edit and continue and you need to stop the server, right click on the system tray icon and select stop.

10. Your default browser should start up, with **Hello World** displayed.

## Part 4: Running on other browsers

While developing, you will need to test your page in a variety of browsers.

1. Follow the browser-setup instructions:
  - [http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting\\_Started/Browsers](http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting_Started/Browsers)
  - If you do not do this step, your debugging experience will not be an enjoyable one. Follow each step, particularly for IE, very carefully.
2. Right click *Customer.htm* in the solution explorer, and click **Browse With**.
  - The **Browse With** option will not be available if your application is already running.
3. Set Internet Explorer as your default browser, then select Google Chrome and click **Browse**.
  - Chrome may not appear on the list by default. If that is the case, get the target from the properties of the chrome shortcut and add it to the list
  - Alternatively you can select the browser to use with the toolbar in Visual Studio.



**For exercises in this class, you should test your site in IE and Chrome.**

- Not all features, such attached JavaScript debugging, are available when running from browsers other than Internet Explorer.

# Creating Basic Client Content

In this section we will cover the basics of adding HTML content to a web page, including the most critical HTML elements and some corresponding ASP.NET controls.

## Customer Service Page Content

The Big-Mouse Cheese Factory wants you to create a customer service page that includes the following content:

- Standard header & footer
- Menu area /w link home
- Welcome message
- Contact information
- E-mail form

You determine that HTML is the markup language that you want to use to get the job done.

## Markup Languages

A markup language is a system for annotating parts of a document to provide additional context for those parts. For example, one part could be marked as the title of the document, another as a heading, and yet another as a paragraph. The markup code must be syntactically distinguishable from the text so that it is not presented to the user.



The idea and terminology evolved from the "marking up" of paper manuscripts, i.e., the revision instructions by editors, traditionally written with a blue pencil on authors' manuscripts.

For more details on markup languages, see:

[https://en.wikipedia.org/wiki/Markup\\_language](https://en.wikipedia.org/wiki/Markup_language)

```
<part1> Some Content </part1>
<part2 id="2">More Content</part2>
<last id="3" />
```

Element

Used to mark parts of content:

	<part1> Some Content </part1>
Tag	Delimits the start and the end of an element. In the following example, <part1> is the <b>begin tag</b> and </part1> is the <b>end tag</b> of the part1 element: <part1>Some Content</part1>
Attribute	Extra metadata of content. In the following example, id is an attribute of the part2 element: <part2 id="2">More Content</part2>
Closed notation	Elements that never contain content can be written in closed notation, where the beginning tag of an element is terminated with ">", and there is no ending tag: <last id="3" />



### Always place quotes around attribute values

There are situations in markup languages where the quotes around an attribute value are optional. The Epic recommendation (for consistency) is to always use quotes, even if they are optional.

## Why use markup?

- Fast to write
- Descriptive
- Easy to parse
- Decouples content from how it is used

## Extensible Markup Language (XML)

XML is a popular markup language that allows the programmer to define their own elements, attributes and document structure.

The goal of XML is to create documents that can be easily read and understood by both humans and computers. Originally XML was created for documents, but it is also widely used to in other contexts, such as databases and web services.

The structure of an XML document is optionally defined in a separate file called the Document Type Definition (DTD), which is referenced from the XML file using the special <!DOCTYPE> element. For example:

Example XML with DTD:

```
<?xml version="1.0"?>
<!DOCTYPE friends SYSTEM "friends.dtd">
<friends>
  <person>
    <name>Sandy Summers</name>
    <dob>07/14/1984</dob>
  </person>
</friends>
```

Example **friends.dtd**:

```
<!ELEMENT friends (person*)>
<!ELEMENT person (name, dob?, sex?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT dob (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
```

XML files that do not include a DTD reference have no restrictions on their structure.



You are not expected to write DTDs as part of this course. If you are interested in learning more, see:

- [http://www.w3schools.com/xml/xml\\_dtd\\_intro.asp](http://www.w3schools.com/xml/xml_dtd_intro.asp)

For more information on the <!DOCTYPE> element, see:

- [http://www.w3schools.com/tags/tag\\_DOCTYPE.asp](http://www.w3schools.com/tags/tag_DOCTYPE.asp)
- <http://www.blooberry.com/indexdot/html/tagpages/d/doctype.htm>

## HTML is similar to XML

---

HTML is a markup language used to represent content on the web. It is based on XML, but has a specific DTD which provides a predefined set of elements, attributes and structure.

The following is an example HTML document:

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    Hello world!
  </body>
</html>
```

## Epic's HTML Conventions

Epic has its own conventions for HTML beyond what is enforced by the DTD. When there are two choices for an element that produces the same result, Epic convention may favor one element over another.

For example, the **b** element marks bold text: `<b>bold text</b>`, but the more modern tag is **strong**: `<strong>strong text</strong>`. They appear the same when text is displayed visually, but when you listen to a screen reader speak a word, the term bold doesn't make as much sense.

Other conventions include:

- Tags must always be written in lower case
- All tags must be closed
- Tags may not be interlaced
  - Bad: `<tag1><tag2></tag1></tag2>`
  - Good: `<tag1><tag2></tag2></tag1>`

## HTML DTDs

For HTML5, the DTD should be: `<!DOCTYPE html>`

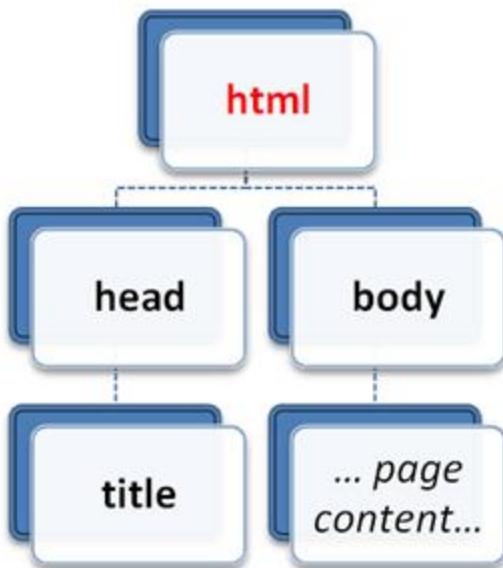


If you omit the `<!DOCTYPE>`, then certain browsers will not render your page correctly.

## Page Tree

The following elements are required by an HTML page:

## Page Tree



html	The outermost (root) element. <ul style="list-style-type: none"><li>Contains all tags other than DTD</li></ul>
head	Location for metadata that is not visible on the page
title	The name of the page that appears as the browser title and browser-tab caption.
body	Location for content that is displayed on the page

## Exercise 2: Create a Simple HTML Page

In this activity you will begin adding content to the customer-service page. In the process, you will learn about many of the elements and attributes available in HTML.

### Part 1: Overview

Recall the requirements for the customer-service page. It must include:

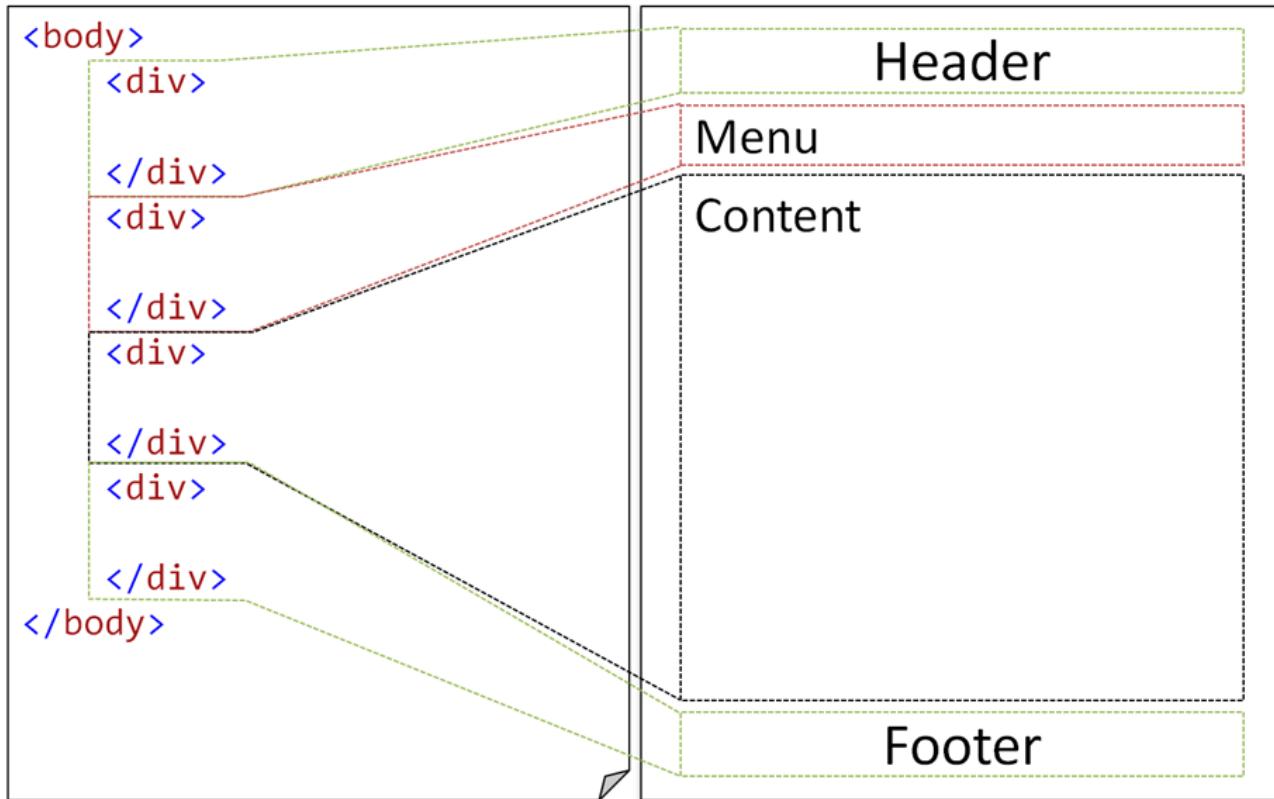
- A standard header & footer
- A menu area with a link home
- A welcome message

- Contact information for various countries
- An e-mail form

You will meet the first three requirements during this exercise.

## Part 2: Dividing the Page Vertically

The page should be divided vertically into four parts. This can be accomplished using the `div` element, as shown in the following diagram:



div	A generic block element used to divide a page.  The <code>div</code> element is considered "generic" because it has no special meaning other than to divide the page.
Block Element	Any element that is arranged vertically by default.

1. Remove *Hello World!* from the `body` tag of `Customer.html`.
2. Insert 4 consecutive `div` elements within the `body` element.
3. Because one `div` looks much like another, it would be useful to insert comments into each one describing what it is used for. HTML comments have the following syntax (this is the same syntax used in XML files):

```
<!-- comment -->
```

4. Insert a comment into each div element describing what it is used for:

```
<body>
  <div>
    <!--header-->
  </div>
  <div>
    <!--menu-->
  </div>
  <div>
    <!--content-->
  </div>
  <div>
    <!--footer-->
  </div>
</body>
```

5. Run your web application. What do you see?

- Your page will appear blank because div elements are transparent by default, and comments are never displayed.

6. To view the content of your page (in IE), perform the following steps:

- a. Add the menu bar to IE: **right click (area to the right of browser tabs) > Check Menu Bar**
- b. select **Menu Bar > view > source**



Notice that standard HTML comments are not stripped from the page when it is sent to the client. This means they take up space, network bandwidth, and are visible to users when they view the source of the page.

Keep these points in mind when using this kind of comment.

7. Close the source view and stop debugging when you are done.

### Part 3: Headings and Paragraphs

Headings are numbered elements of increasing specificity, from 1 to 6. Paragraphs are collections of related text. Both headings and paragraphs are block elements.

1. Place the following headings into the content portion of *Customer.html*.

```
<div>
```

```
<!--content-->
<h1>Customer Service</h1>
<h2>Welcome</h2>
<h2>Contact Us</h2>
<h3>Phone/Mail</h3>
<h3>E-mail</h3>
</div>
```

2. Try moving the `<h3>E-mail</h3>` to the top of the list of headings, then run your site. Do you see any warnings or errors?



**Always order your headings so they make sense hierarchically**

Even though it doesn't make logical sense for a higher-numbered heading to be listed before a lower-numbered heading, the order is not enforced by HTML.

3. Move `<h3>E-mail</h3>` back to the end of the headings list. The resulting page should appear similar to the following:

# Customer Service

## Welcome

## Contact Us

## Phone/Mail

## E-mail

*Customer Service Page with Headings*

4. Under the Welcome heading, add a couple of paragraphs of text. You can find some sample text in the following file:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\Customer.txt

```
<h2>Welcome</h2>

<p>Welcome to the Big Mouse Cheese Factory's customer service page! As
```

our valued customer, your concerns are very important to us.</p>

<p>The Big Mouse Cheese Factory is a multinational company and we strive to provide the best service for all of our customers. If you would like to speak with a customer representative, use the contact information below corresponding to your country of business, or send us an e-mail.</p>

- ! • Like the `div` element, the `p` element is a block element that arranges text vertically.
- Unlike `div` elements, `p` elements may **not** contain other block elements.

5. Content can be emphasized (italicized) using the `em` element or bolded using the `strong` element. Emphasize and bold the corresponding parts in *Customer.html*:

```
<h2>Welcome</h2>
```

```
<p>Welcome to the Big Mouse Cheese Factory's customer service page! As our <em>valued customer</em>, your concerns are <strong>very</strong> important to us.</p>
```

### Inline Element

An element that is rendered in line with the content, and may contain other inline elements.

Inline elements may not contain block elements.

! If you are familiar with web development, then you may have heard of the italics (`<i></i>`) and bold (`<b></b>`) elements. **These tags are should not appear in Epic-written HTML.**

## Part 4: Images

There are two main image formats used at Epic:

### JPEG

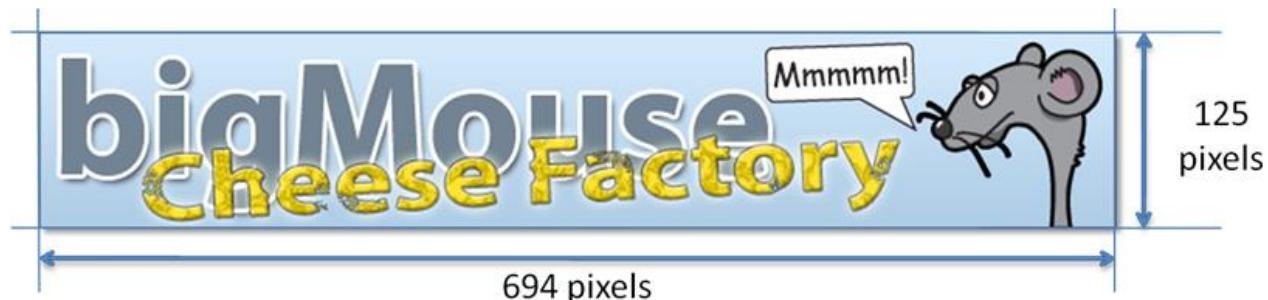
#### Epic's choice for photographs

JPEGs use lossy compression, meaning that the higher the compression rate, the more information is lost. This introduces noise into the image in the form of visual artifacts. However, this noise is difficult to see in most photographs, even at relatively high

	compression rates.
PNG	<p><b>Epic's choice for most non-photograph images</b></p> <p>PNs use lossless compression so that the displayed image appears exactly like the original. Unlike JPEGs, PNs support transparency.</p>

1. Add the following image to the header `div` of *Customer.html*:

- `images/banner.png`



```
<div>
    <!-- header -->
    
</div>
```

2. Run your site and verify that the image appears as expected.

3. Read about the attributes of the `img` element:

<code>src</code>	Indicates path of the image to display, relative to the location of the page.
<code>alt</code>	<p><b>This attribute is required by HTML for all <code>img</code> tags</b></p> <p>Indicates alternate text that is displayed when the image is unavailable or is still downloading.</p> <p>Alternate text is also read by screen readers. A screen reader is an accessibility tool used by those with sight disabilities.</p>
<code>width / height</code>	Used to specify the dimensions of the image.

4. Why do you suppose it is important to always specify a height and width for your images (assuming that you know them ahead of time)?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

5. What does it mean for the `img` to have a start tag with `/`, but to not have an end tag?

- \_\_\_\_\_

6. Change the `src` attribute so that it points to a non-existent file, and then run the site again. Do you see the alternate text?

- Change `src` back to the correct path when you are done.



An `img` tag showing the alternate text

## Part 5: Lists and Links

There are 2 kinds of lists: ordered (numbered) and unordered (bulleted). Each type has a corresponding HTML element:

<code>ul</code>	An element that specifies an unordered list.
<code>ol</code>	An element that specifies an ordered list.
<code>li</code>	An element that specifies a list item. Appears numbered in <code>ol</code> elements and bulleted in <code>ul</code> elements.

1. Add the following list to your menu div. You will make them hyperlinks shortly:

- Home
- Customer Service
- View Inventory

- Place Order

```
<div>
    <!-- Menu -->
    <ul>
        <li>Home</li>
        <li>Customer Service</li>
        <li>View Inventory</li>
        <li>Place Order</li>
    </ul>
</div>
```



Nesting an ordered or unordered list element within another list element will result in an indented sub-list.

For example:

```
<ol>
    <li>First list item</li>
    <li>Second list item</li>
    <ul>
        <li>First sub-list item</li>
        <li>Second sub-list item</li>
    </ul>
</ol>
```

Will render as:

1. First list item
2. Second list item
  - First sub-list item
  - Second sub-list item

The anchor tag is used to link to a location in the same page, or on a different page. The anchor element has the following attributes:

href	The <b>hypertext reference (href)</b> attribute of the anchor tag is used to specify the destination of a link.
title	The title is a description of where the link goes. For sighted users, the title is rendered as a tooltip. The title is also read by screen readers for the visually impaired.



Do not include "Link to ..." or "This is a Link to ..." in the title because screen readers do this automatically. There is no need to make our application stutter.

3. Convert the first two list item elements in the menu div to hyperlinks:

```
<ul>
    <li><a href="Default.aspx" title="Return home">Home</a></li>
    <li>
        <a href="Customer.html" title="View customer service page">
            Customer Service
        </a>
    </li>
    <li>View Inventory</li>
    <li>Place Order</li>
</ul>
```

To link to a specific location within a page, first assign an ID to an element on that page using the id attribute. Next, set href="#idValue". For example, the customer service link could link to the h1 element in *Customer.html* if an id is assigned to that element:

#### Link to:

```
<h1 id="hMain">Customer Service</h1>
```

#### using either (within *Customer.html*):

```
<a href="#hMain" title="View customer service page">
    Customer Service
</a>
```

#### or (within or outside *Customer.html*):

```
<a href="#">Customer.html#hMain" title="View customer service page">
    Customer Service
</a>
```



An anchor tag with an id and no other attributes is commonly referred to as a *named anchor*. Named anchors are useful for marking points within a page that can be jumped to (with hyperlinks).

4. Add an `id` attribute to the `h1` element with the content *Customer Service*. Use `hMain` as the `id`.
5. Rather than linking just to the customer service page from the menu, link directly to the element with id `hMain`. Use the form of the link that includes the page as well as the id of the element on the page.
6. Run your site and verify that the links work. You may need to reduce the size of your browser window so that clicking causes the window to scroll.
7. Try placing a link around the banner image:

```
<a href="Default.aspx" title="Return home">
    
</a>
```

8. Run your page. How does the image appear differently? Why doesn't the link work?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

## Part 6: Special Characters

Certain characters have a syntactic meaning in HTML, or can't be directly entered for other reasons. These characters need to be encoded to appear as expected on the page. The following table shows some common characters and the corresponding code:

Character	Code
<	&lt;
>	&gt;
&	&amp;
"	&quot;
<b>Non-breaking space</b>  Used to add a space that prevents a line break between consecutive words / inline elements.	&nbsp;

Character	Code
©	&copy;

1. Add the copyright message to the footer of *Customer.html*:

```
<div>
    <!--footer-->
    Copyright &copy; 2008-<current year> Big Mouse Cheese Factory
</div>
```

2. Run your site and verify that the copyright message is displayed as expected.
3. Answer the wrap-up questions.

## Wrap Up

---

Your page should appear similar to the following:



- [Home](#)
- [Customer Service](#)
- View Inventory
- Place Order

## Customer Service

### Welcome

Welcome to the Big Mouse Cheese Factory's customer service page! As our *valued customer*, your concerns are very important to us.

The Big Mouse Cheese Factory is a multinational company and we strive to provide the best service for all of our customers. If you would like to speak with a customer representative, use the contact information below corresponding to your country of business, or send us an e-mail.

### Contact Us

#### Phone/Mail

#### E-mail

Copyright © 2008-<current year> Big Mouse Cheese Factory

*Content of the Customer Service Page*

1. Can heading elements be used out of order? Should they?

■

---

2. Can the p element contain other block elements?

■

---

3. What should you use instead of b and i elements?

■

---

4. Why do you suppose it is important to always specify a height and width for your images (assuming that you know them ahead of time)?

■

---

5. What does it mean for img to have a start tag with /, but not to have an end tag?

■

---

6. What is alt text used for in addition to replacing the image if it can't be loaded (or is loading slowly)?

■

---

# Displaying Consistent Content

In this part you will learn the proper way to share content between different pages in your web application.



HyperspaceWeb has its own method of sharing content, but the general idea is the similar to what you will learn in this lesson.

Many other web applications do use the techniques covered here.

Examine the following screenshot of the customer service page and circle the parts that should be shared between pages:



- [Home](#)
- [Customer Service](#)
- View Inventory
- Place Order

## Customer Service

### Welcome

Welcome to the Big Mouse Cheese Factory's customer service page! As our *valued customer*, your concerns are very important to us.

The Big Mouse Cheese Factory is a multinational company and we strive to provide the best service for all of our customers. If you would like to speak with a customer representative, use the contact information below corresponding to your country of business, or send us an e-mail.

### Contact Us

#### Phone/Mail

#### E-mail

Copyright © 2008-<current year> Big Mouse Cheese Factory

*Content of the Customer Service Page*



What should you share?

---



---



Why not copy the duplicate content on each page?

---

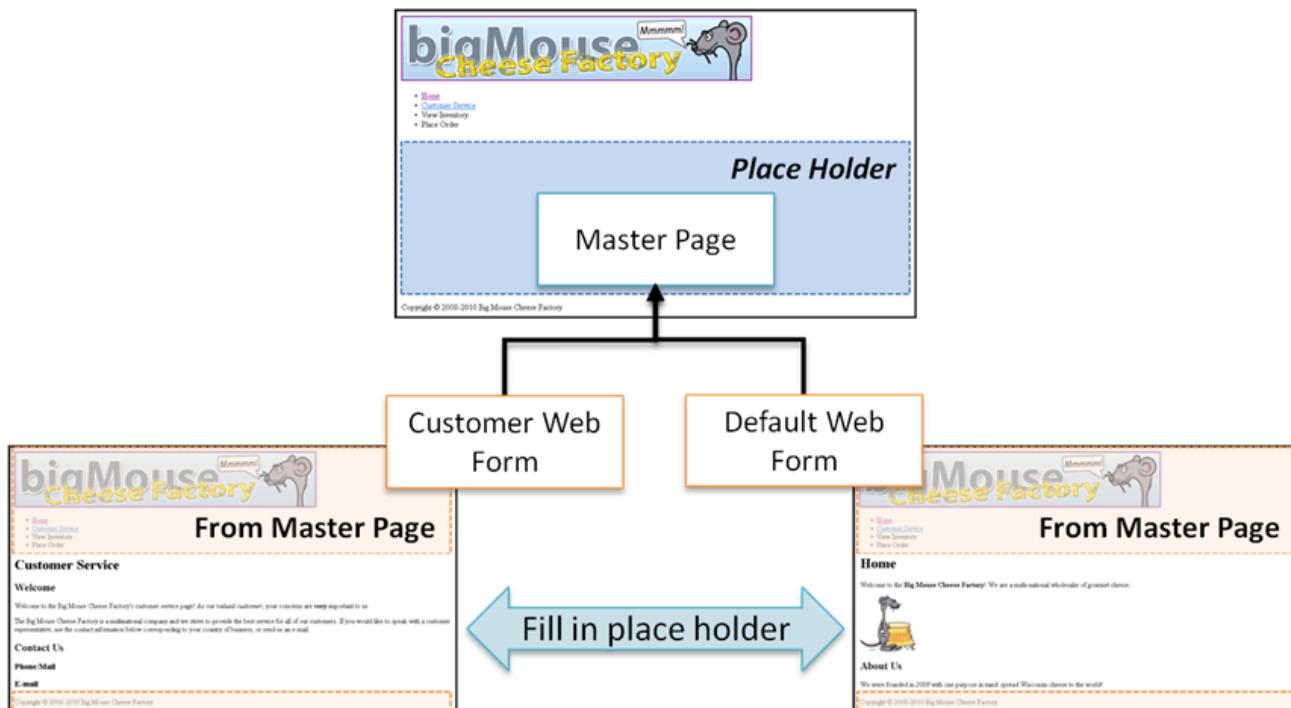


---

## Share Content Using a Master Page

ASP.NET Web Forms have a special page called a master page. Think of this like a template that pre-defines some content and provides placeholders for other pages to fill in more details.

In the following diagram, the master page defines the content contained in the header, menu and footer, and has a placeholder for the main content of the page. Notice that the customer and default web forms each provide different content for the placeholder:



## Exercise 3: Share Content Between Pages

In this activity you will create a master page to share content between pages. Next, you will create the default home page that users will initially see when visiting your site.

## Part 1: Creating a Master Page

Often, the same content is duplicated on several pages. One way to share this content from a single source is through the use of a master page.

1. Create a master page:
  - a. **Solution Explorer > Training.Example.Web.Pages > Right Click > Add > New Item > Web Forms Master Page**
2. Name: *ExampleMaster*
3. Click **Add**
4. Make the following modifications to *ExampleMaster*:
  - a. Change the title to *Big Mouse Cheese Factory*.
  - b. Remove all elements other than `title` from `head`.
  - c. Clear all elements out of the `body`.
5. The resulting content of the master page should appear as follows:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeBehind="ExampleMaster.master.cs"
Inherits="Epic.Training.Example.Web.Pages.ExampleMaster" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <title>Big Mouse Cheese Factory</title>
</head>
<body>
</body>
</html>
```

6. Copy all text from the body of Customer.htm and paste it into the body of ExampleMaster.
7. Inside *ExampleMaster*, clear everything other than the comment from the content div.
  - Do not delete anything from the header, menu or footer divs (see below):

```
<body>
    <div>
        <!--header-->
        <a href="Default.aspx" title="Return home">
            
        </a>
    </div>
```

```
<div>
    <!-- Menu -->
    <ul>
        <li><a href="Default.aspx"
            title="Return Home">Home</a></li>
        <li>
            <a href="Customer.htm#main"
                title="View customer service page">
                Customer Service
            </a>
        </li>
        <li>View Inventory</li>
        <li>Place Order</li>
    </ul>
</div>
<div>
    <!-- Content -->
</div>
<div>
    <!-- Footer -->
    Copyright © 2008-2011 Big Mouse Cheese Factory
</div>
</body>
```

8. Master pages use *content placeholders* to define where pages that inherit from the master page may insert new content. To allow pages to add content to the content div, add a place holder as shown below:

```
<div>
    <!-- content -->

    <asp:ContentPlaceHolder ID="cphContent" runat="server">

    </asp:ContentPlaceHolder>

</div>
```



You may add the *ContentPlaceholder* by hand, or from the toolbox under the Standard heading. If you do not see the toolbox, verify that your solution is not running, then do the following:

- **Main Menu > View > Toolbox (Ctrl + Alt + X)**

You may either edit the ID attribute directly in the code, or through the properties window. If you do not see the properties window, then do the following:

- **Main Menu > View > Properties Window (F4)**

9. Save changes to *ExampleMaster*
10. Create a new page based on the master page:
  - **Solution Explorer > Training.Example.Web.Pages > Right Click > Add > New Item > Web Form with Master Page**



If you can't locate **Web Form With Master Page**, look for **Content Page**.

Depending on what combination of different versions of Visual Studio you have installed, the item template names may vary.

11. Name: *Customer.aspx*
12. Click **Add**
13. Select a Master Page: *ExampleMaster*
14. Click **OK**
15. Copy the text inside the content div from *Customer.htm* and paste it into the `asp:Content` element in *Customer.aspx*:

```
<asp:Content ID="Content1" ContentPlaceholderID="cphContent"
    runat="Server">

    <h1 id="hMain">Customer Service</h1>

    <h2>Welcome</h2>

    <p>Welcome to the Big Mouse Cheese Factory's customer
    service page! As our <em>valued customer</em>, your concerns are
    <strong>very</strong> important to us.</p>

    <p>The Big Mouse Cheese Factory is a multinational
    company and we strive to provide the best service for
    all of our customers. If you would like to speak with a
    customer representative, use the contact information
    below corresponding to your country of business, or send
```

```
us an e-mail.</p>

<h2>Contact Us</h2>

<h3>Phone / Mail</h3>

<h3>E-mail</h3>

</asp:Content>
```

16. Change the customer service link in the menu div of *ExampleMaster* to point at *Customer.aspx* rather than *Customer.htm*.
17. Delete *Customer.htm*:
  - **Solution Explorer > right click Customer.htm > delete**
18. Set *Customer.aspx* as the start page
  - Right-click > Set As Start Page
19. Run the web application
20. Verify that it appears similar to the following:



- [Home](#)
- [Customer Service](#)
- View Inventory
- Place Order

## Customer Service

### Welcome

Welcome to the Big Mouse Cheese Factory's customer service page! As our *valued customer*, your concerns are very important to us.

The Big Mouse Cheese Factory is a multinational company and we strive to provide the best service for all of our customers. If you would like to speak with a customer representative, use the contact information below corresponding to your country of business, or send us an e-mail.

### Contact Us

#### Phone/Mail

#### E-mail

Copyright © 2008-<current year> Big Mouse Cheese Factory

*Content of the Customer Service Page*

## Part 2: Creating the Home Page

1. Add a new *Web Form with Master Page* named *Default.aspx*. Make sure to use *ExampleMaster* as the master page.
2. Add content to this page that meets the following requirements:
  - Include sections for *Home*, and *About Us*. You may create any text that you like.
  - Minimally use all of the following elements at least once:
    - *h1, h2, p, em, strong*
    - *img use images/BigMouse-128x128.png*
3. Set *Default.aspx* as the start page.
4. Run your web application and verify that it appears as expected.

- For example:

The screenshot shows the homepage of the Big Mouse Cheese Factory. At the top, there's a large logo with the text "bigMouse" in grey and "Cheese Factory" in yellow. To the right of the text is a cartoon mouse with a speech bubble saying "Mmmmm!". Below the logo is a navigation menu with links: Home, Customer Service, View Inventory, and Place Order. The main content area has a heading "Home" and a sub-section "About Us" with text about the company's founding in 2008. There are also copyright and footer information.

- [Home](#)
- [Customer Service](#)
- [View Inventory](#)
- [Place Order](#)

## Home

Welcome to the **Big Mouse Cheese Factory!** We are a multi-national wholesaler of gourmet cheese.

## About Us

We were founded in *2008* with one purpose in mind: spread Wisconsin cheese to the world!

Copyright © 2008-2011 Big Mouse Cheese Factory

*Default.aspx*

5. Answer the wrap-up question.

### Wrap Up

How could you add page-specific options to the menu div?

- 

### If you have time

Read more about Master pages in the following tutorials:

<http://www.asp.net/master-pages/tutorials>



# Including Tables and Forms

Applications often need to display complex information as tables, and HTML has special elements for this purpose. Collecting user input is another common feature of applications, which is accomplished in HTML using form elements.

## Scenario

In this section you will complete the customer service page by adding a table of contact information, and a form to send emails to the customer-service team.

## Creating Tables

Suppose you want to display contact information on the customer service page in a table:

Country	Phone	Address
US	608-271-9000	1979 Milky Way Verona, WI 53593

The HTML to do so is:

```
<h3>Phone/Mail</h3>
<table>
  <tr>
    <th>Country</th><th>Phone</th><th>Address</th>
  </tr>
  <tr>
    <td>US</td>
    <td>608-271-9000</td>
    <td>1979 Milky Way<br />
      Verona, WI 53593</td>
  </tr>
</table>
```

table An element used to define an HTML table. Can directly contain tr (row elements)

tr Element used to specify a table row. Directly contains th (table header)

	or <code>td</code> (table data) elements.
<code>th</code>	Element used to specify a <b>table header</b> cell. Can be either a row or column header.
<code>td</code>	Element used to specify a <b>table data</b> cell.
<code>br</code>	Element used to specify a line <b>break</b> . <ul style="list-style-type: none"><li>• Closed notation is required in HTML: <code>&lt;br /&gt;</code></li><li>• This is an <b>inline element</b>, even though it renders to a line break.</li></ul>

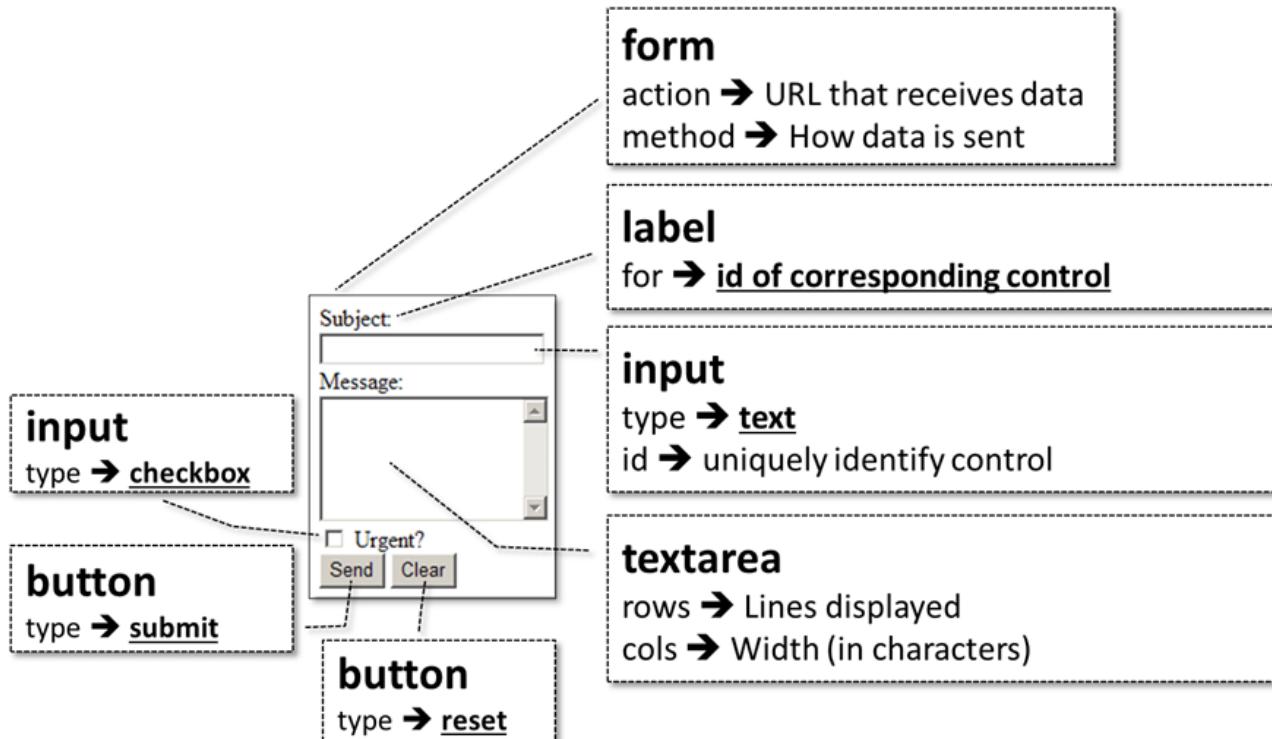


There are more details about tables, such as how to span multiple rows/columns, and how to specify which headers correspond to a given cell. The quiz covers this information, but it is not included in the lecture or workbook.

You are expected to study the reference wiki during lab for this kind of information:  
[wiki > Foundations > Training > WTC > Reference > HTML > Tables](#)

## Creating Forms

Forms are the primary way web pages use to collect user input. The following is a simple example of a form that you'll add to the customer-service page:



### Example Form Elements

```
<h3>E-mail</h3>
<form action="Customer.aspx" method="post">
    <label for="txtSubject">Subject:</label><br />
    <input type="text" id="txtSubject" /><br />
    ...
</form>
```

! Epic prefers using the button element because it is self documenting

Experienced web programmers may know that `<input type="submit">` is identical to `<button type="submit">` in terms of how it is rendered in a browser.

Element	Use	Attributes
form	Container for HTML controls	<b>action:</b> Page that receives the data from the form <b>method:</b> How the data is sent to the target page

Element	Use	Attributes
		<ul style="list-style-type: none"> <li><b>get:</b> Data is passed via the query (request) string</li> <li><b>post:</b> Data is sent in the body of the request.</li> </ul>
input	<code>type="text": textbox</code> <code>type="checkbox": checkbox</code>	<b>value:</b> The value contained in the input control.
button	<code>type="submit":</code> Button that causes the form to send data to the server. <code>type="reset":</code> Button that resets form to default values	
textarea	Control for multi-line text	<b>rows, cols:</b> Specifies the control size

## Exercise 4: Creating Tables and Forms

---

In this activity you will practice creating tables and forms.

### Part 1: Tables

---

- Add the following table beneath the Phone/Mail heading of *Customer.aspx*:

Country	Phone	Address
US	608-271-9000	1979 Milky Way Verona, WI 53593
UK	44-203-002-4300	London, United Kingdom, SE1 2BY 5 More London Place
France	33-1-4448-5600	18 Avenue de Suffren Paris, France, 75015

- To get you started, the first two rows are provided:

```
<h3>Phone/Mail</h3>
```

```
<table>
  <tr>
    <th>Country</th><th>Phone</th><th>Address</th>
  </tr>
  <tr>
    <td>US</td>
    <td>608-271-9000</td>
    <td>1979 Milky Way<br />
      Verona, WI 53593</td>
  </tr>
</table>
```

## Part 3: Forms

1. Add a form just below the *Email* heading that includes the control for entering the subject:

```
<h3>E-mail</h3>
<form action="Customer.aspx" method="post">
  <label for="txtSubject">Subject:</label><br />
  <input type="text" id="txtSubject" /><br />
</form>
```

2. Open *Customer.aspx.cs* by expanding the files beneath *Customer.aspx* in the solution explorer.
3. Set a break point on the opening { of *Page\_Load*.
4. Run your site and navigate to *Customer.aspx*. It should break on the page load.
5. Continue loading the page (press **F5**).
6. Enter some text into the subject textbox and press enter. What happened?

■

---

When a form is sent back to the same page that it came from (using the post method), this is known as "posting back". Traditional ASP.NET applications use post back as the primary method of communicating with the server. Epic will not use this method, for reasons we will discuss in more detail later. The primary reason is that it refreshes the entire page, which is inefficient.



If a form only contains one textbox, then pressing enter when that control has focus will initiate a post back (even if there is no default submit button in the form). **This is a special corner case of HTML forms that you should be aware of.**

7. Add the remaining controls to the form:

```
<label for="areaMessage">Message:</label><br />
<textarea id="areaMessage" rows="5" cols="17"></textarea><br />
<input id="chkUrgent" type="checkbox" />
<label for="chkUrgent">Urgent?</label><br />
<button id="btnSend" type="submit" title="Send message">Send</button>
<button id="btnClear" type="reset" title="Clear form">Clear</button>
```

8. Refresh the page.
9. Try clicking the two buttons. Did either one initiate a post back?

■

- 
10. Remove the `Page_Load` break point.
  11. With the website running, click on the labels. What happens?

■

---

For the situation where you want the user to choose between a discrete set of options, the HTML element `select` should be used. The `select` element may contain zero or more `option` elements. For example:

```
<select id="selExample">
    <option value="val1" selected="selected">Default option</option>
    <option value="val2">Second Option</option>
</select>
```

The value displayed to the user is listed between `<option>...</option>`, while the value returned to the server (or accessed in JavaScript) is in the `value` attribute.

Use the `selected` attribute to indicate the option selected by default.

12. So that the message can be sent to the proper recipient, add an option for the user to select their country. This should be the first field that they need to fill out. The options are:
  - United States (default, `value="US"`)
  - United Kingdom (`value="UK"`)
  - France (`value="FR"`)
13. Run your website. Does the form appear as expected?

14. Try setting the `size` attribute of the `select` element to values between "1" and "3". How does this change the control's behavior? What is the default value?
15. Try setting the `multiple` attribute of the `select` element to "multiple". What happens when you **CTRL+CLICK** different values in the select box?
16. Remove the `multiple` and `size` attributes from the select control.
17. Continue on to the *if you have time* section.

## Wrap Up

---

The Contact and E-mail sections of your customer service page should appear similar to the following:

## Contact Us

### Phone/Mail

Country	Phone	Address
US	608-271-9000	1979 Milky Way Verona, WI 53593
UK	44-203-002-4300	London, United Kingdom, SE1 2BY 5 More London Place
US	33-1-4448-5600	18 Avenue de Suffren Paris, France, 75015

### E-mail

Country:

Subject:

Message:

Urgent?

*Contact and E-mail Sections of Customer.aspx*

1. What happens when you click a label?

■

---

2. How is size="1" different from size > 1?

■

---

3. What does the `multiple` attribute do?

■

---

## If you have time

---

There are some optional table elements that may be useful from time to time: `thead`, `tbody`, and `tfoot`. These elements are especially useful when working with tables in JavaScript, because JavaScript adds them automatically, even if you don't specify them explicitly.

1. Place a `thead` element around the first row of the contact information table.
2. Place a `tbody` element around the remaining rows.
3. Insert a `tfoot` element that spans all three columns with the following message:

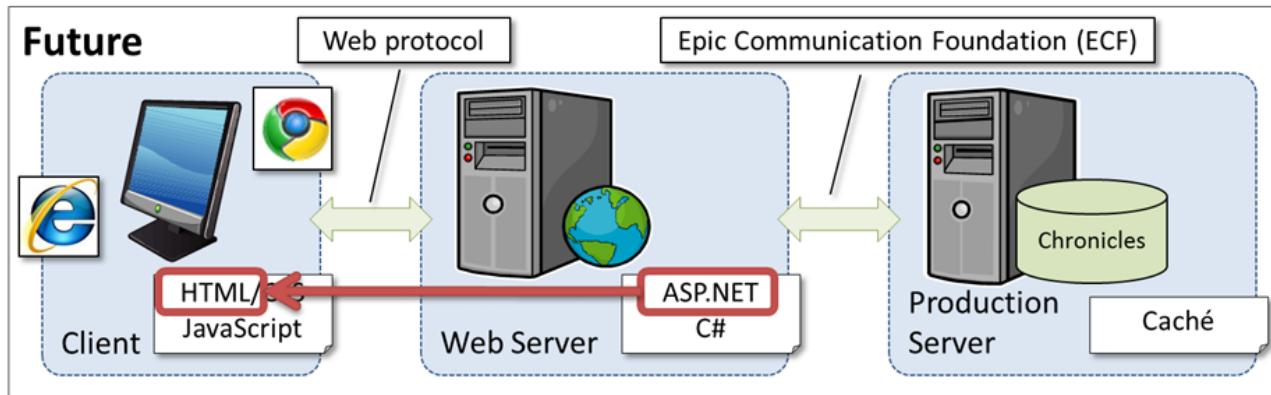
```
<tfoot>
  <tr><th colspan="3">We do not have office locations in other
countries</th></tr>
</tfoot>
```

4. Spend some time looking through the following page:

[http://www.w3schools.com/html/html\\_forms.asp](http://www.w3schools.com/html/html_forms.asp)

# Creating Server Content

Processing content on the server gives you the opportunity to modify the content of the page at run-time before it is sent to the client. For example, you can add or remove sections of the page that aren't applicable given the current context of the user, department and security classes.



Client Elements (i.e., HTML)	Server Controls (i.e., ASP.NET)
<ul style="list-style-type: none"> <li>Not processed on the server</li> <li>Natively understood by browsers</li> </ul>	<ul style="list-style-type: none"> <li>Processed on the server</li> <li>Available in C# code-behind</li> <li>Must be rendered to client elements (HTML) before the browser understands them</li> </ul>

## Why create content in ASP.NET vs. HTML (or vice versa)?

- HTML requires less work for the server because it doesn't need to be rendered
- ASP.NET is more flexible, because you can adjust its content on the server before it is rendered and sent to the client.



There are many phases in the page life cycle and this lesson will introduce them as they are used. However, if you would like to have the big picture in mind before proceeding, [take some time to read through this MSDN article](#).

Note that HyperspaceWeb rarely uses the post-back model (depicted in the MSDN article as the Event handling phase) to process client events on the server because post-back is not efficient. Instead, client events are handled on the client in JavaScript. Details related to client scripting are covered in the **Designing Client Behavior** lesson. An explanation of why the post-back model is not efficient is provided in the **Implementing Business Logic** lesson.

## Exercise 5: Creating Server Content

In this exercise, you will create several additional pages using ASP.NET server controls that render to HTML, rather than writing the pages directly in HTML. The pages include:

- Customer Service (additional content)
- View Inventory
- Details/Edit Inventory
- Place Orders

### Part 1: CustomerService Page

1. You want to replace the HTML control *btnSend* with an identical ASP.NET control, but may want to undo this later. If you comment it out using `<!-- -->`, will it be completely removed from the client?

■

- 
2. ASP.NET has an alternate comment method that actually removes the contained markup from the page entirely. Comment out *btnSend* using `<%-- ... --%>`:

```
<%--<button id="btnSend" type="submit" title="Send message">Send  
</button>--%>
```

3. Highlight the comments and then press **CTRL+K** followed by **CTRL+U**. What happened?

■

- 
4. Keeping *btnSend* highlighted, press **CTRL+K** followed by **CTRL+C**. What happened?

■

- 
5. Ensure *btnSend* is commented out.
  6. On the customer service page, place an ASP.NET button directly below *btnSend*.
    - To view the toolbox, type **CTRL + ALT + X**
    - The button control is located here: **Toolbox > Standard > Button**
  7. Copy the following HTML attribute values from *btnSend* to the corresponding attributes in the new `asp:Button`:

HTML Attribute	Corresponding ASP.NET Attribute
<code>id="btnSend"</code>	<code>ID="btnSend"</code>
<code>&lt;button ...&gt;Send&lt;/button&gt;</code>	<code>Text="Send"</code>
<code>title="Send message"</code>	<code>ToolTip="Send message"</code>

8. Run the site and browse to *Customer.aspx*.

9. Note that **you will receive an error**:

### Server Error in '/' Application

Control 'cphContent\_btnSend' of type 'Button' must be placed inside a form tag with runat=server.



ASP.NET attempts to simplify its pages by wrapping all body content in a form element. This allows you to place form controls anywhere on the page. In the next several steps, you will move the form element to the master page and change it so that ASP.NET is aware of it.

10. Close the error page.

11. Leaving all other control elements where they are, remove just the form element from *Customer.aspx*:

```
<h3>E-mail</h3>
```

*Remove this line: <form action="Customer.aspx" method="post">*

```
<label for="selCountry">Country:</label><br />
<select id="selCountry">
<option value="US" selected="selected">United States</option>
<option value="UK">United Kingdom</option>
<option value="FR">France</option>
</select><br />
<label for="txtSubject">Subject:</label><br />
<input type="text" id="txtSubject" /><br />
<label for="areaMessage">Message:</label><br />
<textarea id="areaMessage" rows="5" cols="17"></textarea><br />
<input id="chkUrgent" type="checkbox" />
<label for="chkUrgent">Urgent?</label><br />
<%--<button id="btnSend" type="submit" title="Send
message">Send</button>--%>
```

```
<asp:Button ID="btnSend" runat="server" Text="Send" ToolTip="Send message" />
<button id="btnClear" type="reset" title="Clear form">Clear</button>

Remove this line: </form>
```

12. Switch to the master page
13. Keeping all other content the same, insert a new form element just inside the body element.
14. Add `runat="server"` as an attribute of the form element, so ASP.NET is aware of it while rendering the `asp:Button`:

```
<body>
  <form runat="server">
    ... do not change other tags in the body
  </form>
</body>
```

15. Run the site and browse to `Customer.aspx` again.
16. View the source of `Customer.aspx` in Internet Explorer (**View > source**). What attributes were added to `<form>` by ASP.NET?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

17. What additional `input` elements were added automatically by ASP.NET? They will be contained within new `div` elements, just below the opening `form` tag.

Element	<code>id</code> attribute	<code>type</code> attribute
input		hidden
input		hidden
input		hidden



These hidden elements store state on the client between requests to the server (i.e., postbacks). This is critical for traditional ASP.NET web applications, but HyperspaceWeb tends to avoid using postbacks because they are not efficient.

For more information on \_\_VIEWSTATE, see:

- <https://msdn.microsoft.com/en-us/library/ms972976.aspx>

For more information on \_\_EVENTVALIDATION, see:

- <https://msdn.microsoft.com/en-us/library/system.web.ui.page.enableeventvalidation.aspx>

18. Scroll down to the email controls and look at the input element with type *submit*. This is the HTML-rendered version of the ASP.NET Button that you added earlier. Write the value of the id attribute below.

Element	type Attribute	Server ID	Client id
input	submit	btnSend	

19. Remove the `runat="server"` from `btnSend`, then refresh the customer service page.  
20. View the source and then compare the result with how the button appears on the page. What happened?

■

---

21. Is `runat="server"` required for ASP.NET controls?

■

---

---

22. When you are done, add the `runat="server"` attribute back into the button.

## Part 2: Inventory Page

---

1. Stop your web application if it is still running
2. Create a new folder in the `Training.Example.Web.Pages` project called *Inventory*.



### Note that the folder is capitalized

This is because .NET classes automatically append their containing folder to the namespace. In order to follow namespace conventions, folders that contain .NET classes must be capitalized.

Eventually, all pages contained in this directory will be protected from access by anonymous users. Directories make it easier to manage security, so this is the preferred method for non-HyperSpaceWeb applications.

3. Create a directory called *images* within *Inventory*.

Now there are three categories of images. This folder will contain thumbnails and expanded pictures of inventory items.

The images folder can be in lower case because it will not be used as part of a namespace.

4. Add all images from the following path to the new images folder in *Inventory*:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\productImages

5. Within the *Inventory*, add a new *Web Form with Master Page* called *Details.aspx*.

- This page will contain an expanded view of an inventory item.

6. In *Details.aspx*, add "Details" as the level-1 heading (*h1* element).

7. Within *Inventory*, add a new *Web Form with Master Page* called *Catalog.aspx*.

- This page will contain a table of all inventory items

8. Add "Catalog" as the level-1 heading.

9. In *Catalog.aspx*, add an ASP.NET table just below the heading.

- Use the Table control from the Standard heading in the toolbox.

10. Give the table the ID *tblInventory*.

The ASP.NET equivalent controls for various table elements are listed below:

HTML	ASP.NET
tr	asp:TableRow, asp:TableHeaderRow
th	asp:TableHeaderCell
td	asp:TableCell

HTML	ASP.NET
img <ul style="list-style-type: none"><li>• alt</li><li>• src</li></ul>	asp:Image <ul style="list-style-type: none"><li>• alternateText</li><li>• ImageUrl</li></ul>
a <ul style="list-style-type: none"><li>• href</li><li>• title</li></ul>	asp:Hyperlink <ul style="list-style-type: none"><li>• NavigateURL</li><li>• ToolTip</li></ul>

11. Add the following rows to the table using ASP.NET controls:

Name	Image	Price	Details
Blue		90.00	<a href="#">View</a>

~Inventory/images/Blue-Thumb.png

Link to ~Inventory/Details.aspx

- The goal is to use the ASP.NET equivalent of tr, td, img and so on.
- Another important point is that ASP.NET controls that are part of a larger control will not need their own runat="server" attributes. This includes asp:TableRow, asp:TableHeaderRow, asp:TableCell and asp:TableHeaderCell. Having runat="server" in the asp:Table element is sufficient.
- However, any ASP.NET controls you place within the table cells will need their own runat="server" attributes. This is because these are separate controls that just happen to be placed within the table. For example, asp:Image and asp:HyperLink will need their own runat="server" attributes.



If the ASP.NET controls don't appear on the web page, but they do appear in the source with the "asp" prefix, what attribute did you forget to use?

12. Edit *ExampleMaster* and update the menu div so that the "View Inventory" option points to [~/Inventory/Catalog.aspx](#). Use an ASP.NET style hyperlink.

The ~ character is not valid in HTML, but it is valid in server controls, where it represents the root of the site. During the render process, ASP.NET will translate ~ to the corresponding relative path.

For example, [~/Inventory/images/Blue-Thumb.png](#) will be rendered as **images/Blue-Thumb.png** from within the Inventory folder.

13. Save and run your page. Navigate to *Catalog.aspx*. Why is the banner image missing?

■

---

14. Try to follow the *Home* link on the menu. What happens? What is causing this problem?

■

---

15. Add `runat="server"` to the banner image's `img` element (do not change it to `asp:Image`).
16. Change the `src` path of the `img` element so that it is an absolute path (using ~).
17. Add `runat="server"` to the link around the banner image, and change the path so that it is an absolute path.
18. Fix both anchor tags in the menu.
19. Run your site. Verify that all images appear and links work as expected.

## Part 3: Order Page

---

In this part of the exercise you will be provided with a significant amount of content on the Orders page. The reason you are not expected to produce the content of this page yourself is that the controls on this page are not used in HyperspaceWeb, mainly because they rely on postback communication with the server.

It's still worth including this page because it will help us make the case for why postback is not efficient during a later exercise.

1. Within the *Inventory* folder, add a new *Web Form with Master Page* called *Order.aspx*
2. Use *Place Order* as the level-1 heading
3. Paste the content of **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\Order.txt** below the heading.
4. The markup in this file is intentionally not indented. To reformat your code: **Main Menu > Edit > Format Document (Ctrl+K, Ctrl+D)**
5. Within *Inventory*, add a new *Web Form with Master Page* called *Thank You.aspx*

6. Use *Thank you for your order* as the level-1 heading.
  7. Change the Place Order list item on the master page so that it is a hyperlink pointing to *Order.aspx*.
  8. Run the website and follow the **Place Order** link. Try navigating through the various options using **Next** and **Previous**.
  9. Look at *Order.aspx*. What ASP.NET control is providing the functionality illustrated in the previous step?
- 
- 

10. Answer the wrap-up questions.

## Wrap Up

---

1. When should you add `runat="server"` to an ASP.NET control?
- 
- 

2. When should you add `runat="server"` to HTML elements?
- 
- 
- 
- 
- 
- 

3. How does running on the server change an HTML control on the client?
- 
- 

4. What is a hidden element and when might you use one?
- 
- 

5. Are hidden elements secured? Why (or why not)?
- 
-

# Creating Custom Controls

Custom controls are a useful way to package up content that you plan reuse in a variety of places. For example, many sites include a page to place orders, so a good candidate for a control would be the UI that collects the credit card information. Another example would be a clock control that displays the current time. Both of these examples are built using a set of HTML elements and text that is combined together to create a useful unit with a distinct purpose.

## Examples

### Credit Card Entry

Visualization of the credit card entry control:

The screenshot shows a user interface for entering credit card information. It includes a dropdown menu labeled "Type of card" with options "Visa", "Master Card", and "American Express". The "American Express" option is highlighted with a blue background. Below the dropdown are two input fields: one for "Card #" and one for "Expires on".

HTML for the control:

```
<label for="lstType" id="lblType">Type of card: </label><br />
<select size="4" id="lstType">
    <option value="visa">Visa</option>
    <option value="ms">Master Card</option>
    <option value="amx">American Express</option>
</select><br />
<label for="txtNumber" id="lblNumber">Card #: </label><br />
<input type="text" id="txtNumber" /><br />
<label for="txtDate" id="lblDate">Expires on: </label><br />
<input type="text" id="txtDate" /><br />
```

The ASP.NET server control you could create to render to the HTML:

```
<etewp:BillingInfo ID="biOrder" runat="server" DefaultCard="amx" />
```

### Clock

Visualization of the clock:

12:00:00 AM

HTML for the control:

```
<span id="spanHours">12</span>:  
<span id="spanMinutes">00</span>:  
<span id="spanSeconds">00</span>  
<span id="spanAmPm">AM</span>
```

The ASP.NET server control you could create to render to the HTML:

```
<etcw:Clock runat="server" ID="clkMain" IsTwentyFourHour="false" />
```

## Kinds of ASP.NET Custom Controls

After you have decided that a custom control is worth making, the next choice is what kind of control to create. There are two different flavors of custom controls in ASP.NET web forms:

### User Controls

User controls are like reusable web pages. They have a markup file, but have the extension .aspx for control (instead of aspx for page). UserControls are better for more complex controls because the markup page makes it easy to assemble lots of different pieces together.

That said, user controls are trickier to package and distribute across projects.

### Web Controls

Web controls are written entirely in C# and do not have a markup page. This make it more tedious to add the content to the control, but it ends up being easier to package and distribute. Simpler controls with less content tend to be created as web controls.

### Why choose one kind of control over the other?

It mainly has to do with preference, as either control will work for most scenarios. HyperspaceWeb did it this way:

- Activities have *UserControl* as an ancestor in their class hierarchy
- Most other controls have *WebControl* as an ancestor in their class hierarchy. For example:
  - Entry fields

- Category select fields
- Record select fields
- etc...

## Exercise 6: Creating Custom Controls

---

In this exercise you will create both a custom User Control and Web Control.

### Part 1: Create a Billing-info User Control

---

In this part you will create a billing info user control. The markup is relatively straight-forward, so that will be provided to you.

1. Create a new user control:

- **Solution Explorer > Training.Example.Web.Pages > Inventory > Right Click > Add > New Item > Web Forms User Control**
- Name: *BillingInfo.ascx*

2. The markup for this control is provided for you in the following file:

- **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\billing.txt**
- Append this markup at the bottom of *BillingInfo.ascx*:

### Part 2: Use the Billing-info User Control

---

In order to add both user controls and web controls to ASP.NET markup, you must first register the control at the top of the page.

Register Directive	Indicates that the user or web control is available for use on this web page. For example:  <code>&lt;%@ Register Src="~/Inventory/BillingInfo.ascx" TagName="BillingInfo" TagPrefix="etewp" %&gt;</code>
Src	An attribute pointing to a user control
TagName	The portion of the tag to the right of ":". Usually the .NET class name is used. For example <i>BillingInfo</i> .
TagPrefix	The portion of the tag to the left of ":". Usually the prefix is based on an acronym of the namespace of the control. For example, <i>etewp</i>

	(Epic.Training.Example.Web.Pages.Inventory).
--	----------------------------------------------



### The namespace to tag prefix relationship is based on namespaces in XML.

If elements from two different namespaces are used in the same XML file, then a namespace prefix must be prepended to all elements so that it is clear where the elements are defined.

The exception to this rule is the default namespace. If no prefix is provided, the default namespace is assumed.

1. Open *Inventory/Order.aspx*
2. To register the billing info control, place the following *Register* directive directly below the *Page* directive at the top of *Order.aspx*:

```
<%@ Register Src="~/Inventory/BillingInfo.ascx"  
    TagName="BillingInfo"  
    TagPrefix="etewp" %>
```

3. Save and build your solution.
4. Scroll to the bottom of *Order.aspx*.
5. Within the final wizard step, add an instance of the *BillingInfo* control:

```
<asp:WizardStep ID="stpBillingInfo" runat="server"  
    Title="Billing Information">  
    <h2>Billing Information</h2>  
    <etewp:BillingInfo ID="biOrder" runat="server" />  
</asp:WizardStep>
```



### You should receive IntelliSense help when you start typing the tag prefix.

However, sometimes IntelliSense takes extra time to catch up with newly registered controls, so don't be alarmed if it doesn't work right away.

6. Run your web application and navigate to the billing-info step of the order page. Do you see the control?

## Part 3: Create a Custom Attribute

You want to add an attribute to the *BillingInfo* user control that will allow you to choose the default credit card within the markup. Attributes in markup correspond to properties within the corresponding class. This means you will need to modify the *BillingInfo* class.

1. Open *BillingInfo.ascx.cs*
2. Within the *BillingInfo* class, add an enumerated type to represent the different kind of credit cards:

```
public partial class BillingInfo : System.Web.UI.UserControl
{
    public enum CardType
    {
        visa,
        ms,
        amx
    }
    ...
}
```

The name of each enumeration must match the *Value* of the list items within the markup (from the *asp:ListBox* in *BillingInfo.ascx*):

```
<asp:ListItem Text="Visa" Value="visa"/>
<asp:ListItem Text="Master Card" Value="ms"/>
<asp:ListItem Text="American Express" Value="amx"/>
```

3. Add public property named *DefaultCard* of type *CardType* (matching the enumeration just created):

```
public CardType DefaultCard { get; set; }
```

This property will become available as an attribute of *etewp:BillingInfo*.

4. Next you need to connect the property value assigned in markup with the actual item that is selected in the list. You can do this when the control is initialized by creating the *Page\_Init* method within *Inventory/BillingInfo.ascx.cs*.

```
protected void Page_Init(object sender, EventArgs e)
{
    foreach (ListItem li in lstType.Items)
    {
        if (li.Value == DefaultCard.ToString())
        {
            li.Selected = true;
```

```
        break;  
    }  
}  
}
```

5. Save and build your solution
6. Switch back to *Order.aspx*
7. Indicate that the *DefaultCard* is *amx*
  - You should get IntelliSense help on both the attribute name as well as possible values.
8. Run your solution and navigate to the billing-info step.
9. Verify that *American Express* was selected by default.

## Part 4: Clock WebControl Overview

In this part you will create a clock web control. The clock will render to either: 12:00:00 AM, or 24:00:00, depending on if it is set to 12 or 24 hours. Instead of creating a user control, you will create a web control entirely in C#. Creating the content will be a bit more work, but sharing it across assemblies will be far easier.



The clock will not keep time yet. We will add this feature using JavaScript in a later lesson.

The code for the control you will create has the following properties:

Product	Product independent (goes in <i>Code</i> folder)
Owner	Training
Application	Core (Foundations-level control)
Functional Area	Controls.Time
Platform	Web

1. Based on the above properties, where (in terms of a folder path) should you create the project and associated C# files for this control (the correct answer is provided later)?

- ---
- 2. What should the namespace be (the correct answer is provided later)?

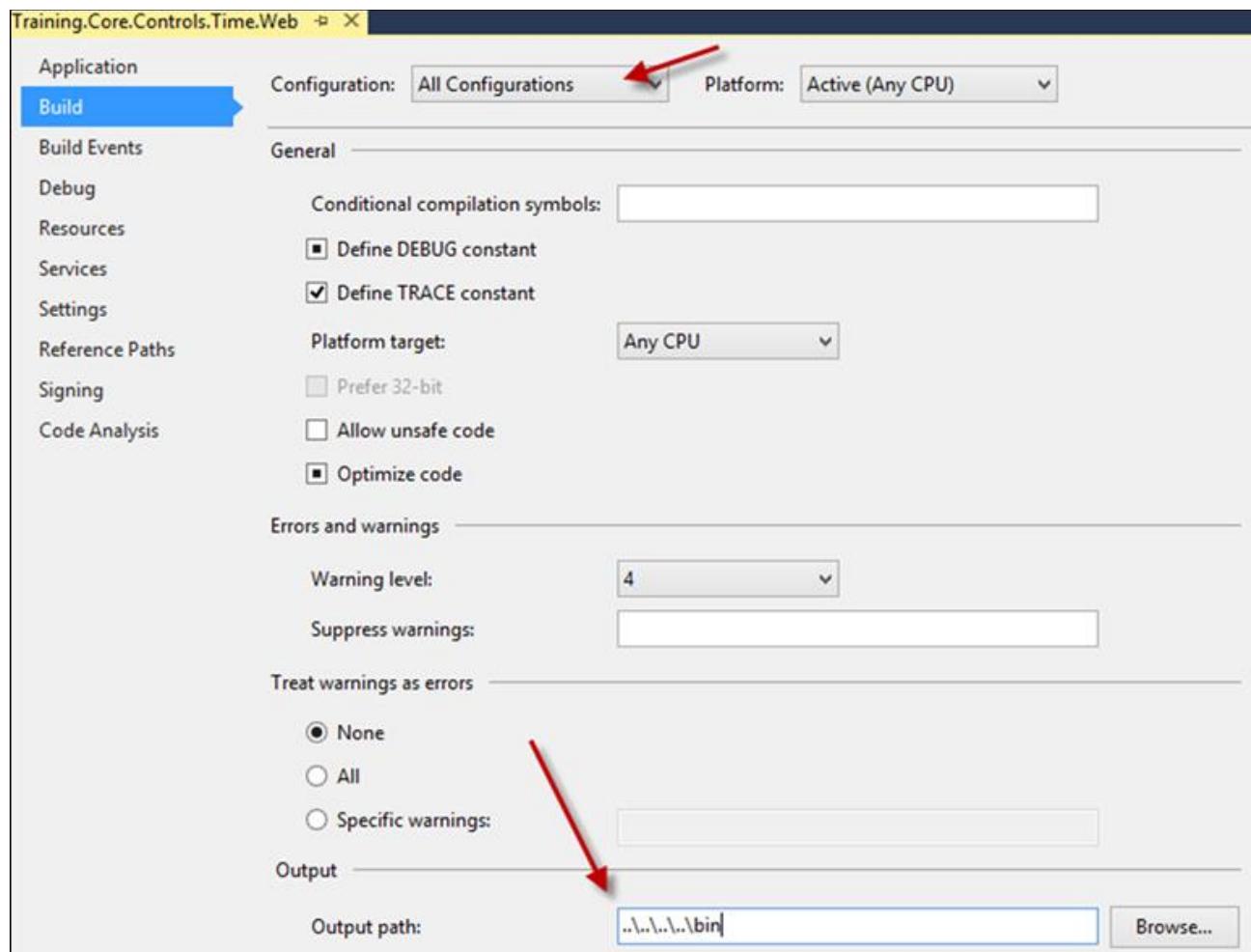
---

## Part 5: Add a new Project for the clock control

---

1. Open your *BigMouse* solution.
2. Add a new project:
  - **File > Add > New Project > Visual C# > Class Library**
  - Name: **Controls.Time**
  - Location: **C:\EpicSource\WTC\Code\Training\Core\Web**
3. The clock control will be created as part of a class library within the Code branch. What does this tell you about how web controls will typically be used at Epic?

---
4. Delete *Class1.cs*
5. Rename the project
  - From: *Controls.Time*
  - To: *Training.Core.Controls.Time.Web*
6. Edit the project properties
7. Change the assembly name and default namespace to *Epic.Training.Core.Controls.Time.Web*.
8. On the Build tab, change the output path for all configurations to: ...\\...\\...\\...\\bin
  - In other words, the output path should be in the bin folder that is four steps back from the project location.
  - This will place it within the `Code/bin` folder



*Changing the build path for all configurations*

9. Save and close the project properties
10. Under References in the solution explorer, add the following reference:
  - Assemblies > Framework > System.Web



Don't forget to add the reference, otherwise you will not be able to complete the following steps.

## Part 6: Create the Control

1. Add a new class called *Clock.cs*
2. Make sure that the class is *public*
3. Indicate that the class inherits from *WebControl*

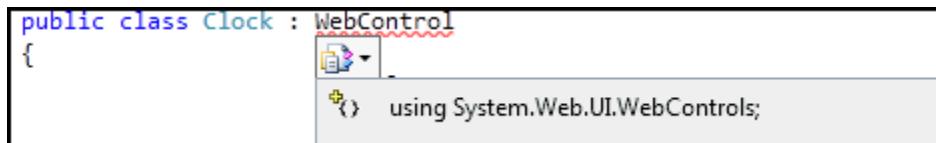
Notice that *WebControl* is not recognized. This is because it lives in the *System.Web* assembly, but you have not yet included the proper using statement. Fortunately, Visual Studio is smart enough to resolve this problem automatically.

4. Place the cursor over *WebControl*. You will notice that a blue underline appears. This means there is a problem, but it can be automatically resolved:

```
namespace Epic.Training.Core.Controls.Time.Web
{
    public class Clock : WebControl
```

*Blue underline indicates the possibility of auto-resolving an error*

1. Press *Ctrl + .* (press and hold control, followed by the period key). This will expand the menu of options that you have to resolve the problem



*Auto-resolve options for WebControl*



### Try auto-resolving problems

The instructions in this workbook will not always remind you to add the required using statements. If you try using a class that doesn't exist, try auto-resolving it before asking the trainer for help.

2. Select the first option, *using System.Web.UI.WebControls;*
3. Create four fields to store the hours, minutes, seconds and AM/PM marker. The data type should be *HtmlGenericControl*:

```
private HtmlGenericControl _hours;
private HtmlGenericControl _minutes;
private HtmlGenericControl _seconds;
private HtmlGenericControl _amPm;
```

*HtmlGenericControls* can render to any HTML tag. By default, they render to span, which is a generic in-line element. You can set a more specific element when instantiating the *HtmlGenericControl*.

4. The clock should render to a div that contains a span corresponding to each of the controls created above. To do so, define a default constructor (constructor without parameters) that calls the base *WebControl* constructor with the argument "div":

```
public Clock(): base("div")
{
}
```

5. Within the constructor, initialize each of the private members to a span. We will also set the ID attribute and some default *InnerText* at this time:

```
_hours = new HtmlGenericControl("span") {
    ID = "spanHours",
    InnerText = "12"
};
_minutes = new HtmlGenericControl("span") {
    ID = "spanMinutes",
    InnerText = "00"
};
_seconds = new HtmlGenericControl("span") {
    ID = "spanSeconds",
    InnerText = "00"
};
_amPm = new HtmlGenericControl("span") {
    ID = "spanAmPm",
    InnerText = "AM"
};
```

6. Although you have created the span controls, they are not actually within the *WebControl* yet. Next, add them to `this.Controls`, with the delimiters usually used for a clock:

```
Controls.Add(_hours);
Controls.Add(new LiteralControl(":"));
Controls.Add(_minutes);
Controls.Add(new LiteralControl(":"));
Controls.Add(_seconds);
Controls.Add(new LiteralControl(" "));
Controls.Add(_amPm);
```

7. To clean up the file a bit, remove and sort your *using* statements:

- **Main Menu > Edit > IntelliSense > Organize Usings > Remove and Sort**

8. Build your solution so an assembly is generated



If you do not build the assembly, you will not be able to reference it later.

9. The final code should be similar to the following:

```
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;

namespace Epic.Training.Core.Controls.Time.Web
{
    public class Clock : WebControl
    {
        private HtmlGenericControl _hours;
        private HtmlGenericControl _minutes;
        private HtmlGenericControl _seconds;
        private HtmlGenericControl _amPm;

        public Clock(): base("div")
        {
            _hours = new HtmlGenericControl("span") {
                ID = "spanHours",
                InnerText = "12"
            };
            _minutes = new HtmlGenericControl("span") {
                ID = "spanMinutes",
                InnerText = "00"
            };
            _seconds = new HtmlGenericControl("span") {
                ID = "spanSeconds",
                InnerText="00"
            };
            _amPm = new HtmlGenericControl("span") {
                ID="spanAmPm",
                InnerText="AM"
            };
            Controls.Add(_hours);
            Controls.Add(new LiteralControl(":"));
            Controls.Add(_minutes);
            Controls.Add(new LiteralControl(":"));
            Controls.Add(_seconds);
            Controls.Add(new LiteralControl(" "));
            Controls.Add(_amPm);
        }
    }
}
```

## Part 7: Use the Control

---

Because the Clock control exists in a separate project from the web application, you must include the clock assembly as a reference.

1. Add the clock control assembly reference to the web application:

- **Solution Explorer > Training.Example.Web.Pages > References > Right Click > Add Reference > Browse**
  - In the above path, use the Browse button, not the expandable list
- Navigate to: **C:\EpicSource\WTC\Code\bin\**
- Select **Epic.Training.Core.Controls.Time.Web.dll**



#### Do not use a project reference

Instead, navigate directly to the DLL as directed above. Project references will not work in HyperspaceWeb because the paths are not maintained correctly when the project is published.

2. Set a project dependency so that the clock project compiles before the web application:

- **Training.Example.Web.Pages > Right Click > Build Dependencies > Project Dependencies**
- Check **Epic.Training.Core.Controls.Time.Web**
- Click **OK**

3. Open the master page and register the clock assembly:

```
<%@ Register  
    Assembly="Epic.Training.Core.Controls.Time.Web"  
    Namespace="Epic.Training.Core.Controls.Time.Web"  
    TagPrefix="etcw"  
%>
```

Note that you are registering **an entire namespace**. This means that if you created additional controls, they would automatically be available on the master page with the prefix *etcw* (after the assembly is recompiled).

4. Build your solution.
5. Add a clock control just below the banner image:

```
<div>  
    <!--header-->  
    <a runat="server" href("~/Default.aspx" title="Return home">
```

```
<img runat="server" src("~/images/banner.png"
    alt="Big Mouse Cheese Factory"
    width="694" height="125" />
</a>
<etcw:Clock runat="server" ID="clkMain" />
</div>
```

6. Save the master page.
7. Set the Pages project as the start project:
  - **Training.Example.Web.Pages > Right Click > Set as StartUp Project**



#### You cannot execute a class library

When you start debugging, Visual Studio will attempt to run the currently selected project. Because the clock project is a class library, not an application, attempting to run it would cause an error.

8. Run your web application
9. Verify that the clock appears directly below the banner image
10. View the source of your page to see how the clock was rendered. Notice that there is a problem with the span IDs. What is it?
  - a. If you can't figure out the problem, try adding a second clock with ID="clkSecond".
  - b. Look carefully at the id of corresponding spans between the two clocks.

The issue is that the component pieces of the clock do not have unique ids. This happened because Clock is not a naming container. Naming containers are used by ASP.NET to guarantee that component parts of a control are assigned unique IDs.



For more information on Naming containers, see:

- [https://msdn.microsoft.com/en-us/library/system.web.ui.inamingcontainer\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.web.ui.inamingcontainer(v=vs.110).aspx)

11. Make *Clock* a naming container, by implementing the *INamingContainer* interface:

```
public class Clock : WebControl, INamingContainer
```

12. Run your site again, and verify that the spans are assigned an id that would be unique, even if multiple clocks exist on the same page.

## Part 8: Add an Attribute

1. Add a public bool property called *IsTwentyFourHour*:

```
public bool IsTwentyFourHour { get; set; }
```

2. Override the protected method *OnInit*. If *IsTwentyFourHour* is true, set the *InnerText* of *\_hours* to "24", and the *InnerText* of *\_amPm* to a null string.

```
protected override void OnInit(EventArgs e)
{
    if (IsTwentyFourHour)
    {
        _hours.InnerText = "24";
        _amPm.InnerText = String.Empty;
    }
}
```

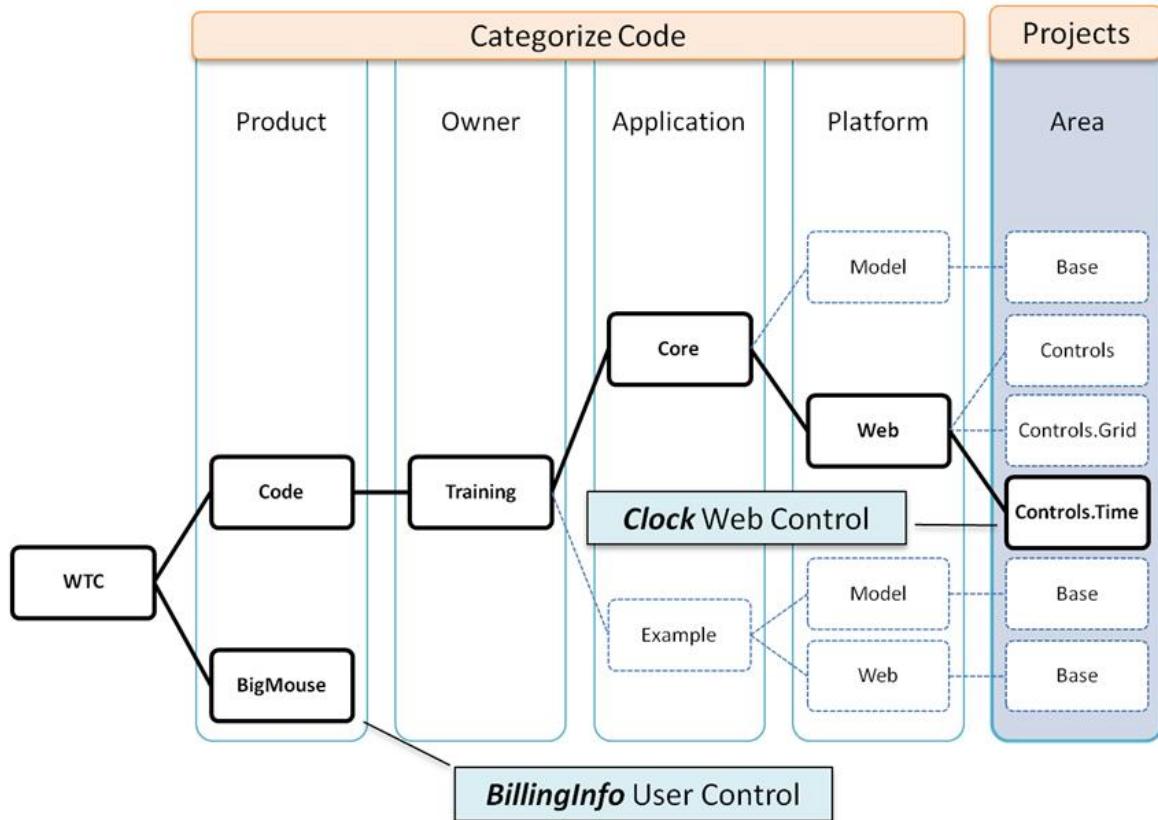
3. Rebuild your solution.
4. Change the clock on your master page so that it displays in 24h time.

```
<etcw:Clock runat="server" ID="clkMain" IsTwentyFourHour="true" />
```

5. Run your site and verify that it worked.
6. Answer the wrap-up questions.

## Wrap Up

1. Verify that the source code for your new custom controls was added to the following folder locations:



2. Place a check under the type of control that matches the given description:

Description	UserControl	WebControl
Has a markup file		
Usually used for product-specific controls, such as Activities		
Usually used for shared controls (record select field, category select field, etc.)		
Easier to distribute		
Easier to create		

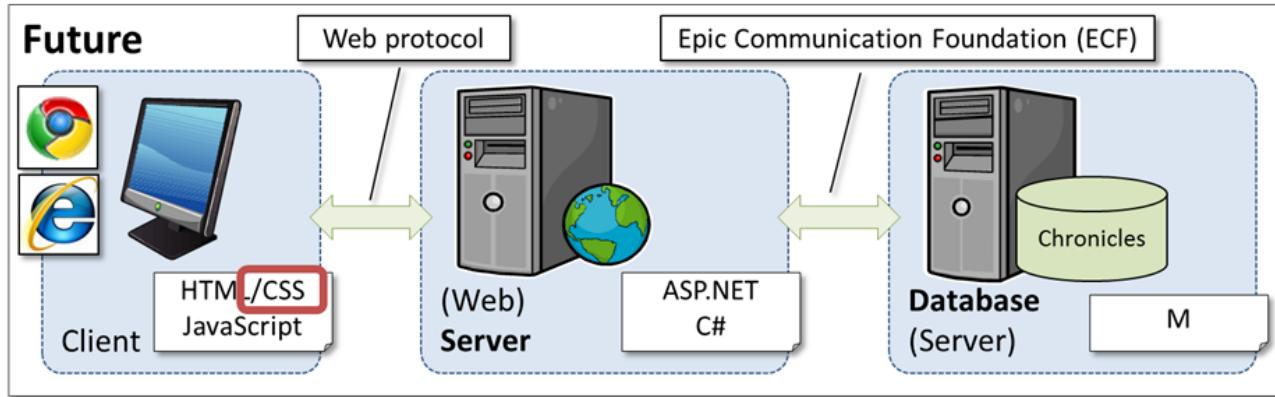
# Lesson 3: Setting Layout

<b>The Big Picture.....</b>	<b>3•3</b>
By the End of This Lesson, You Will Be Able To.....	3•3
<b>Applying Basic Styles .....</b>	<b>3•4</b>
Follow Along: Creating a New Style Sheet.....	3•4
Follow Along: Applying Styles .....	3•5
Follow Along: Changing Font Size .....	3•5
Follow Along: Exploring Developer Tools.....	3•7
Follow Along: Applying Background Color.....	3•8
Follow Along: Layering Background Images.....	3•8
Current Layout.....	3•11
Arranging Block Elements .....	3•12
Follow Along: Achieving Consistent Defaults.....	3•13
Desired Layout.....	3•14
Exercise: Applying Block-Element Styles.....	3•14
Part 1: Determine Styles to Adjust.....	3•14
Part 2: Arrange Block Elements .....	3•15
Part 3: Headers and Links.....	3•17
Part 4: Final Adjustments.....	3•19
If you have time .....	3•20
Wrap Up .....	3•20
<b>Formatting Tables .....</b>	<b>3•23</b>
Arranging Table Elements.....	3•23
Exercise: Adjust the layout of Tables.....	3•24
Part 1: Table Formatting.....	3•24
Part 2: Finishing Touch.....	3•25
Wrap Up .....	3•25
If You Have Time .....	3•26
<b>Laying out Form Elements .....</b>	<b>3•27</b>
Use Same UI Guidelines as VB.....	3•27
Using Layout Tables.....	3•27
Exercise: Adjust the Layout of Forms.....	3•28
Part 1: The Email Form.....	3•28
Part 2: Create the login page .....	3•29

If you have time.....	3•31
Wrap Up.....	3•31
<b>Resolving Conflicting Styles.....</b>	<b>3•33</b>
Style Conflicts.....	3•33
Cascade Algorithm .....	3•34
Specificity Calculation.....	3•34
Examples.....	3•35
Exercise: Fix the Details Page .....	3•36
Scenario.....	3•36
Part 1: Get the details page .....	3•37
Part 2: Fix the Page.....	3•37
Wrap Up.....	3•38
Exercise: Predict the Style.....	3•38
<b>Arranging Elements.....</b>	<b>3•41</b>
Follow Along: Floating Block Elements.....	3•41
Alternate Page Layouts.....	3•42
Browser Differences .....	3•43
Consistent Browser Arrangement.....	3•43
Centering Pairs of Block Elements.....	3•44
The Position Style.....	3•45
Static.....	3•46
Relative.....	3•47
Absolute.....	3•48
Fixed.....	3•49

# Setting Layout

## The Big Picture



A big advantage of HTML/ASP.NET is that it specifies what the content of a page is without enforcing any particular layout or theme. Cascading Style Sheets (CSS) can then be used to give a page its look and feel. These sheets can then be swapped to apply different themes/skins to the page.

### By the End of This Lesson, You Will Be Able To...

- Describe how style and layout is controlled in web applications
- Troubleshoot layout and styling problems

# Applying Basic Styles

## Follow Along: Creating a New Style Sheet

1. Use the following path to create a new style sheet:
  - **Solution Explorer > Training.Example.Web.Pages > Right-Click > Add > New Item > Style Sheet > ExampleMaster.css**
2. Add the following code to the style Sheet:

```
body
{
    font-family: Verdana, Arial, sans-serif;
}
```

Selector	The portion of a style that determines which elements will be assigned the styles in the following code block. The code block is delimited with braces.  Example:  selector { style: values; }
Style	Styles indicate the properties of the element that will be assigned the given values. The values themselves have different syntax depending on the style. The values are usually delimited by spaces or commas.
Inherited Styles	Certain styles are <i>inherited</i> , meaning that if an element's style is not explicitly assigned a value (for example, by a selector), then it will be assigned the same value as its parent in the HTML tree.
font-family	A style that indicates which fonts should be used by the selected elements and their child elements. Typically, multiple font families are listed because any given client may not have a particular font included. Fonts are comma delimited. The client will scan the fonts from left to right and choose the first one that it has installed.  For a consistent look and feel, the <i>font-family</i> style is inherited by most elements.

	<b>Note:</b> Form elements do not inherit the <i>font-family</i> style.
--	-------------------------------------------------------------------------

## Follow Along: Applying Styles

There are several ways that styles can be applied to elements.

- They can be directly applied to elements in the markup using the *style* attribute
  - They can be embedded in the HTML using the *style* element.
  - (preferred) **External style sheets** can be linked in the *head* element of the page using the *link* element
1. Link to the new CSS file in the master page

```
<head runat="server">
    <title>Big Mouse Cheese Factory</title>
    <link rel="stylesheet" href="ExampleMaster.css"
        type="text/css" media="all" />
</head>
```

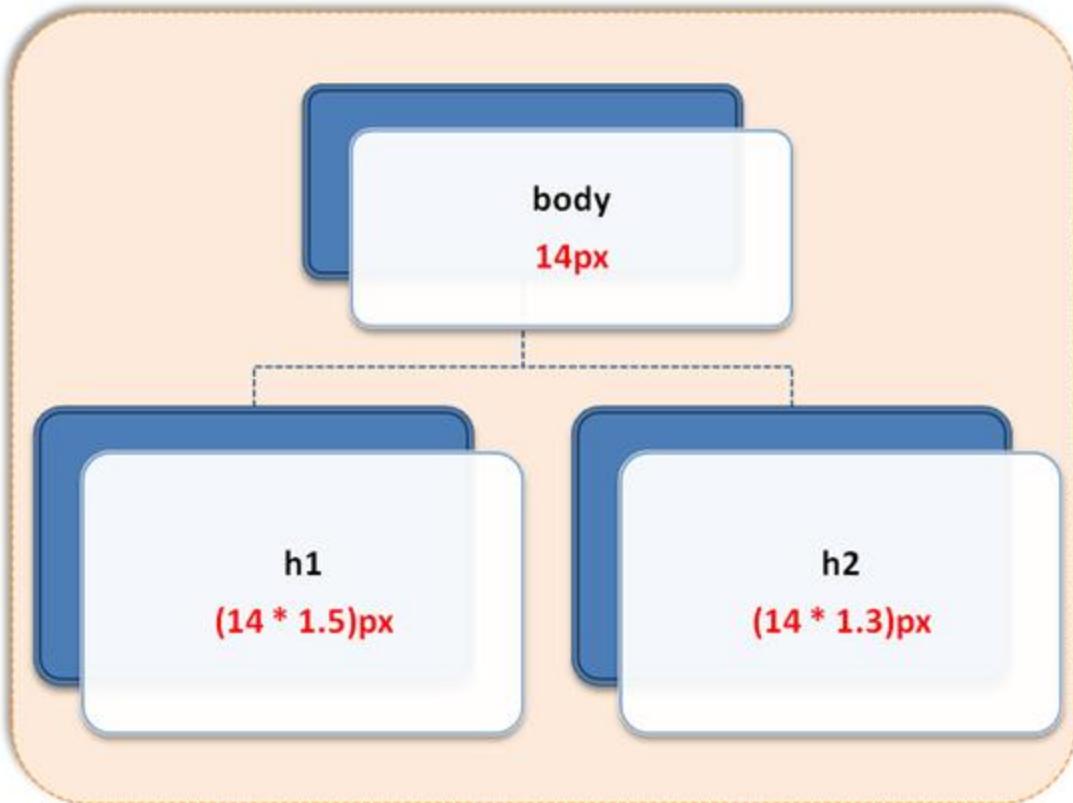


The media attribute of the link element is used to describe when this style sheet should be included. The various options are:

<u>Value</u>	<u>Description</u>
screen	Computer screens (this is default)
tty	Teletypes and similar media using a fixed-pitch character grid
tv	Television type devices (low resolution, limited scroll ability)
projection	Projectors
handheld	Handheld devices
print	Print preview mode/printed pages
braille	Braille feedback devices
aural	Speech synthesizers
all	Suitable for all devices

## Follow Along: Changing Font Size

Like the *font-family* style, the *font-size* style is also inherited. The font size can either be specified as an absolute pixel size, or as a percentage of the inherited/default size. Examine the following diagram to determine how the font size values will be inherited and applied.



Use the following steps to adjust the font sizes of your page:

1. Set the default font size for the entire page by adding it to the `body`-element selector:

```
body
{
    font-family: Verdana, Arial, Sans-Serif;
    font-size: 14px;
}
```

2. Adjust all level 1 headings so that they are 50% larger than standard text. Do so by adding a `h1`-element selector:

```
h1
{
    font-size: 150%;
}
```

3. Specify that level-2 headings are 30% larger than standard text:

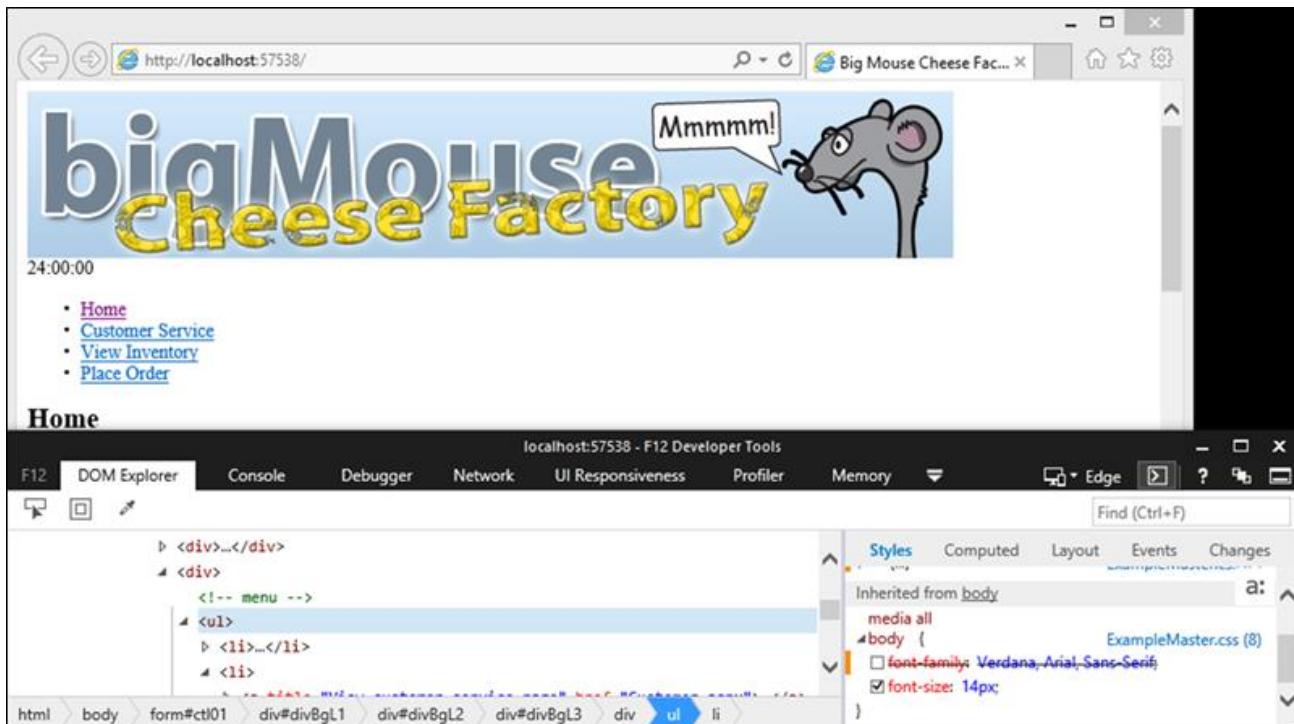
```
h2
{
    font-size: 130%;
}
```

## Follow Along: Exploring Developer Tools

Debugging style sheets can be a frustrating process without the proper tools. Each browser has a similar method for viewing, disabling and changing styles applied to the page elements.

1. Run your web application in IE
2. Open the developer tools (see the following screenshot)
3. Try toggling the various styles to see what happens.
4. Watch the video about the IE Developer Toolbar linked here:  
<http://brainbow.epic.com/Learning/Profile.aspx?csn=35545>

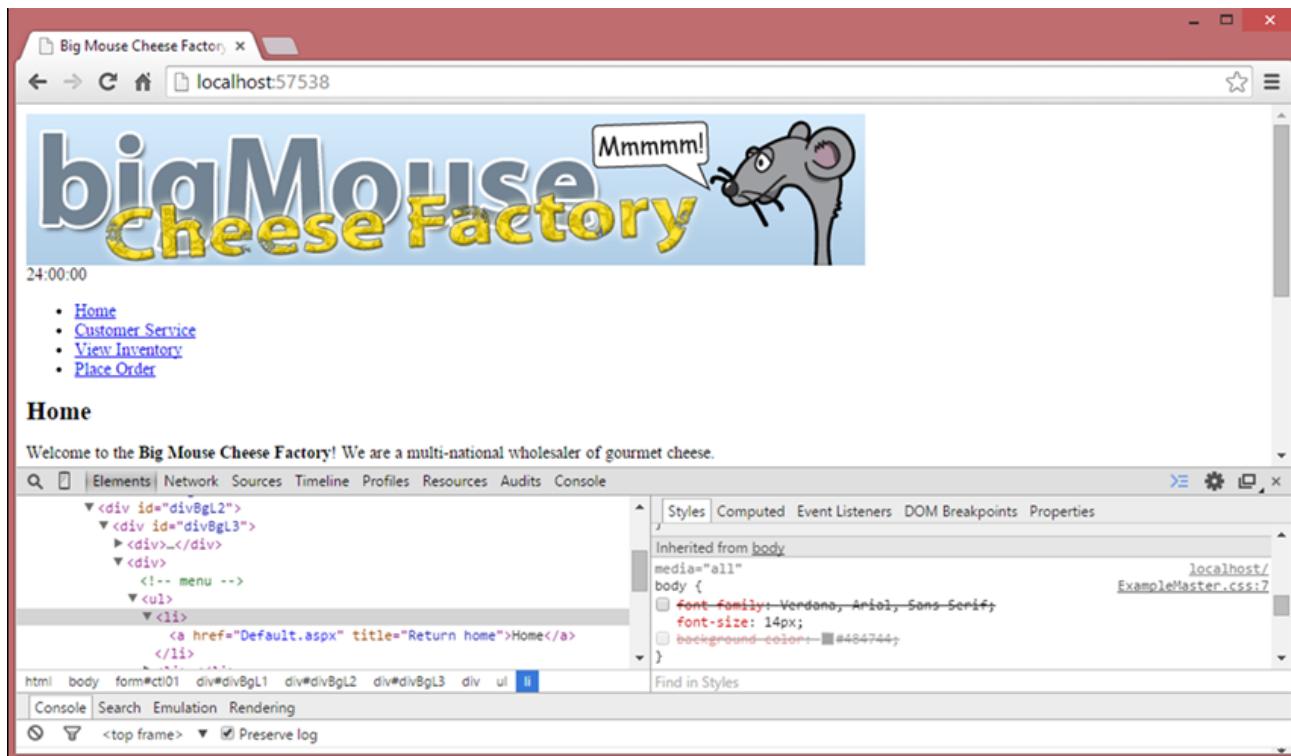
**Path:** From any page -> press F12 -> DOM Explorer



### Developer Tools for Internet Explorer

5. Run your web application in Chrome and then experiment with the CSS options.

**Path:** From any page -> press F12



Developer tools in Chrome (JavaScript Console)

## Follow Along: Applying Background Color

The background color of all elements is transparent by default. The one exception is the body element, which is white. The default can be overridden using the *background-color* style.

background-color	<p>The background color can be set to either a predefined color name or a hexadecimal value, #RRGGBB, where R is a red digit, G is a green digit, and B is a blue digit. For example:</p> <ul style="list-style-type: none"> <li><b>Predefined:</b> Red, Black, or White</li> <li><b>Hexadecimal:</b> #FF98CE</li> </ul>
------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. Apply a dark grey background to the body of the master page:

```

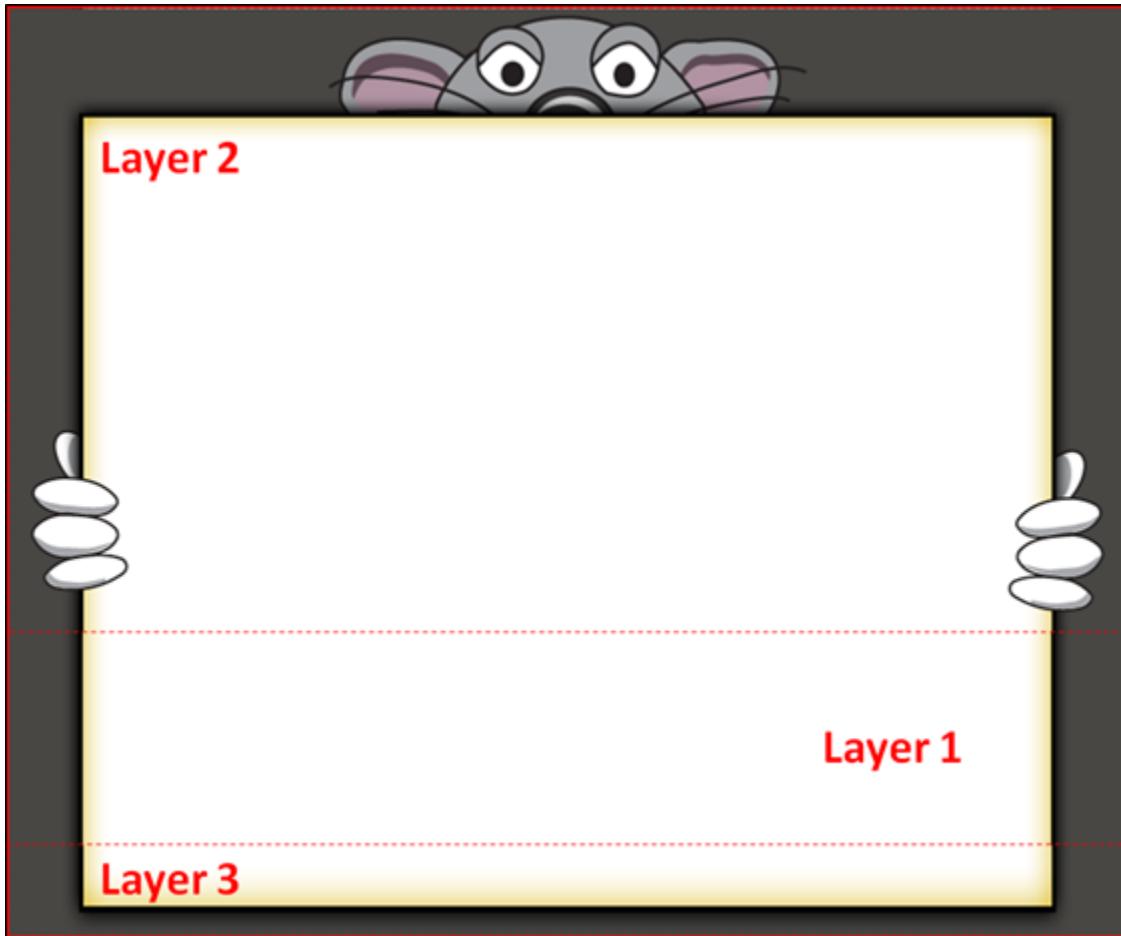
body
{
    font-family: Verdana, Arial, sans-serif;
    font-size: 14px;
    background-color: #484744
}

```

## Follow Along: Layering Background Images

It is common for websites to use images in the background to achieve a slick-looking theme. Often, the background image seems to shrink and grow with the content.

This effect is achieved by layering several different background images together using nested div elements.



*An example of layered images*

1. Add three additional div elements to the master page, just inside the form element.
  - Be sure not to delete the elements that are already on the page.

```
<body>
    <form runat="server">
        <div id="divBgL1">
            <div id="divBgL2">
                <div id="divBgL3">

                    ... Other elements ...

                </div>
            </div>
        </div>
```

```
</form>
</body>
```

2. Add a selector that selects just the *divBgL1* element. This element will contain the image that grows and shrinks with the content of the page:

```
#divBgL1
{
    background-image:url(images/BGLayer1.png);
    background-repeat:repeat-y;
    background-position:center;
    min-width: 800px;
}
```



Web essentials will complain when referencing images that are smaller than a certain size with this error:

*"Performance: The image is only X bytes and should be embedded as a base 64 dataURI to reduce the number of HTTP requests."*

**You should ignore this error.**

In general this is not bad advice, but in the HyperspaceWeb framework images and icons are combined into one large image (i.e., sprite map). That way the requests are reduced without embedding the image data into the CSS file.

ID selector	A selector with syntax <code>#id</code> that selects just the element with the given id.
background-image	The style used to apply a background image to an element. The value has the syntax: <code>url(relative path to image)</code>
background-repeat	<p>The style that specifies how an image is repeated if the size of the element that it is applied to is larger than the image itself. Possible values are:</p> <ul style="list-style-type: none"> <li>• <b>inherit</b>: Use the containing element's style</li> <li>• <b>no-repeat</b>: Do not repeat the image</li> <li>• <b>repeat</b>: Repeat the image in both the x and y directions</li> <li>• <b>repeat-x</b>: Repeat only horizontally</li> <li>• <b>repeat-y</b>: Repeat only vertically</li> </ul>

background-position	Specify the horizontal and vertical positioning of the background image. The syntax is <code>background-position: [Horizontal] [Vertical];</code> Possible values are: <ul style="list-style-type: none"><li>• <b>Horizontal:</b> left, center, right</li><li>• <b>Vertical:</b> bottom, center, top</li></ul>
min-width	Use to specify the minimum width of an element.

3. Add a selector to apply styles to the `divBgL2` element:

```
#divBgL2
{
    background-image: url (images/BGLayer2.png);
    background-repeat: no-repeat;
    background-position: center top;
}
```

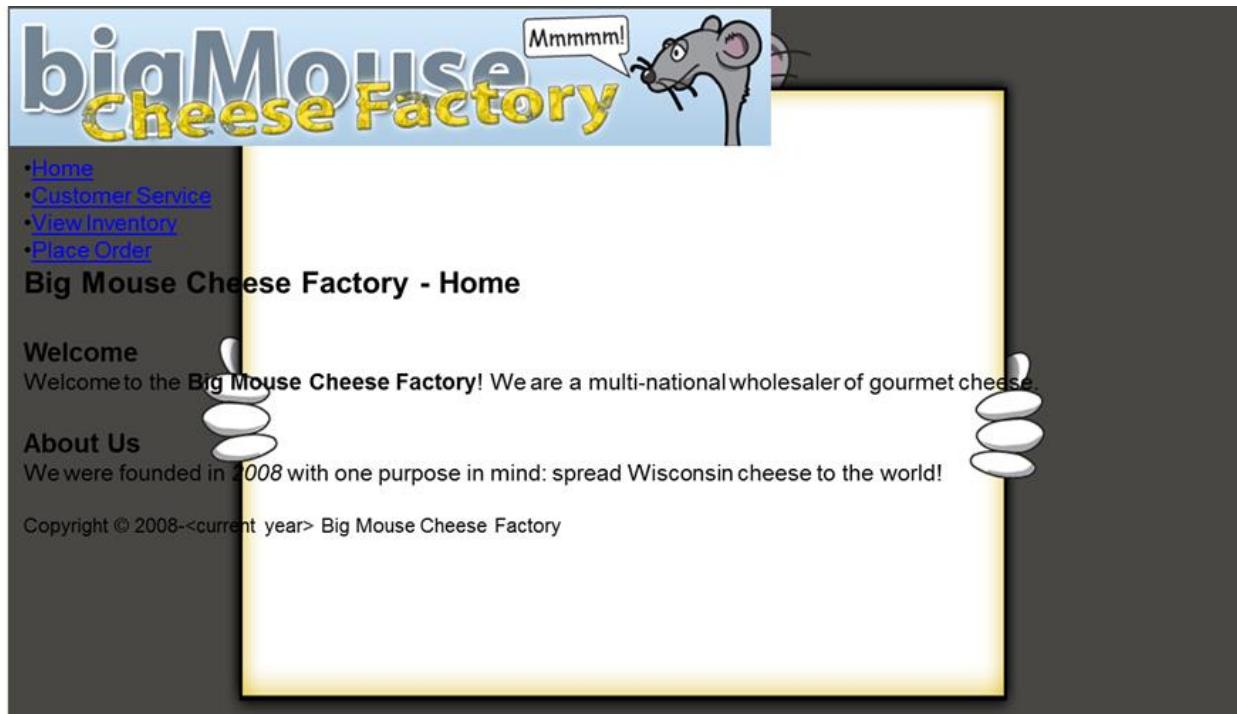
4. Add the final image to the third background div with id `divBgL3`:

```
#divBgL3
{
    background-image: url (images/BGLayer3.png);
    background-repeat: no-repeat;
    background-position: center bottom;
}
```

## Current Layout

---

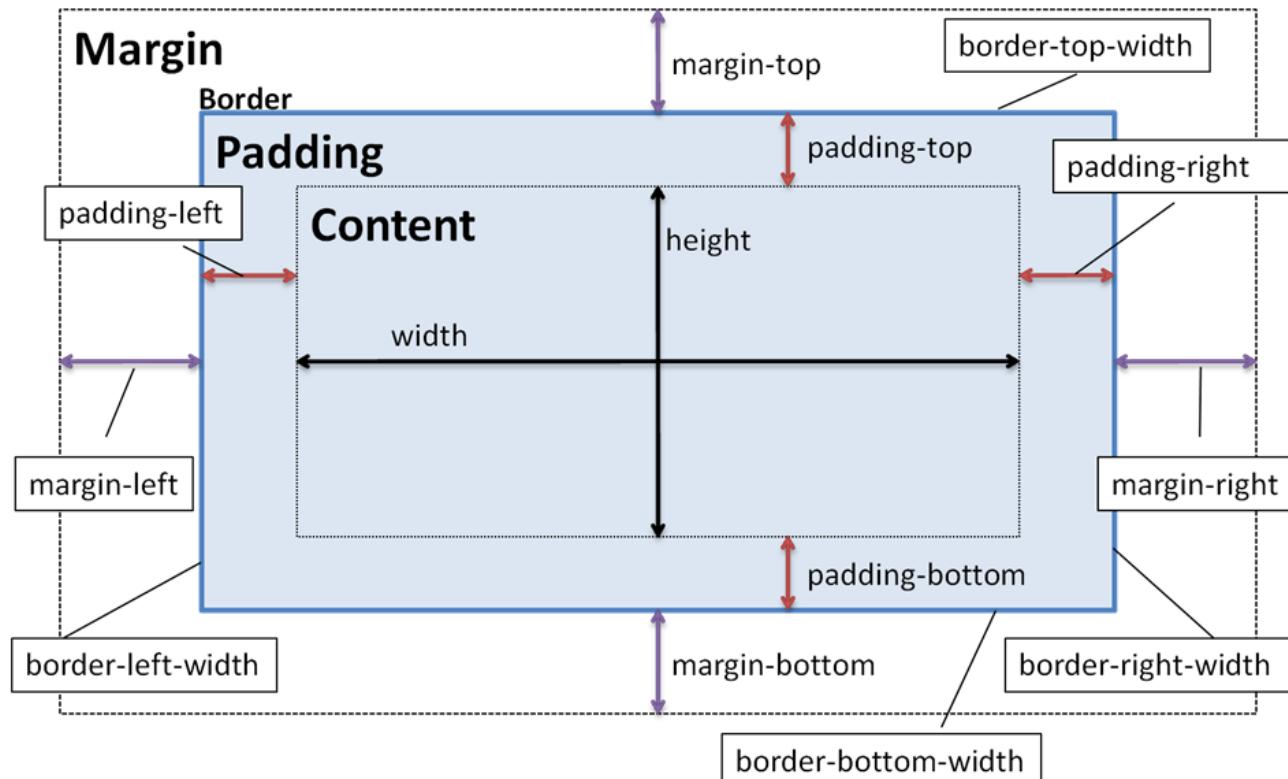
Although the background image is centered and grows with the content, the content itself is not centered on the background image. In order to properly center the content, you will need to understand the CSS block model.



*Page content is not centered on the background image*

## Arranging Block Elements

In general, you can specify the size of the top, right, bottom and left of several different layout styles:



### Block model styles

Margin	The <b>transparent</b> space around an element.
Border	The line surrounding a block element that separates the margin from the padding
Padding	The space that separates the border from the content. The padding will include the background color or image that you apply to the block element.
Content	The space where contained elements and text can appear. The height and width styles apply to the content portion of a block element.



For the CSS quiz, be able to calculate the total width of a block element.

## Follow Along: Achieving Consistent Defaults

One challenge to web development is that every browser has its own set of default values for CSS styles. One way to achieve consistent defaults is to use the *star* selector to apply the same value to all elements. The defaults can be overridden later with more specific selectors.

1. Set default margins and padding to zero for all elements:

```
*  
{  
    margin: 0;  
    padding: 0;  
}
```



Normally the \* selector is strongly discouraged and will result in CSS warnings in Visual Studio. Normally you should obey all such CSS warnings, but one exception is the purpose for which we are currently using the \* selector, to set consistent cross-browser margins and padding.

For details on CSS warnings that you might encounter, see:

[http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting\\_Started/Web\\_Essentials#Css\\_warnings](http://wiki.epic.com/main/Foundations/HyperspaceWeb/Public/Getting_Started/Web_Essentials#Css_warnings)

## Desired Layout

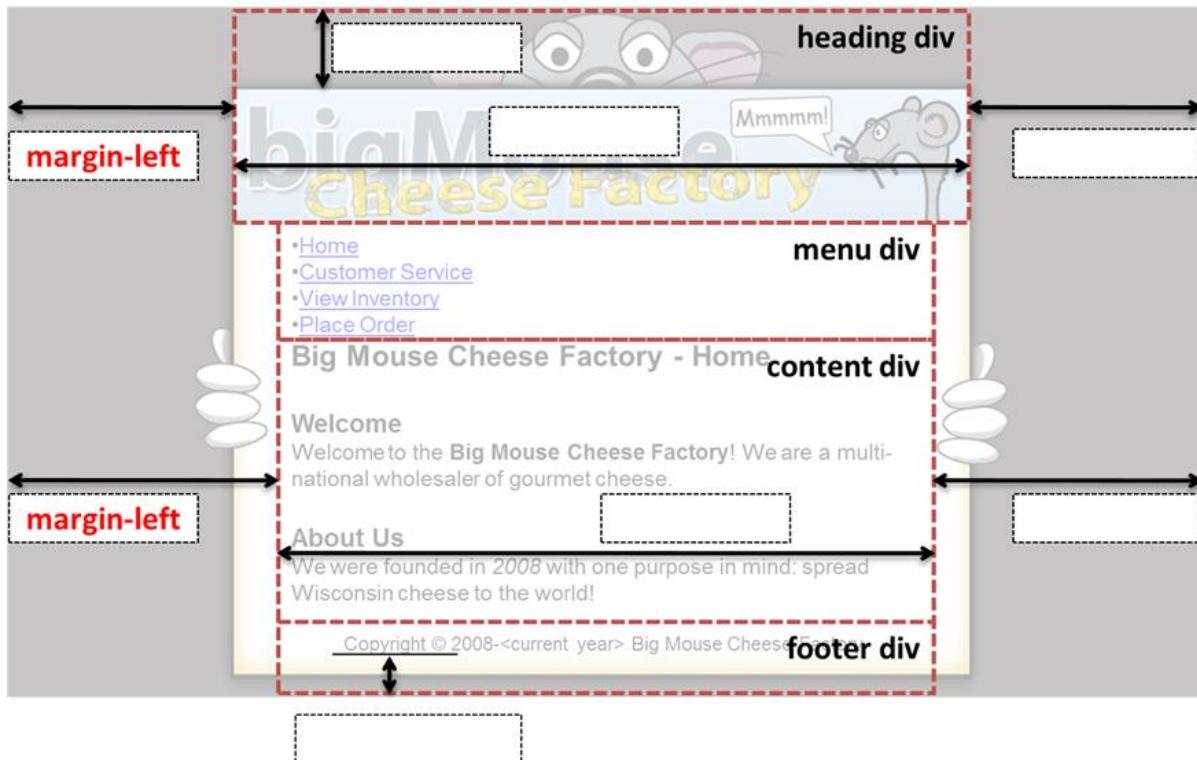


*Content is centered on the background images*

## Exercise: Applying Block-Element Styles

### Part 1: Determine Styles to Adjust

In the following figure, write the style used to arrange the corresponding block element. You do not need to give the size, just the style to use.



Write the style corresponding to each empty box

## Part 2: Arrange Block Elements

In this part, you will fix the layout so that the page content is centered properly on the background images. **After each step, it is a good idea to run your site to see the impact of the change.**

1. Complete all prior follow-along steps
2. The header content needs to move down 78 pixels so that the mouse head isn't covered.
  - Give the header div the id *divHeader*.
  - Apply an id selector with the style `padding-top: 78px;`.
3. There are two requirements to center a block element on the screen. The first is that the width of the block element has to be finite.
  - Add the style `width: 694px;` to the *divHeader* id selector.
4. The rest of the div elements need to be narrower to prevent the text from covering the mouse's hands. We could apply an id to each element and then use three selectors, but there is a way that involves less work:
  - Add the attribute `class="innerContent"` to the menu, content and footer div elements.
  - Add a new selector, `.innerContent` and apply the style `width: 620px;`.
5. What is the difference between a class selector and an id selector, in terms of syntax?

The second requirement to center block elements is that the left and right margins must be set to *auto*. Because all of the elements that we want to center are all direct children of *divBgL3*, we can select all of them simultaneously using the operator **>**.

Child-selection operator	<p><b>Syntax:</b></p> <p>Selector 1 &gt; Selector 2</p> <p>Select all elements matching selector 2 that are also children of elements matching selector 1.</p> <ul style="list-style-type: none"><li>• The parent and child selectors may be based on element, class or id</li><li>• The <b>&gt;</b> operator may be used multiple times within the same selector</li></ul>
--------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6. Add the selector: `#divBgL3 > div`
7. Within the selector from the previous step, apply the style `margin: 0 auto;`

There are many ways to apply the margin styles:

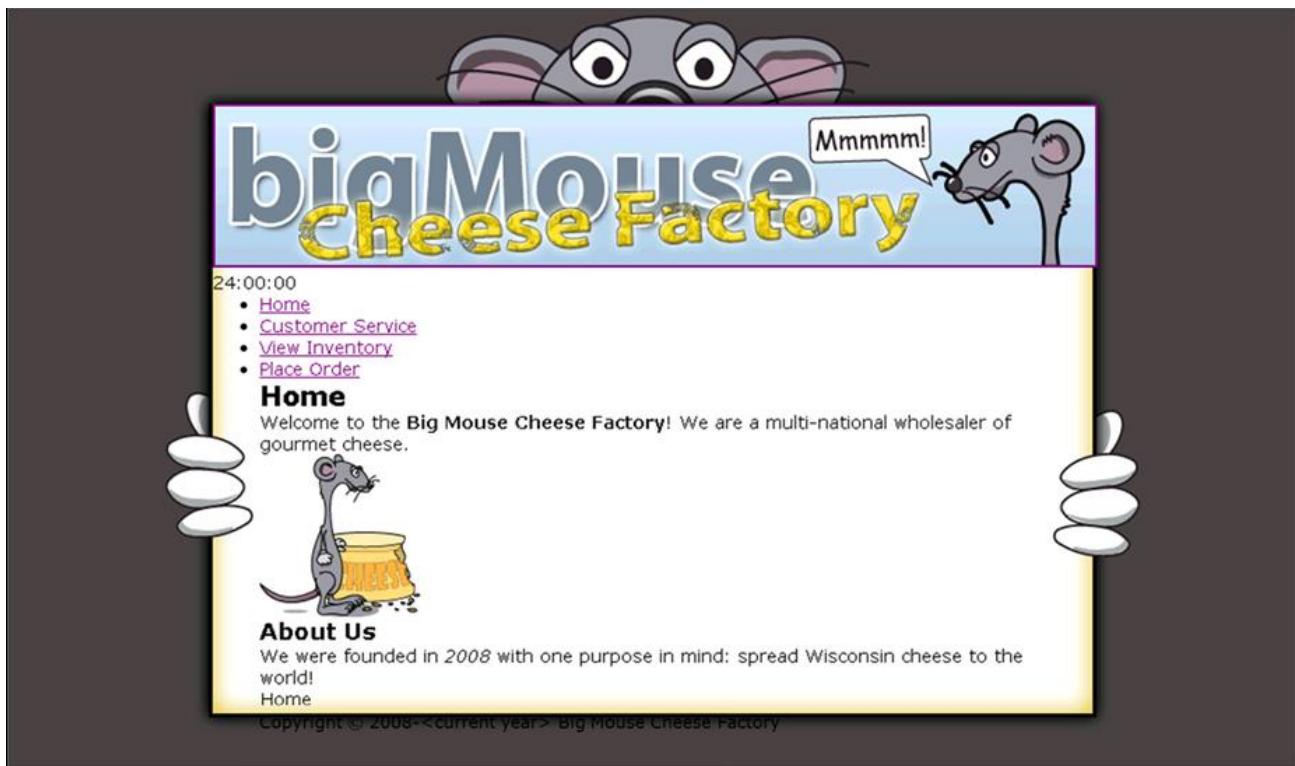
- Uniform margin to all sides: `margin: 5px;`
- Top/bottom, then left/right: `margin: 5px 10px;`
- Top, right, bottom, left (clockwise): `margin: 2px 3px 5px 1px`

Additionally, you can access the *margin-top*, *margin-right*, *margin-bottom* and *margin-left* styles individually. Padding styles can be applied in the same way.



Be sure to remember the different ways to apply margin and other block styles.

8. At this point, your web application should appear similar to the following:



*Web application after some adjustments*

9. The footer text needs to be moved onto the light portion of the background.
  - a. Add the id *divFooter* to the footer div.
  - b. Use an id selector to change the padding of the footer so that the bottom padding is 30px.
10. Center the copyright text by adding the style `text-align: center;` to *divFooter*.
11. The banner image has a border that is preventing it from aligning perfectly with the background. Use an element selector to apply the style `border-style: none;` to all *img* elements.

### Part 3: Headers and Links

1. We still need to account for h3 elements in *ExampleMaster.css*. Use an element selector to set the font-size of h3 tags to 110%.
2. The **color** style refers to the font color.
  - Add the style `color: #4083FF;` to all h1 and h2 elements.

So far, you have learned about element selectors, ID selectors (using a #) and class selectors (using a .). Additionally, there is a style of selector called a pseudo class that is prefixed by a colon (:).

pseudo-class	Classes that are dynamically added to or removed from elements based on the state of the page.
--------------	------------------------------------------------------------------------------------------------

3. The customer prefers that hyperlinks are more subtle. The color blue is fine, but links should not be underlined unless they have focus (via tabbing or a mouse click), are active (being followed) or the mouse cursor is hovering over them. Additionally, visited links should remain blue.
- Ensure that both visited and unvisited links remain blue and are not underlined by default. To do so, use the following selectors and styles:

```
a:link, a:visited  
{  
    color: blue;  
    text-decoration: none;  
}
```

The selector *a:link* refers to all anchor elements that contain the href attribute, while *a:visited* refers to all such elements that additionally have been followed by the user.

A single set of styles enclosed in braces can be applied to a set of distinct selectors separated by commas. In other words, the following two code segments are equivalent:

```
Selector1, Selector2, Selector3  
{  
    Style1;  
    Style2;  
}
```

```
Selector1  
{  
    Style1;  
    Style2;  
}  
  
Selector2  
{  
    Style1;  
    Style2;  
}  
  
Selector3  
{  
    Style1;  
    Style2;  
}
```

4. Next, have the links become underlined in certain situations:

```
a:focus, a:hover, a:active  
{  
    text-decoration: underline;  
}
```

:focus	Pseudo-class applied to an element that has focus
:hover	Pseudo-class applied to an element when the mouse cursor hovers over it
:active	Pseudo-class applied to an element when the mouse button clicks it and the mouse button is held down. It is removed once the mouse button is released.
text-decoration	Style used to apply additional decoration to text. Possible values include any combination of <i>underline</i> , <i>overline</i> , <i>line-through</i> and <i>blink</i> .

## Part 4: Final Adjustments

---

The menu currently takes too much vertical space. You can reduce the vertical space requirement by applying the `display: inline;` style to the `li` elements contained within the menu div.

display	<p>The style that adjusts how elements are displayed. Common values include:</p> <ul style="list-style-type: none"> <li>• <b>none</b>: The element is not displayed and doesn't take up space</li> <li>• <b>inline</b>: The element is displayed like an inline element</li> <li>• <b>block</b>: The element is displayed like a block element</li> </ul>
Descendant selector	<p><b>Syntax:</b></p> <pre>selector 1 &lt;space&gt; selector 2</pre> <p>Selects all elements matching selector 2 that are contained in the subtree below elements matching selector 1.</p> <p>Remember that the "&gt;" operator only selects direct children of selector 1, while the space operator selects all matching descendants.</p>

1. Apply the `display: inline;` style to all descendant `li` elements contained within the menu div.
2. Since you removed the default padding and margin, headings are a little tight. Fix this by increasing the `margin-top` style of your headings.
  - Use 20px for h1
  - Use 15px for h2
  - Use 10px for h3

3. If you navigate to the Inventory and Place Order pages, you'll notice that the content is so short that the hands of the mouse in the background image are clipped. To prevent this:
  - a. In ExampleMater.master, add an id to the div containing the content. Call it "divContent"
  - b. In ExampleMaster.css, add an ID selector on divContent with the style min-height: 200px;

## If you have time

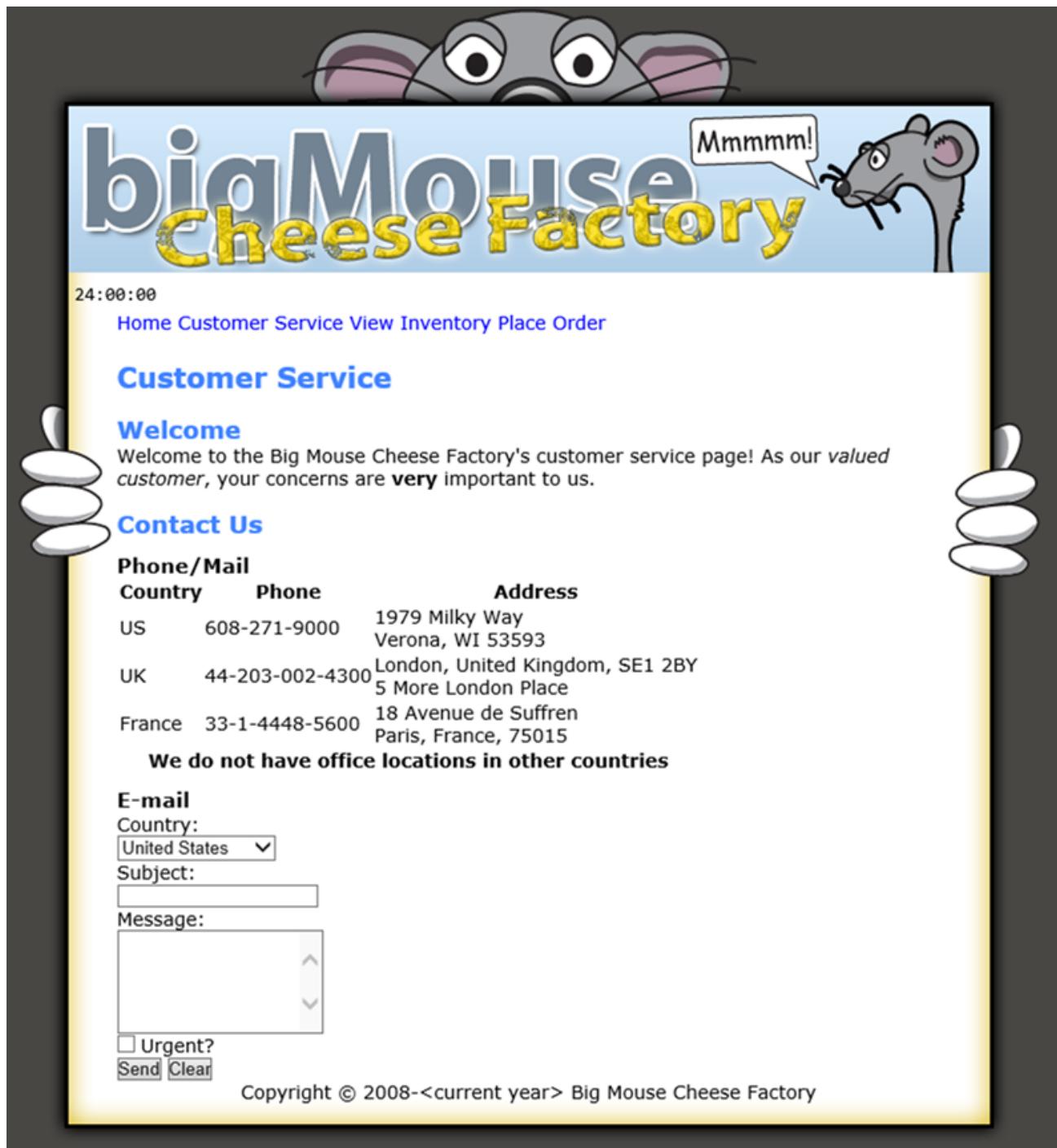
---

Experiment with the style of your custom clock web control as you see fit. You can apply a CSS class to ASP.NET elements using the *CssClass* attribute. This will apply the class to the outer div representing the clock.

## Wrap Up

---

1. After you finish your exercises, *Catalog.aspx* should look similar to the following (depending on how you formatted your clock):



*Final appearance of the customer service page*

2. What two styles are required to center a block element?

- \_\_\_\_\_
- \_\_\_\_\_

3. Fill out the following table that compares the various selection methods:

Method	Element	Class	Id	Pseudo-class
Syntax				
When to use	When selecting all elements of _____ type	When selecting one or more elements that are _____ the same type.	When selecting _____ element.	Selecting elements based on _____

4. Which CSS class name is more maintainable?

(A) Name based on what the element is used for	(B) Name based on applied styles
<pre>.menuItem {   font-weight: bold; } ... &lt;ul&gt;   &lt;li class="menuItem"&gt;     log in   &lt;/li&gt;   ... &lt;/ul&gt;</pre>	<pre>.bold {   font-weight: bold; } ... &lt;ul&gt;   &lt;li class="bold"&gt;     log in   &lt;/li&gt;   ... &lt;/ul&gt;</pre>

•

---

5. Why do you need to be careful with ID selectors when using server controls?

■

---



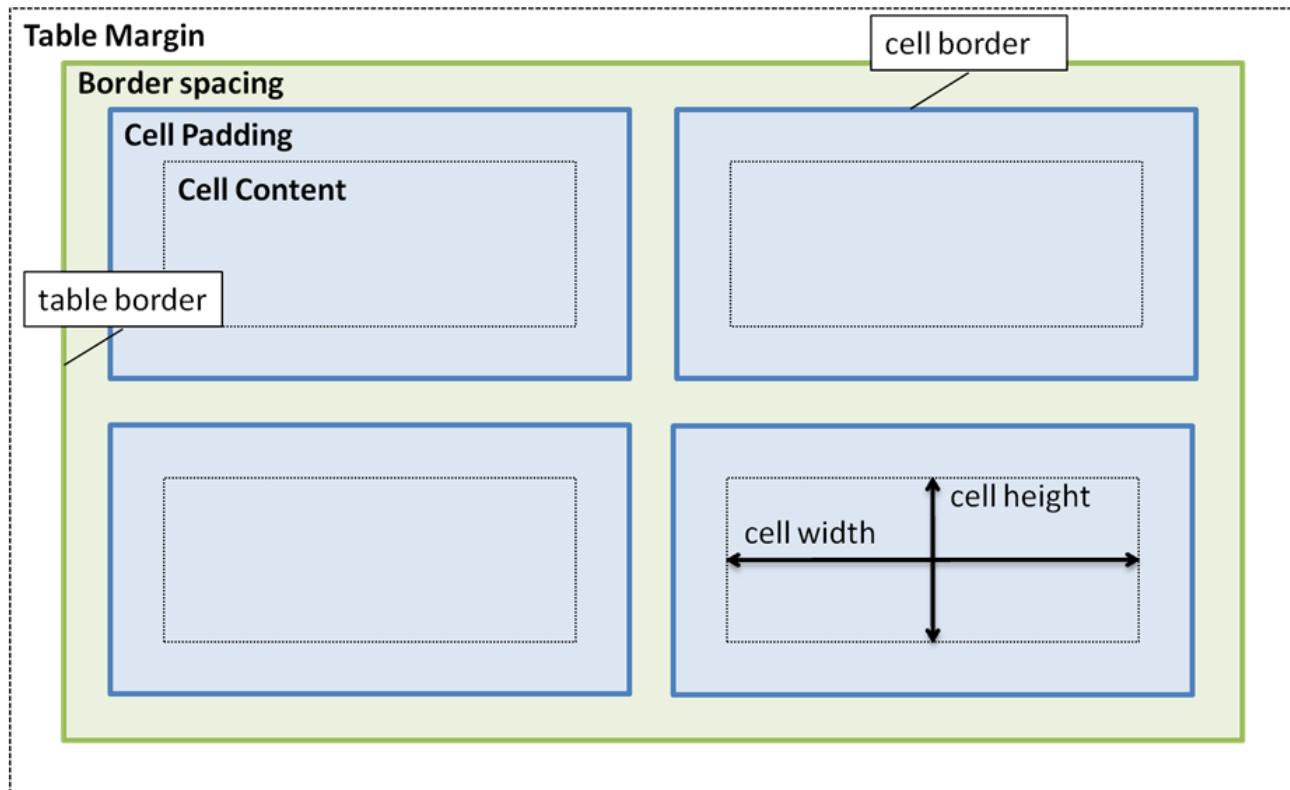
---

# Formatting Tables

In this section we will cover the table model and how to format tables.

## Arranging Table Elements

Table elements are similar to block elements, but have a few additional styles:



*Illustration of various table styles*

Table Margin	The transparent area around the table
Table Border	The area that divides the cell spacing from the table margin
Border spacing	The area between the cell borders and the table border. This area is filled with the background color and/or image of the table.
Cell border	The line area that divides the border spacing from the cell padding.
Cell padding	The area that divides the cell border from the cell content. The cell padding is filled with the background color and/or image of the cell.

Cell content	The area of the cell that contains sub elements and/or text. The height and width of a cell refers to this area.
--------------	------------------------------------------------------------------------------------------------------------------



Be able to calculate the total height or width of a table for the CSS quiz.

## Exercise: Adjust the layout of Tables

In this exercise you will adjust the layout of the contact information on the customer service page.

### Part 1: Table Formatting

1. You want to format all tables in the website. What kind of selector should you use? \_\_\_\_\_
  - Element
  - Class
  - Pseudo-class
  - Id
2. Correctly apply the following styles to adjust how all tables appear:
  - Tables
    - border: 2px solid black;
    - margin: 5px 0;
    - border-spacing: 3px;
  - Cells (remember there are two kinds of cells, header and data)
    - border: 1px dotted black;
    - padding: 2px;
3. Run and view *Customer.aspx* to verify that table appears as expected.
4. Navigate to *Order.aspx* and click through several of the wizard steps. What do you notice about how most of the forms are formatted?
  - \_\_\_\_\_
5. Assuming that we want all tables that contain data (not those that are used to layout forms) to look the same, what is the appropriate kind of selector to use? \_\_\_\_\_
  - ID
  - Class

6. Change how the table styles are applied so that they only apply to tables containing data, such as the customer service and inventory tables.
7. Run your site. Verify that tables on the order page do not have borders.

## Part 2: Finishing Touch

---

1. Apply the style border-collapse: collapse to the table. What happens?

■

---

2. Change the header background of the table to black and the text to white.
3. Answer the following wrap-up questions.

## Wrap Up

---

You should have applied a class rather than an ID, so that you can handle the situation where there are two (or more) distinct data tables on the same page. This way you just need to apply the class *dataTable*:

```
.dataTable
{
    border: 2px solid black;
    margin: 5px 0;
    border-collapse: collapse;
}

.dataTable th, .dataTable td
{
    border: 1px dotted black;
    padding: 2px;
}

.dataTable th
{
    color: White;
    background-color: Black;
}
```

1. Do table elements have padding?

■

---

2. What does border-collapse do?

- ---
  - ---
  - ---
3. How are most forms in Order.aspx organized?
- ---
  - ---
4. What is the total width of the table?
- Assume the following:
    - Content of column 1 is **50px** wide, column 2 is **100px** wide, and column 3 is **150px** wide.
    - Table borders are **2px** and cell borders are **1px**
    - Border spacing is **3px**
    - Cell padding is **2px**
    - Table margins are **0**
    - Borders are not collapsed
  - ---
  - ---
5. What if the borders are collapsed?
- **Hint:** **Border** spacing and some thinner borders go away. Try subtracting the difference from the previous answer.
  - ---

## If You Have Time

---

Begin studying the CSS language reference. There is material there that is on the exam but is not covered in this training companion:

[wiki > Foundations > Training > WTC > Reference > CSS](#)

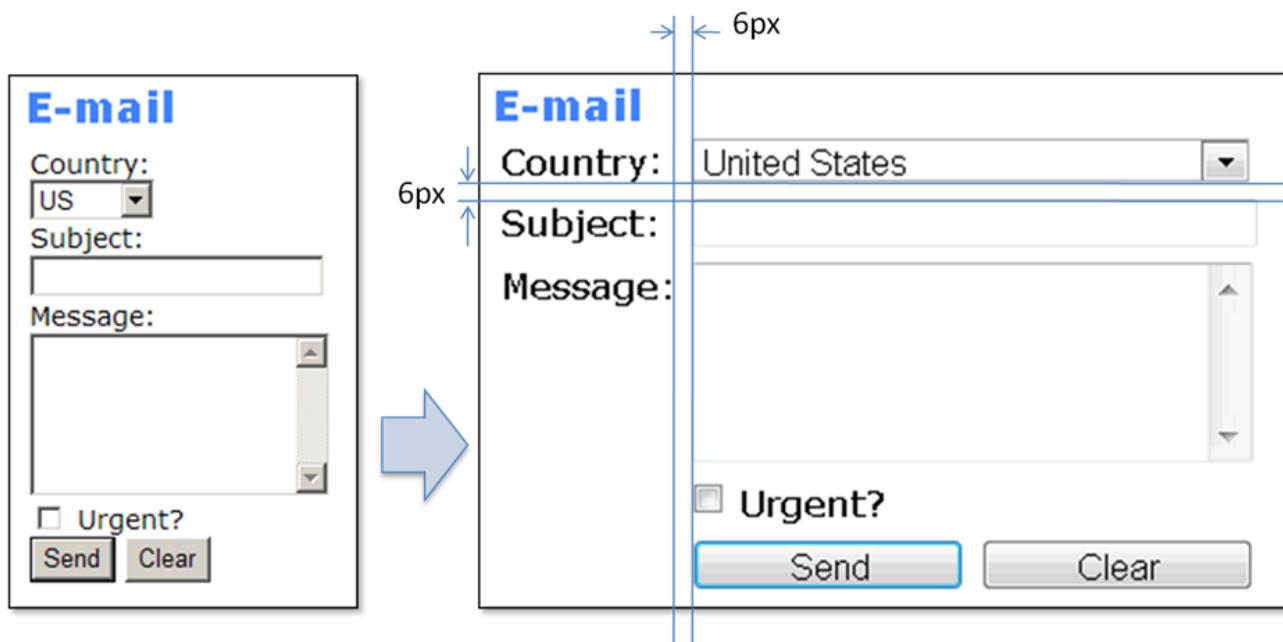
# Laying out Form Elements

In this section you'll learn how to arrange form elements for user input.

## Use Same UI Guidelines as VB

In general, the same guidelines will apply as those in all GUI applications at Epic.

- Controls must be consistently aligned
- Controls are separated by 6 pixels



*Desired format of the Email Form*

## Using Layout Tables

To simplify how forms are arranged, Epic conventions specify that a two-column table without borders should be used, where the first column contains labels and the second column contains controls.

**E-mail**

<b>Country:</b>	United States <input type="button" value="▼"/>
<b>Subject:</b>	
<b>Message:</b>	
	<input type="checkbox"/> <b>Urgent?</b>
	<input type="button" value="Send"/> <input type="button" value="Clear"/>

An example layout table. Note that the borders should not actually be displayed.

The above image illustrates what the table would look like if borders were included. The following CSS shows one way in which the UI guidelines could be followed:

```
table.layoutTable
{
    border-collapse: collapse;
}

table.layoutTable td
{
    padding: 3px;
    vertical-align: top;
}
```

Notice that the CSS class is connected to a specific kind of element, tables. Although this form of selector is not required, it prevents the associated styles from being applied to non-table elements that are assigned the *layoutTable* class.

## Exercise: Adjust the Layout of Forms

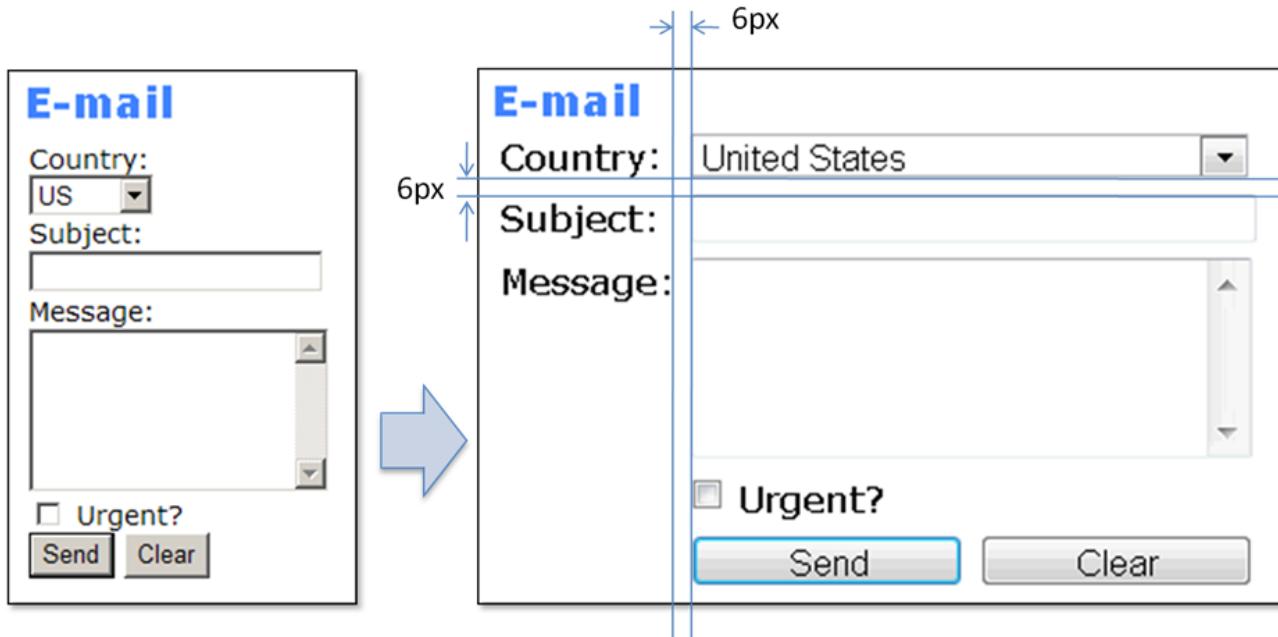
In this activity you will fix the layout of various forms in the Big-Mouse Cheese Factory site so that they meet the UI-style guidelines.

### Part 1: The Email Form

- By default, the *font-size* and *font-family* styles are not inherited by form elements. Modify *ExampleMaster.css* so that the same font family in the body also applies to input, select, button and textarea elements:

```
select, input, textarea, button
{
    font-family: inherit;
    font-size: inherit;
}
```

2. Fix the layout of the email form (using a layout table) so that it looks similar to the following screenshot:



Desired format of the Email Form



#### Form elements don't line up perfectly in terms of width.

That is, even if you assign the same width to input, select and textarea elements, they still won't line up properly. You can try to correct this if you like, but it is browser dependant.

Foundations controls will correct this issue.

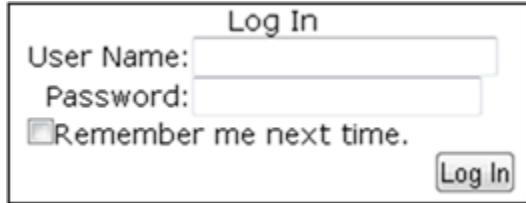
## Part 2: Create the login page

In this part of the exercise, you'll create the login page. In a later exercise you'll activate this page so users can authenticate themselves.

The learning objective is to learn how Microsoft-written ASP.NET controls can be styled.

1. Create a new Web Form with Master page called *Login.aspx*.
2. Add a link to *Login.aspx* to the menu on your master page.

3. Within the content, place a new *asp:Login* control from the toolbox.
  - **Toolbox > Login (heading) > Login**
4. Run your site and navigate to the login page. Notice that the style and layout is not acceptable:



The image shows the standard ASP.NET Login control layout. It features a title "Log In" at the top left. Below it are two text input fields: "User Name:" and "Password:", each with its own label. To the right of the "User Name:" field is a "Remember me next time." checkbox. At the bottom right is a large "Log In" button.

*Default layout of the asp:Login Control*

5. Fix the login control so it looks similar to the following:



The image shows a stylized version of the login control. The title "Log In" is bold and centered at the top. Below it are two text input fields: "User Name:" and "Password:", each with its own label. The "Log In" button is located at the bottom left and has a blue border, indicating it is the active element.

Note that you can apply CSS classes to the entire control, text boxes and title text individually:

```
<asp:Login ID="loginForm" runat="server" DisplayRememberMe="false"
  CssClass="loginTable">
  <TextBoxStyle CssClass="loginTextBox" />
  <TitleTextStyle CssClass="loginTitleTextStyle" />
</asp:Login>
```

Try your best to style the login control yourself. If you get stuck, refer to the following styles:

```
.loginTable
{
  text-align: left;
  border-collapse: collapse;
}

.loginTable td
{
  text-align: left;
  padding: 3px;
  vertical-align: top;
}

.loginTextBox
{
```

```
width: 200px;  
}  
  
.loginTitleTextStyle  
{  
    font-size: 16px;  
    font-weight: bold;  
}
```



Most complex ASP.NET controls expose their various parts using nested styling elements. Use IntelliSense to discover what is available for a particular control.

## If you have time

1. Certain selectors within *ExampleMaster.css* are actually specific to pages other than the master page. Move these selectors/styles to CSS files that are only included with the pages for which they are relevant. You'll need to include an *asp:ContentPlaceholder* in the head of the master page to get this to work.
  - **HINT:** Look ahead to the first few steps of the next exercise for an example of how to do this.
2. In *Order.aspx*, the wizard step with id *stpShippingMethod* needs to be formatted. Arrange the controls using a layout table.
3. Format your custom billing control, *Billing.ascx*.

## Wrap Up

1. What are the advantages of layout tables?
  - \_\_\_\_\_

2. What disadvantages are there?
  - \_\_\_\_\_
  - \_\_\_\_\_
  - \_\_\_\_\_
  - \_\_\_\_\_



# Resolving Conflicting Styles

A common problem is to have two separate values applied to the same style by different selectors. When this happens, it is important to understand which value will win, and why.

## Style Conflicts

The following is an example of HTML and CSS code that includes conflicting styles:

### HTML:

```
<body>
    <div id="bg" class="bg">
        What color am I?
    </div>
</body>
```

### CSS:

```
div
{
    background-color: White;
}
.bg
{
    background-color: Black;
}
#bg
{
    background-color: Red;
}
body > #bg
{
    background-color: Green;
}
```



What color is the div background?

Hint: Which selector do you think is the most specific?

- 

\_\_\_\_\_

## Cascade Algorithm

The cascade algorithm is used to determine the style applied to each element property. The algorithm is as follows:

For each element  $E$ :

1. Collect all styles in selectors that select  $E$
2. Order styles by *selector specificity*
3. For each property  $P$  of element  $E$ :
  - a. Apply the style from most specific selector
  - b. If no applied style: *Inherit style from parent*
  - c. If not an inherited style: *Apply browser default*

## Specificity Calculation

Points are awarded to an assigned style based on its selector:

Style (most specific)	ID	Class	Element (least specific)
1 if the styles are assigned in the style attribute of an element's markup, 0 otherwise	The count of IDs in the selector	The count of all <b>classes, pseudo-classes</b> and <b>attribute</b> selectors.	The count of all <b>elements</b> in the selector.

The specificity of two selectors are examined from left to right. When a difference is identified, the result is returned.

For example:

- 1,0,0,0

Is more specific than:

- 0,3,1,2

Because the value of the Style score is higher in the first selector score when compared to the second score:

- **1,0,0,0 > 0,3,1,2**

Another example:

- 0,5,6,2
- 0,5,6,3

In this case, the style, id and class scores are all the same, so the tie breaker come from the Element score:

- 0,5,6,3 > 0,5,6,2

## Examples

Selector	Specificity Score	Specificity Ranking 1 (most) - 5 (least)
div > .bg [type="button"]	0,0,2,1  1. 0, because it is applied in CSS, not the style attribute of HTML 2. 0, because there are no # characters in the selector 3. 2, because there is one class (. character) and one attribute (set of square brackets) 4. 1, because there is one HTML element (div)	
div		
.bg		
<div style="...> ... </div>		
div > #bg .bg:hover		



### Do not use !important in your selectors

The `!important` style modifier is a way to give one style precedence over another. For example, the color of the h1 text will be red, even though the selector with higher specificity is orange:

```
h1 {color: red !important;}  
body h1 {color:orange;}
```

If `!important` is overused, conflicting styles become increasingly more difficult to debug.

## Exercise: Fix the Details Page

In this exercise you will practice resolving conflicting styles.

### Scenario

In the Big-Mouse Cheese Factory example, administrators want a page where they can edit certain details of a cheese product. The intention is for read-only fields to be grayed out, while editable fields have a white background. As shown in the example below, even though editable fields are applied the CSS class `editable`, their backgrounds are not changing from gray to white.

Your task is to find the source of this issue and resolve it.

A screenshot of a web application interface for editing cheese details. On the left, there's a large image of a wedge of cheese with the word "BLUE" written on it. To the right, there's a title "Details" and a form with several fields. Above the form, a CSS rule is shown in a callout box:

```
.editable  
{  
background-color: White;  
}
```

The form contains the following data:

Item:	110
Name:	American Blue
Weight:	25 lb
Price:	\$90.00
Description:	Strong blue cheese flavor running throughout

Below the form are three buttons: "Edit", "Accept", and "Cancel". In the first version of the form (left), all fields are grayed out. In the second version (right), the "Name", "Weight", and "Price" fields are white, indicating they are editable, while the "Item" field remains grayed out.

An example of a desired style not being applied as expected

## Part 1: Get the details page

1. Copy the text for the details page from this file into the content placeholder of *Details.aspx*:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\Details.txt
2. In the master page, add a new content placeholder within the head element. Give it the id *cphHead*:

```
<head runat="server">
    <title>Big Mouse Cheese Factory</title>
    <link rel="stylesheet" href="Syste.Master.css" ... />
    <asp:ContentPlaceHolder ID="cphHead" runat="server">
        </asp:ContentPlaceHolder>
</head>
```

3. Add the following CSS file to your Inventory folder:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\css\Details.css
4. In *Inventory/Details.aspx*, add a new *asp:Content* control with ContentPlaceHolderID="cphHead".
5. Link to Details.css within this control:

```
<asp:Content ID="contentHead" ContentPlaceHolderID="cphHead"
              runat="server">
    <link rel="stylesheet" href="Details.css"
          type="text/css" media="all" />
</asp:Content>
```

## Part 2: Fix the Page

1. Run the website and navigate to *Details.aspx*.
2. Verify that the three editable fields are incorrectly grayed out:
  - Name
  - Weight
  - Price
3. Determine where the conflict is and devise a way to fix it. The goal is for the *Item* textbox to remain gray, while the others have white backgrounds.



You can use the computed styles option in the IE Developer Tools to help with this task.

1. Using the DOM explorer, select one of the elements that is styled incorrectly
2. On the right half of the developer tools, select the **Computed** tab.
3. Find the background style and see why `.editable` is not being applied.

## Wrap Up

---

1. What was the conflict?

- ---

---

---

2. There are several ways to fix the conflict. The instructor will provide one method you could have used. Write this method in the space below:

- ---

---

---

## Exercise: Predict the Style

---

In this exercise you must determine what the text and background colors of elements in a webpage are. Consider the following HTML and CSS, and write your answers in the HTML tree below.

### Markup:

```
<html>
<head>
    <title>Web Page Title </title>
    <link href="style.css" ... />
</head>
<body>
    <div class="directions">
        <h1>Can You Predict the Style?</h1>
        <p>Your job is to predict the background color and font
            color for each of the elements on this page.
            Can you do it?</p>
    </div>
    <div class="tableSection">
```

```
<h2>Table with Colored Cells</h2>
<table>
    <tr><td>cell 1</td><td id="secondCell">cell 2</td></tr>
    <tr><td>cell 3</td><td>cell 4</td></tr>
</table>
</div>
<div class="listSection">
    <h2>My List</h2>
    <ul>
        <li>Bicycling</li>
        <li id="kick">Kickboxing</li>
        <li>Step Aerobics</li>
        <li>Hiking</li>
    </ul>
</div>
<div>
    <p>This fun exercise has been brought to you by Epic</p>
</div>
</body>
</html>
```

**style.css:**

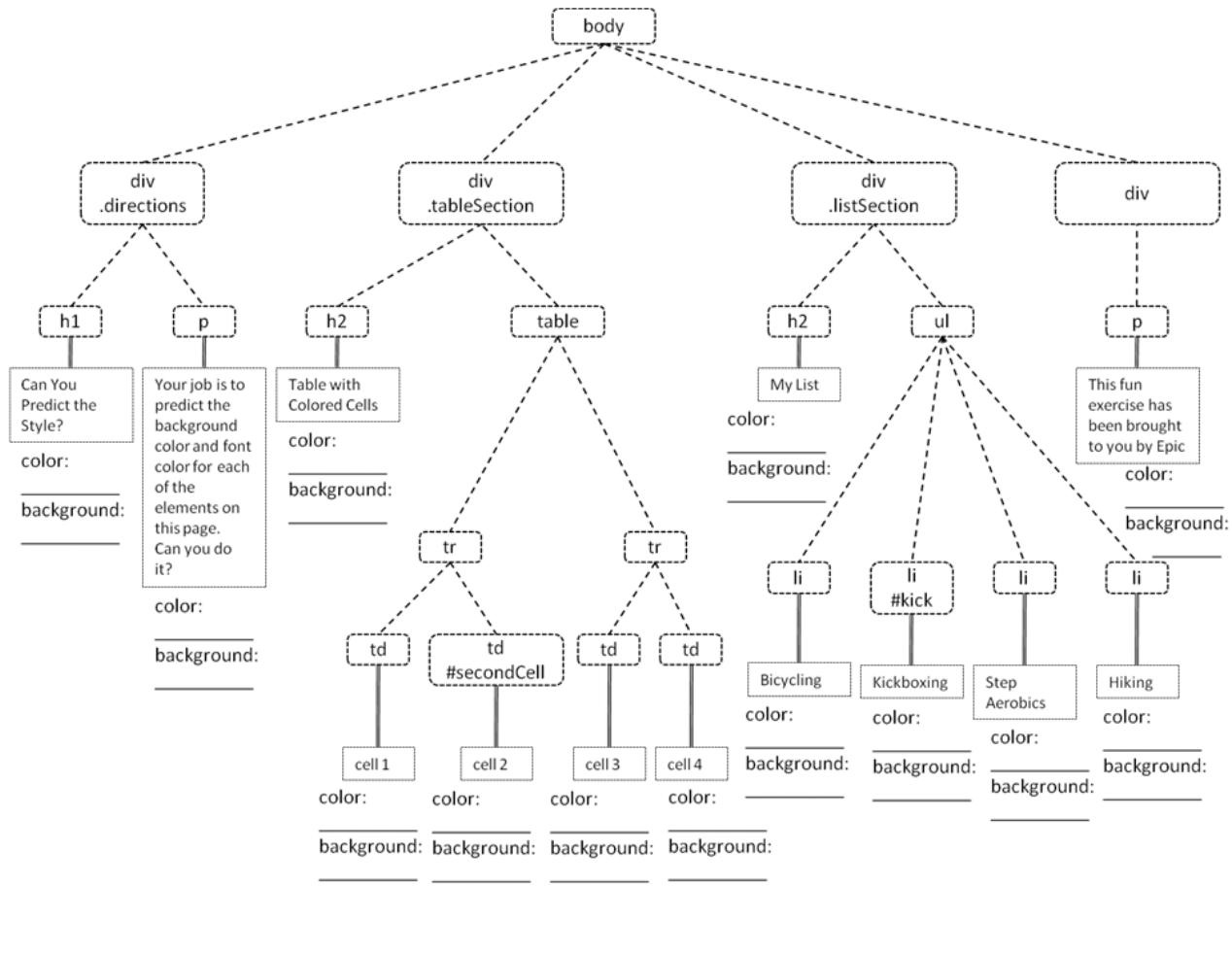
```
body {
    background-color: #646467; /* grey */
    color: White;
}

.directions, .tableSection table {
    background-color: #86A4FF; /* blue */
}

.tableSection {
    background-color: #89B255; /* green */
}

.listSection, #secondCell {
    background-color: #FFB39F; /* peach */
    color: Black;
}

.listSection h2, #kick {
    color: Maroon;
}
```



# Arranging Elements

## Follow Along: Floating Block Elements

It is possible to arrange block elements alongside each other that are normally stacked vertically.

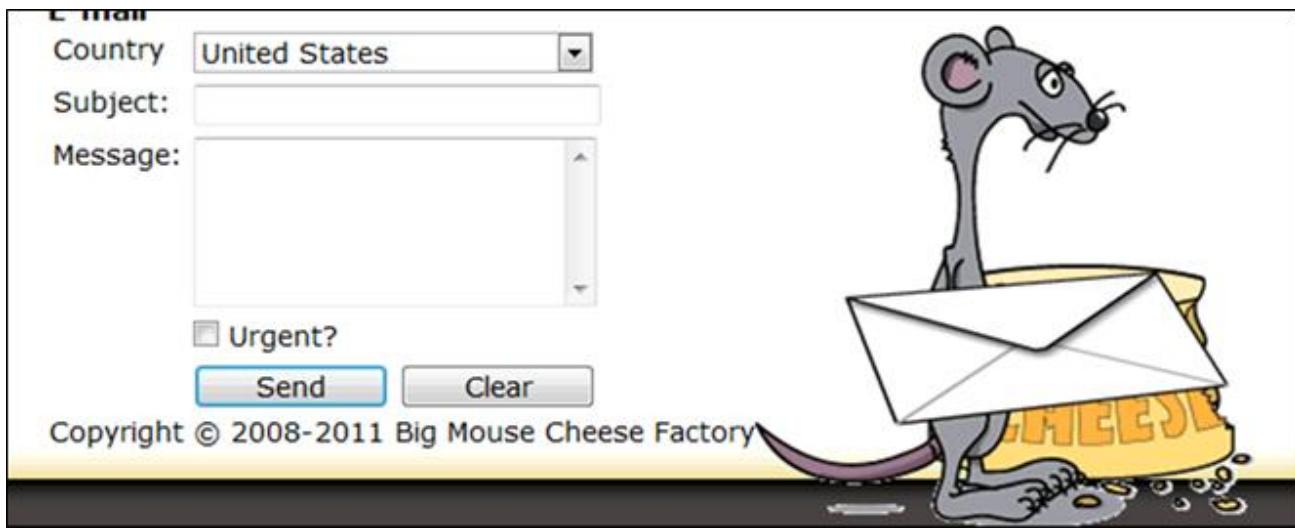
1. Add the following image to *Customer.aspx*, just above the email form:

```
<h3>E-mail</h3>

<img runat="server"
    style="float:right; display:block;"
    alt="Email mouse"
    src("~/images/BIG-MOUSE-CONTACT-256x257.png"
    width="257" height="257" />

<table class="layout">
    ...
</table>
```

2. Run your site and navigate to *Customer.aspx*.
3. Notice that the image is floating to the right, but it is displacing the page footer:



*Floating block element is displacing the footer*

4. Add the style `clear: both;` to the `#divFooter` selector in *ExampleMaster.css*.
5. Refresh *Customer.aspx*. You should notice that the footer is not displaced anymore:

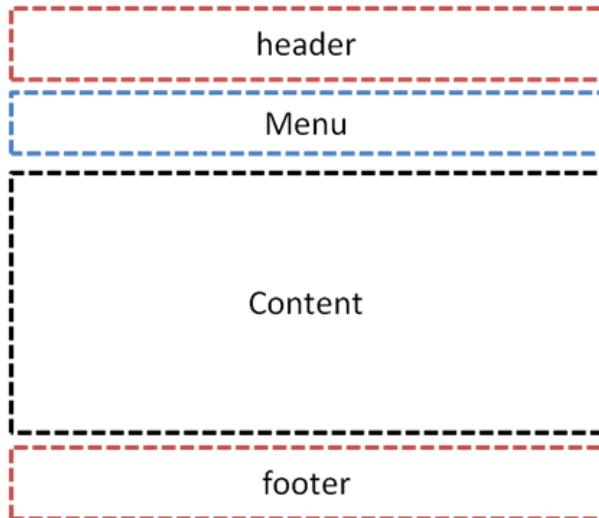


*Footer is clear of floating elements*

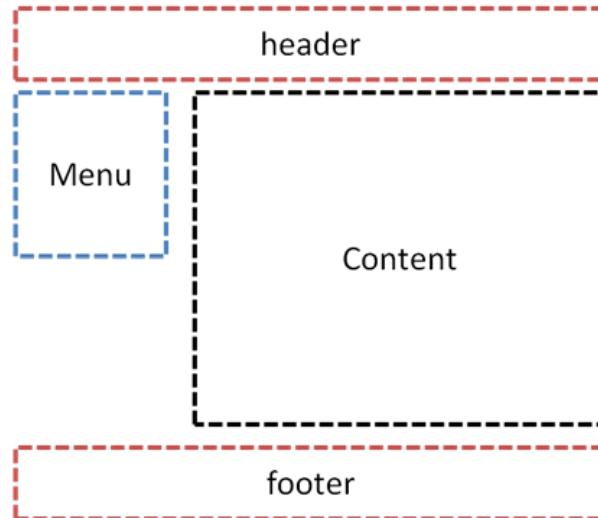
## Alternate Page Layouts

One application of floating block elements is to achieve an alternate layout on the master page. For example, you could float the menu div to the left of the content, which would be an alternative to displaying the menu list items in-line:

### Current Menu Location



### Alternate Location



*Floating the menu div for an alternative layout*

The additional styles you would need for the alternative layout are:

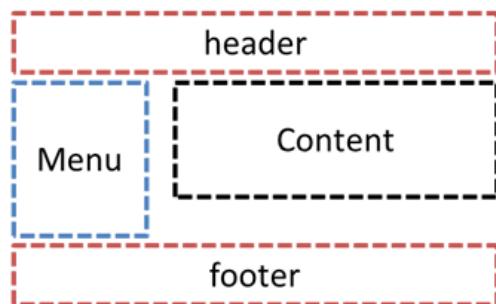
- `#divMenu { float: left; }`
- `#divFooter { clear: left; }`

## Browser Differences

When floating elements, you are likely to encounter differences in how the same page is rendered on different browsers. For example, in Firefox the menu will not actually appear to the left of the content div unless the content div has a left-margin that is wide enough to accommodate the menu. Internet Explorer, on the other hand, will float the menu to the left:

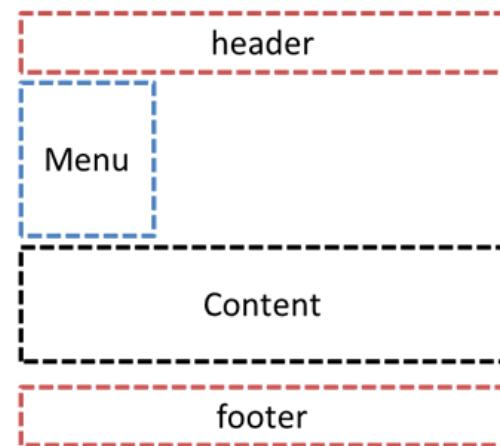
### Internet Explorer

- `#divMenu → float: left;`
- `#divFooter → clear: left;`



### Firefox

- `#divMenu → float: left;`
- `#divFooter → clear: left;`



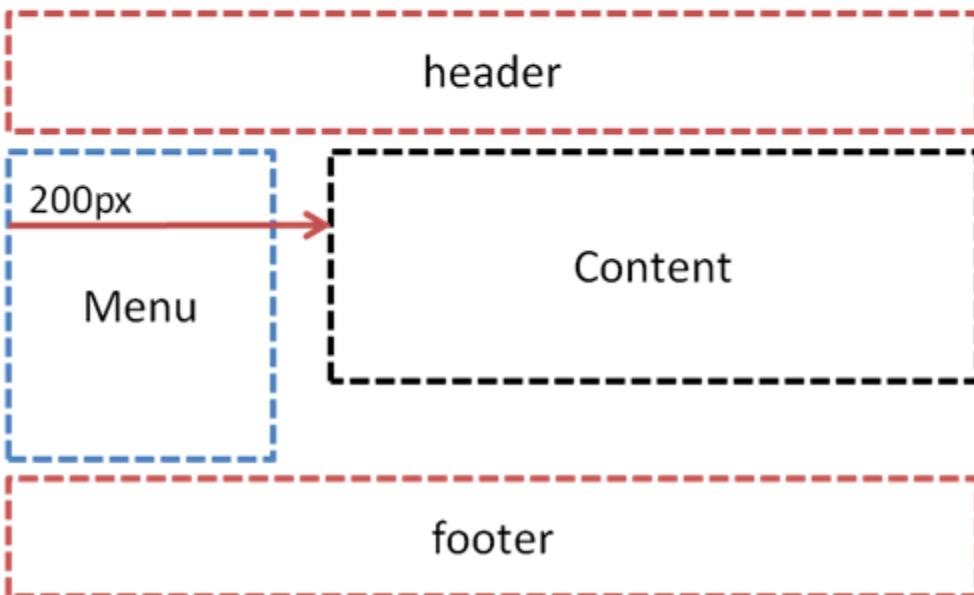
## Consistent Browser Arrangement

If the content is given a large enough left margin, both browsers will render the page in the same way. Assuming the menu div will fit within a 200px margin, the following styles will achieve the desired result in both browsers:

- `#divMenu { float: left; }`
- `#divFooter { clear: left; }`
- `#divContent { margin-left: 200px; }`

## Firefox & Internet Explorer

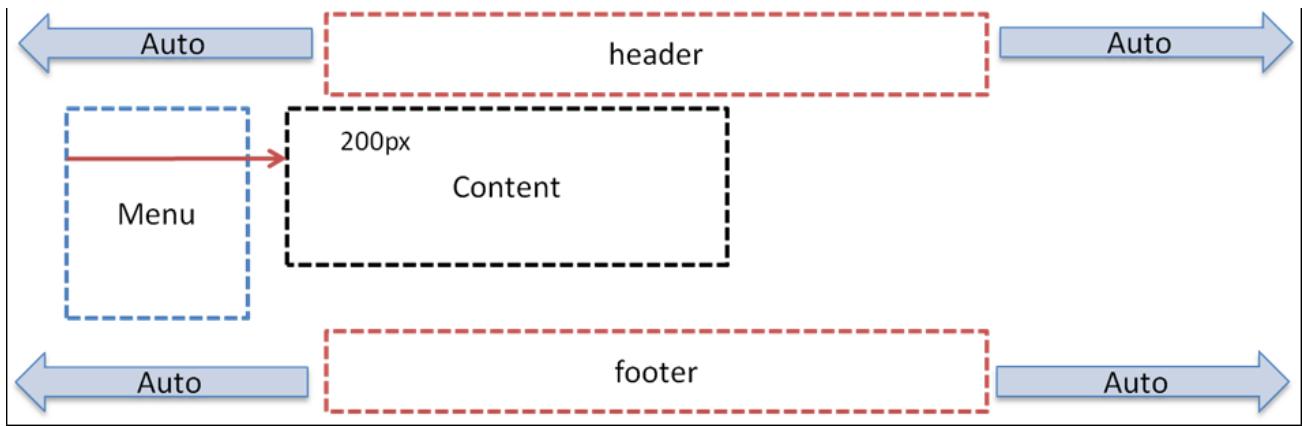
- #divMenu ➔ *float: left;*
- #divFooter ➔ *clear: left;*
- #divContent ➔ *margin-left: 200px;*



This is why it is critical to test your page in all supported browsers, not just one.

## Centering Pairs of Block Elements

Suppose you wanted to center the various divs on the page in the middle of the browser window, similar to the Big-Mouse Cheese Factory's master page. If you tried this with the alternate layout, you would run into the following problem:



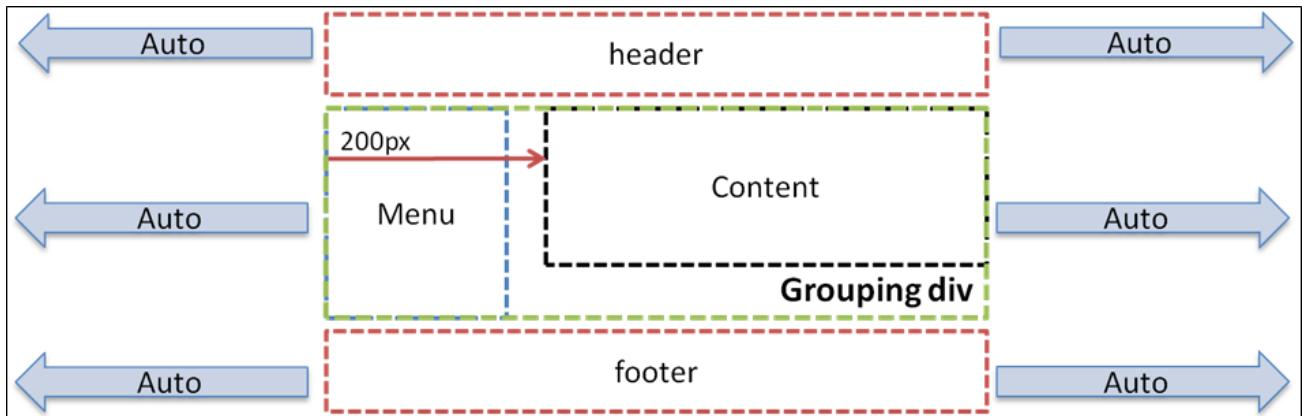
*The menu and content div elements are not centered*



Why are the menu and content divs not centered?

• \_\_\_\_\_

In this case, the solution is to wrap the menu and content div elements in yet another "grouping" div. The grouping div can then be assigned a finite width and left and right margins of auto, so that it will be centered along with the header and footer:



*Illustration of a grouping div*

## The Position Style

By default, HTML elements are arranged using their position in the markup. For most pages, this works fine. However, there are some cases where you may want to set the exact position of the element based on Cartesian coordinates relative to different parts of the page. The *position* style can be used for this purpose.



Foundations positions popups with this style in HyperspaceWeb.

The following examples use this content:

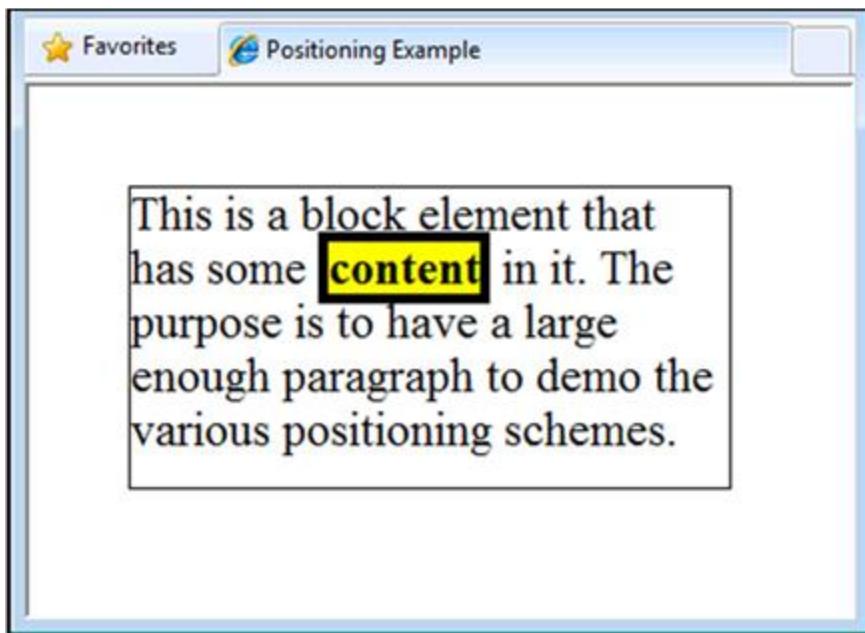
```
<body>
  <div>
    This is a block element that has some <strong>content</strong> in it.
    The purpose is to have a large enough paragraph to demo the various
    positioning schemes.
  </div>
</body>
```

## Static

---

Static positioning is the default position of an element in the flow layout of the page. In this case, the strong element appears inline with the paragraph.

```
strong
{
  position: static;
  background-color: yellow;
  border: 5px solid black;
}
```

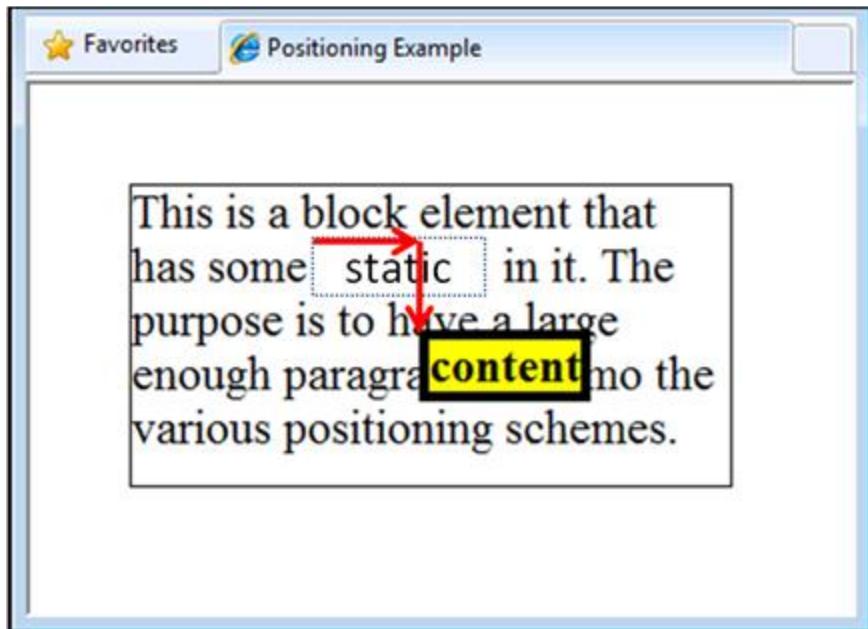


*Static Position Example*

## Relative

Relative positioning is used to offset an element from its static position. The offset can be achieved by setting values to the top, right, bottom or left styles. For example, a relative position can be used to move the strong over and down by 50 pixels:

```
strong
{
    position: relative;
    top: 50px;
    left: 50px;
    background-color: yellow;
    border: 5px solid black;
}
```



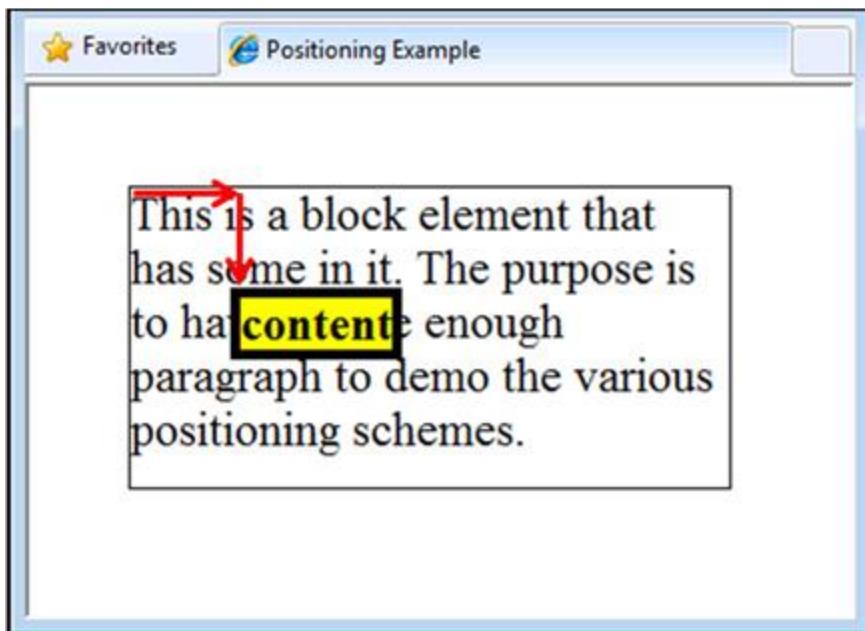
*Relative position offsets from static position*

## Absolute

Absolute positioning is the same as relative, but the origin is moved to the nearest containing block element. One important detail is that for an element to serve as a container, it must not be positioned statically.

In this example, the strong element is positioned absolutely within its containing div, which is positioned absolutely within the body element.

```
strong
{
    position: absolute;
    top: 50px;
    left: 50px;
    background-color: yellow;
    border: 5px solid black;
}
div
{
    position: absolute;
    top: 50px;
    left: 50px;
}
```



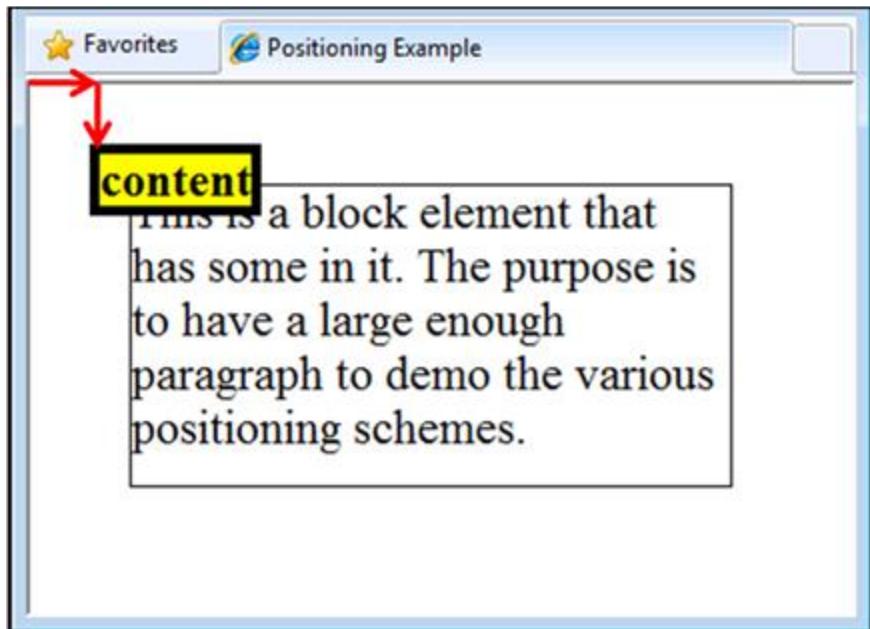
Absolute Position Example

## Fixed

---

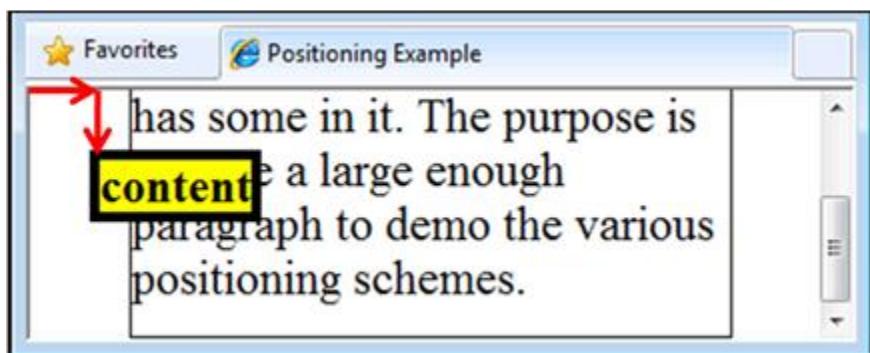
Fixed positioning moves the origin to the top-left of the browser window.

```
strong
{
    position: fixed;
    top: 30px;
    left: 30px;
    background-color: yellow;
    border: 5px solid black;
}
```



*Fixed Position Example*

Elements that have a fixed position also do not scroll with the content:



*Fixed Position with Scrolling Content*

# Lesson 4: Designing Client Behavior

<b>The Big Picture.....</b>	<b>4•5</b>
<b>Javascript Overview.....</b>	<b>4•7</b>
<b>Connecting Client Scripts.....</b>	<b>4•8</b>
Follow Along: Create a New JavaScript File.....	4•8
Exercise: Page-Specific Scripts.....	4•8
Part 1: Create and connect the script file.....	4•9
Part 2: JavaScript IntelliSense in Visual Studio.....	4•9
Part 3: Using Namespaces.....	4•10
Part 4: Naming your functions .....	4•13
Part 5: IntelliSense for Functions.....	4•13
Part 6: Finding JavaScript Errors.....	4•14
Wrap Up .....	4•16
<b>Subscribing to Client Events .....</b>	<b>4•18</b>
Scenario: Connect to Click Event of Send .....	4•18
Follow Along: Prevent Post Back.....	4•18
Follow Along: Connect to button click events.....	4•19
The Client ID of Server Controls.....	4•19
Managing Client IDs.....	4•20
Managed Controls Example.....	4•21
Exercise: Subscribe to Client Events.....	4•23
Part 1: Add Training.Core.Controls.Web to your solution .....	4•23
Part 2: Add a PageSettings Control to CustomerService.aspx .....	4•25
Part 3: Behavior of the Customer Service Page .....	4•26
Part 4: Connect to the Click Event of btnSend .....	4•28
Part 5: Understanding Function.CreateDelegate .....	4•30
Part 6: Using the Event Object.....	4•31
Part 7: Foundations Shortcuts .....	4•31
Wrap Up .....	4•32
If you have time .....	4•33
<b>Validating Forms.....</b>	<b>4•34</b>
Exercise: Validate Email Fields .....	4•34
Part 1: Validate the Form.....	4•35
Part 2: Save the Form Controls in Fields.....	4•38
Part 3: Debugging.....	4•42
If you have time .....	4•42

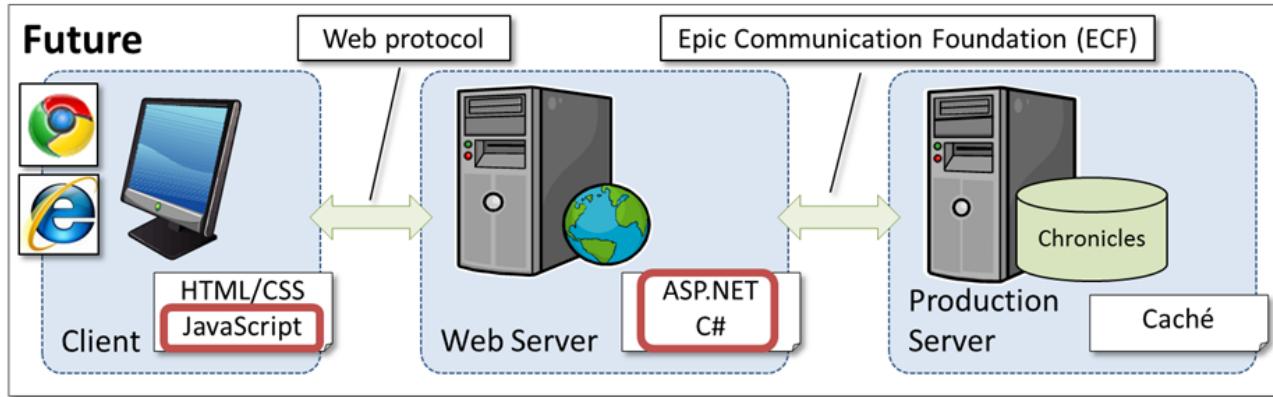
Wrap Up.....	4•42
<b>Changing Content and Layout .....</b>	<b>4•44</b>
Behavior after First Exercise .....	4•44
Behavior after Second Exercise.....	4•44
Document Object Model (DOM).....	4•45
Exercise: Display Error Messages.....	4•46
Part 1: Prepare your Solution.....	4•47
Part 2: Shortcuts and Browser compatibility issues.....	4•50
If you have time.....	4•53
<b>Creating Script Controls.....</b>	<b>4•54</b>
Clock Should Track Time.....	4•54
The ScriptControl.....	4•54
Working with Timeouts .....	4•55
Working with Intervals .....	4•55
Maintaining Encapsulation.....	4•55
Exercise: Make the Clock Keep Time.....	4•56
Part 1: Change Clock to a ScriptControl .....	4•56
Part 2: Setup the clock to tell time.....	4•59
Part 3: Passing server-control properties to the client.....	4•61
Part 4: Accessing custom controls on the page.....	4•62
Part 5: Casting for IntelliSense .....	4•64
Wrap Up.....	4•66
<b>Loading Data on the Client.....</b>	<b>4•67</b>
Populate the Inventory at Run Time.....	4•67
Load Pattern .....	4•67
Simulated Load Pattern.....	4•69
<b>Using the CollectionGrid to Edit Data.....</b>	<b>4•70</b>
Advanced Features of the Collection Grid.....	4•70
Parameter Validation .....	4•70
Exercise: Populate the Inventory Table.....	4•71
Part 1: Get the sample data.....	4•71
Part 2: Get the CollectionGrid ScriptControl.....	4•73
Part 3: Create the Cheese JavaScript Class.....	4•75
Part 4: Populate the inventory list .....	4•79
Exercise: Make the Cheeses Editable.....	4•80
If you have time.....	4•87





# Designing Client Behavior

## The Big Picture



*Client behavior is written in JavaScript, server behavior is in C#/ASP.NET*

The behavior of a web application can be written either on the server in C#/ASP.NET, or it can be written on the client in JavaScript. What are the advantages of one over the other?



What are the advantages of client behavior?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_



What are the advantages of server behavior?

- 

- 

-

# Javascript Overview

To learn the basics of JavaScript, study [the JavaScript Language Basics wiki](#). After completing the material from the wiki:

1. Sign up to take the [JavaScript Certification Exam](#)
2. Continue working through the remainder of this lesson

# Connecting Client Scripts

Like style sheets, there are numerous ways to connect client scripts to a web page:

- Embedded in the markup of the page inside a script element (<script>code</script>)
- Connected to the event attributes of HTML elements
- **(preferred)** Within an external JavaScript file (\*.js)

## Follow Along: Create a New JavaScript File

1. Create a new JavaScript file in your Epic.Training.Example.Web.Pages project:
  - **Training.Example.Web.Pages > right-click > Add > New Item > JavaScript File > Shared.js**

Rather than connecting *Shared.js* to the page using a link in the head element (as is standard), we will instead use the ASP.NET *ScriptManager* control. This will have the additional benefit of also sending the Microsoft Ajax Library scripts.

2. Add a script manager to the master page, just inside the form element:

```
<form ...>
    <asp:ScriptManager ID="sm" runat="server">
        <Scripts>
            <asp:ScriptReference Path="~/Shared.js" />
        </Scripts>
    </asp:ScriptManager>
    ...

```

3. Add the following code to *Shared.js*, to ensure that it is loading as expected:
  - `alert("Hello World!");`
4. Run the web application to verify that the script is loading.
5. Comment out the alert.

## Exercise: Page-Specific Scripts

In addition to scripts that are shared with all pages via a master page, some scripts are specific to a single page, user control or web control. In this exercise you will create such a script for the customer service page.

Along the way, you will explore several quirks of the JavaScript language, and install Visual Studio extensions to make your experience programming in JavaScript a bit easier.

## Part 1: Create and connect the script file

1. Create a new file named *CustomerBehavior.js* in the root folder. It should appear just below *Customer.aspx* in the solution explorer.
2. Add a script manager proxy to the bottom of *Customer.aspx*, just inside the content control. The proxy is in the toolbox under AJAX Extensions.

```
...  
<asp:ScriptManagerProxy ID="smp" runat="server">  
  </asp:ScriptManagerProxy>  
</asp:Content>
```

ASP.NET allows only one script manager per page. Since you already added a script manager to *ExampleMaster*, you cannot add another one to *Customer.aspx*. To access the original script manager, ASP.NET provides a control called *ScriptManagerProxy*.

3. Add a script reference to *CustomerBehavior.js*:

```
...  
<asp:ScriptManagerProxy ID="smp" runat="server">  
  <Scripts>  
    <asp:ScriptReference Path="~/CustomerBehavior.js" />  
  </Scripts>  
</asp:ScriptManagerProxy>  
</asp:Content>
```

4. Test the script to make sure it is loading:
  - a. Add an alert to *CustomerBehavior.js* that indicates it is loading
  - b. Run your website
  - c. Navigate to *Customer.aspx*
5. Comment the alert in *CustomerBehavior.js*

Like C#, JavaScript supports block comments using `/* */` and single-line comments using `//`. Remember, you can comment out blocks of text using **CTRL + K, CTRL + C**.

## Part 2: JavaScript IntelliSense in Visual Studio

Using the script manager to include \*.js files also includes Microsoft's AJAX JavaScript library. In order to receive IntelliSense help on the large variety of options available, you must include this line at the top of your script:

```
/// <reference name="MicrosoftAjax.js" />
```

1. Include the following comment at the very top of *Shared.js*:

- `/// <reference name="MicrosoftAjax.js" />`

You can also add your own scripts to IntelliSense. In this case, the attribute is path instead of name:

- `/// <reference path="RelativePathToFile" />`

This will recursively add references that are included in the file you are referencing. Note that the path is relative to the file that you are including the reference in.

- Use the **name** attribute when the .js file is coming from an assembly
- Use the **path** attribute when the .js file is part of the current project



Reference comments must start at the very top of the JS file. If you have multiple references, they must be together without extra carriage returns.

The script references must also be listed in dependence order.

2. Include a reference to *Shared.js* at the **very top** of *CustomerBehavior.js*:

- `/// <reference path="Shared.js" />`

3. Update IntelliSense to reflect the new references using either:

- **CTRL + SHIFT+ J** or **Edit > IntelliSense > Refresh Remote References**



Some developers prefer the **ENTER** key to commit Intellisense text over the default **TAB** key.

To change this:

1. Click **Tools->Options**
2. Select **Text-Editor->JavaScript->Miscellaneous**
3. Under **Statement completion** uncheck the option marked **Only use Tab or Enter to commit**
4. Click **OK**

Now you can use the same keystrokes as C# to commit Intellisense completion.

## Part 3: Using Namespaces

The global namespace is the root object containing all data accessible by JavaScript. Variables defined outside the scope of another structure will be placed within the global namespace by default. This is something we want to avoid.

An alternative is to define a namespace to contain all structures you create. For example:

```
Type.registerNamespace("Epic.Training.Example.Web.Pages");
```

In this case, the only addition to the global namespace is the object Epic. There is no need to check if the namespace is already created or not as `Type.registerNamespace` will only initialize the namespace if necessary.

Namespaces in JavaScript should follow the same standards and conventions as those used by .NET classes.



Note that `Type.registerNamespace` is part of the Microsoft Ajax library.

1. In `Shared.js`, register a namespace called: `Epic.Training.Example.Web.Pages`
  - Note that as you type, you will see the various options that you have via IntelliSense

```
/// <reference name="MicrosoftAjax.js" />
Type.registerNamespace("Epic.Training.Example.Web.Pages");
```

2. Next, Create an empty object called Shared that is part of the namespace above.
  - `Epic.Training.Example.Web.Pages.Shared = {};`
  - This will be a utility library that contains useful functions shared throughout the web application.
3. Add an alert function in Shared:

```
Epic.Training.Example.Web.Pages.Shared = {
    alert: function(message)
    {
        alert("Shared alert: " + message);
    }
};
```



Function names should always be in camel case (first letter is lower case, first letter of consecutive words capitalized: `thisIsCamelCase`).

Namespace, module and class names should be in Pascal case (first character capitalized, first letter of following words capitalized: `ThisIsPascalCase`).

4. Save `Shared.js`
5. Switch to `CustomerBehavior.js` and then update IntelliSense.

6. From within *CustomerBehavior.js*, call the function that you just defined. Notice what IntelliSense displays as you type.

```
Epic.Training.Example.Web.Pages.Shared.alert("CustomerBehavior.js");
```

7. Try running your page and navigate to *CustomerBehavior.js*. Do you see the message?
8. Modify alert so that the full message is stored in a separate variable:
  - Notice that the keyword var is omitted when assigning *completeMessage*:

```
Epic.Training.Example.Web.Pages.Shared = {  
    alert: function(message)  
    {  
        completeMessage = "Shared alert: " + message;  
        alert(completeMessage);  
    }  
};
```

9. Add an alert in *CustomerBehavior.js*, just below the call to *Shared.alert* that outputs the value of *completeMessage*:

```
Epic.Training.Example.Web.Pages.Shared.alert("CustomerBehavior.js");  
alert(completeMessage);
```

10. Run your application. Why is the local variable available outside the function? (HINT: Think about using a variable in Caché without NEWing it)

▪ \_\_\_\_\_

11. Next, modify *Shared.alert* by declaring *completeMessage* using var.

12. Run your application again and navigate to the customer service page. Is *completeMessage* still available outside of the alert function?

▪ \_\_\_\_\_

13. Where do you suppose variables that don't use var before being assigned a value are stored?

▪ \_\_\_\_\_

14. When you are done experimenting, remove `alert (completeMessage)` from `CustomerBehavior.js`.

## Part 4: Naming your functions

In order to prevent functions on the call stack from being anonymous on certain browsers, Epic convention states that all functions should be named. The name comes after the function keyword and follows this pattern:

```
method: function Application$Class$method () { ... }
```

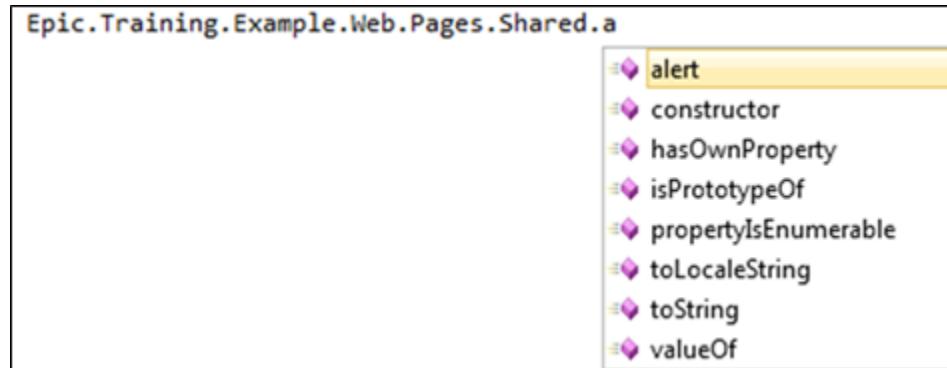
You can also add the owner and/or functional area portions of the namespace, if they are required to avoid a name conflict.

- Modify the alert method by adding a name:

```
alert: function Example$Shared$alert (message) {
    var completeMessage = "Shared alert: " + message;
    alert (completeMessage);
}
```

## Part 5: IntelliSense for Functions

1. Currently, the IntelliSense help on your `Shared.alert` method is quite sparse:



*Default function IntelliSense*

2. To flesh out the IntelliSense help, add comments that summarize what the function does and indicate the data type and purpose of the parameter:

```
alert: function Example$Shared$alert (message) {
    /// <summary> Output a string to the user in a popup.
    /// This is an example of a function in a shared module. </summary>
    /// <param name="message" type="String">String to display</param>
    var completeMessage = "Shared alert: " + message;
    alert (completeMessage);
```

{}

3. Update IntelliSense, then try using it:

```
Epic.Training.Example.Web.Pages.Shared.alert(  
    alert(String message)  
    Output a string to the user in a popup. This is an example of a function in a shared module.  
    message: String to display
```

*IntelliSense enhanced with XML comments*



There are several other kinds of XML comments that you will encounter:

- **returns:** Specify the return type of a function
- **field:** Use inside the prototype of a class to document fields.

The specific attributes for these and more can be found [here](#).

## Part 6: Finding JavaScript Errors

Visual Studio can find some JavaScript errors, but it does so in a way that is very passive. Often, you won't know that you have an error until you encounter it at runtime.

JSHint is a useful tool that is more aggressive about identifying errors. JSHint is included, along with a number of other useful tools, in the Web Essentials Extension for Visual Studio 2013. In 2015, the JSHint features are being included in Epic's custom Hyperspace Visual Studio Extension.



A JavaScript file that contains JSHint errors should not pass PQA.

1. Close all instances of Visual Studio
2. If you are using VisualStudio 2013 Install [this extension to Visual Studio](#)
3. Start Visual Studio
4. You should see a new menu item on the main menu bar called **Web Essentials**
5. Settings should be set up automatically



Ensure that Web Essentials is pointing to the correct configuration file with the following steps. These steps will install Epic's Hyperspace Extension, which includes JSHint for Visual Studio 2015.

1. Close Visual Studio
2. If you are using Visual Studio 2013:
  - a. Browse with Windows Explorer to **M:\fnd\FndGUI\HyperspaceWeb\VS2013**
  - b. Execute **ConfigureVS2013.cmd**
3. If you are using Visual Studio 2015:
  - a. Browse with Windows Explorer to **M:\fnd\FndGUI\HyperspaceWeb\VS2015**
  - b. Execute **ConfigureVS2015.cmd**
4. Start Visual Studio and open your solution

5. Open *BigMouse.sln*.
6. Add the following line of code to *CustomerBehavior.js*

```
alert("test");
```
7. Build your solution to check it for errors.
8. If you are using Visual Studio 2015, follow this menu path:
  - Main Menu > Hyperspace > Check Current Document (CTRL+SHIFT+ALT+J)
9. View the **Error List** tab.
  - If you don't see the error list tab, you can open it from the View menu.
10. You should see an error similar to the following: 'alert' is not defined
11. Normally you should check these keywords to make sure they weren't used before being var'ed. In this case, they were defined either outside *Shared.js*, or in a way that JSHint doesn't recognize.
12. You can disable these warnings by adding the following comment just below the `///<reference>` comments:
  - a. in *CustomerBehavior.js*:
    - `/*global alert*/`
13. Build your solution again. The errors should be gone.
14. Comment out the alert.



From this point on, fix all JSHint errors in your JavaScript code.

Also pay attention to the error window in Visual Studio. This is where JavaScript errors found by the built-in parser will appear.

## Wrap Up

---

1. What happens when you use a variable within a function without defining it using var?

- \_\_\_\_\_
- \_\_\_\_\_

---

2. Why is it poor practice to add numerous variables and functions directly to the global namespace?

- \_\_\_\_\_

---

3. How can we avoid polluting the global namespace?

- \_\_\_\_\_

---

4. What will happen at runtime if you forget to include all the relevant /// comments at the top of your JavaScript File?

- \_\_\_\_\_

---

5. How will a function appear on the call stack if you don't follow Epic's convention of adding Application\$Class\$method to the right of the function keyword?

- \_\_\_\_\_

---

6. What is the syntax to throw an error?

- \_\_\_\_\_

---

7. How can you prevent JSHint from complaining about variables defined outside of the file it is checking?

- \_\_\_\_\_

---



# Subscribing to Client Events

A common use case for JavaScript is subscribing to and handling client events resulting from user actions. For example, if the user clicks a button, we may want to execute a segment of code to handle that button click.

## Scenario: Connect to Click Event of Send

When **Send** is clicked, we want to validate the form and then send the message. To be able to do this we need to connect to the click event.

There are several ways to accomplish this task, such as adding code to *OnClientClick* attribute of the ASP.NET button control, which renders to the onclick attribute in HTML. However, we already stated earlier that this method is not preferred. Instead, we will connect to the click event from within the JavaScript file.

Even if *OnClientClick* is not used, we still need to supply some code to prevent the button click from posting the form back to the server, which causes the entire page to refresh.

The screenshot shows a web application interface titled "E-mail". It contains fields for "Country" (with "US" selected), "Subject", and "Message". There is also a checkbox labeled "Urgent?" and two buttons: "Send" and "Clear". A red arrow points from the "Send" button to a callout box containing the title "OnClientClick" and a bulleted list of three items: "Not preferred", "Better to hook events in JavaScript", and "\"return false;" prevents post back". Below the callout box is the text "Customer.aspx" and the ASPX code for the button:

```
<asp:Button ID="btnSend" runat="server" Text="Send"  
    ToolTip="Send message" CssClass="button"  
    OnClientClick="return false;" />
```

*Disabling Post Back of the Send Button*

## Follow Along: Prevent Post Back

1. Start the web application and then navigate to the customer service page.
2. Verify that clicking **Send** causes the page to refresh.
  - You can verify this by adding an alert back into CustomerBehavior.js
3. In order to prevent post back when **Send** is clicked, add the following attribute to *btnSend*:

- `OnClientClick="return false;"`
4. Refresh the page, then verify that clicking **Send** does not cause the page to refresh.
  5. Be sure to comment out the alert from CustomerBehavior.js

## Follow Along: Connect to button click events

1. Start the web application and navigate to Customer.aspx
2. Open the developer tools using **F12**
3. Type the following using the JavaScript console:

```
function test() { alert("Hello world!"); }

document.getElementById("btnClear").addEventListener("click", test);
```

The `document` object has a method named `getById` that allows you to obtain a reference to an HTML element using its client ID. HTML elements have a method called `addEventListener` that you can use to subscribe to client events.

In the previous example, you created a function named `test`, then you hooked that function up to the click event of the **Clear** button.

1. Verify that clicking **Clear** displays a popup containing the text "Hello world!".
2. Try subscribing to **btnSend** in the same way. Does it work? What is the error?

3. Inspect the source of the **Send** button. What is the client ID?

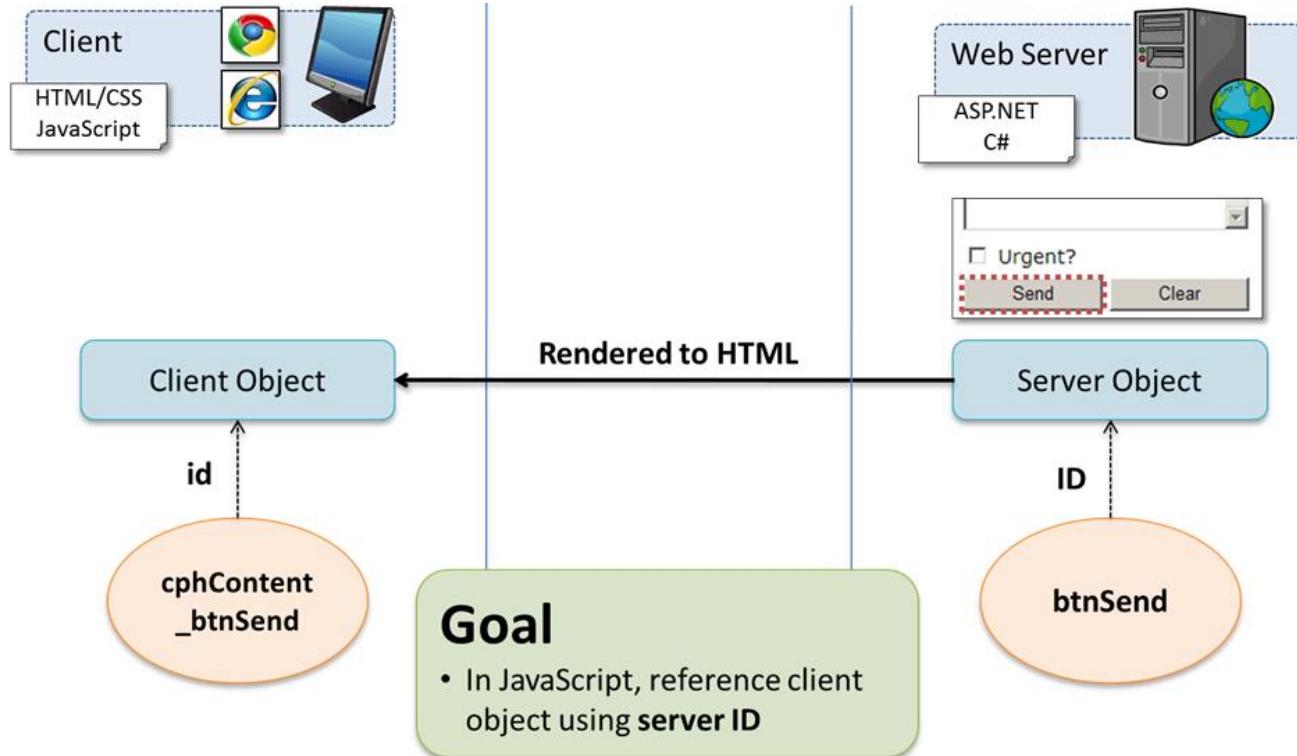


What is true about the client id of controls with `runat="server"`?

## The Client ID of Server Controls

The complication with server controls (i.e., controls with the attribute `runat="server"`), is that the client ID will change depending on where the server control is located on the page. A prefix is

added to the server ID for every other control implementing the interface `INamingContainer` that the server control in question is nested within. This is what you observed in the previous follow-along, and is illustrated in the following diagram:



*The client-object ID is different than the server-object ID*

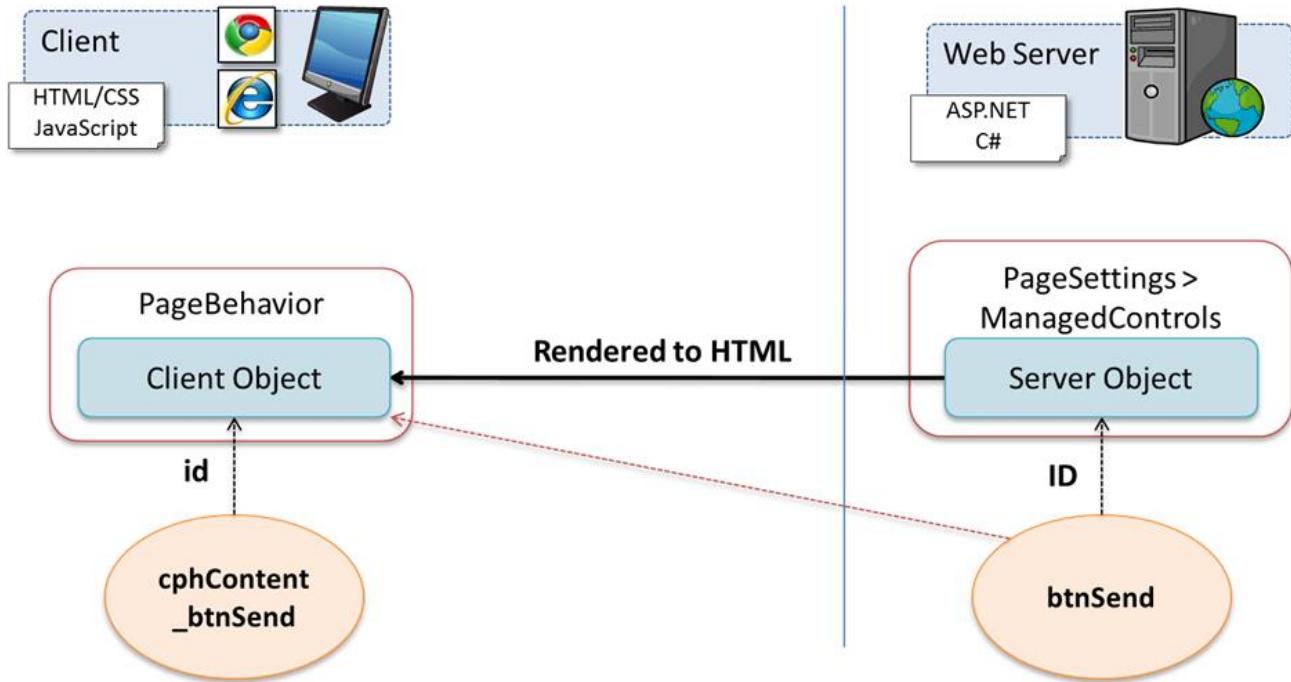
The goal is to be able to reference the client object in JavaScript using the server ID.

You will need to access the HTML element in order to subscribe to client events, and you need the client ID to access the HTML element. Hard coding the client ID assigned to the server control within JavaScript is not maintainable, because it could change if the structure of the page changes.

## Managing Client IDs

In HyperspaceWeb, Epic looks up the client ID assigned by ASP.NET as the page is rendered. The Client ID is used to set up a mapping between the server ID and the resulting HTML element in JavaScript. Since the server ID never changes, the code is more maintainable.

We will use a similar pattern in this class.



*PageSettings (server) and PageBehavior (client) help to manage IDs*

There are two steps to managing the ID of a server control:

1. To indicate that a server control should be mapped using the server ID, register it in the *ManagedControls* section of a special server control named *PageSettings*.
2. The client objects become available in JavaScript via the *PageBehavior* class.

You will get experience doing these steps in the exercise at the end of this section.



The *PageSettings* and *PageBehavior* classes will be provided to you as part of a special training framework written especially for this class. In HyperspaceWeb, you'll use analogous classes named *ViewSettings* and *ViewBehavior*, which are part of the HyperspaceWeb framework.

Other web applications that are not part of HyperspaceWeb can also make use of *PageSettings* and *PageBehavior*.

## Managed Controls Example

The following example illustrates how you can manage the client ID of a input element of type text.

First, add the element that you want to manage to your markup, and register it in the *PageSettings* control:

**Markup:**

```
<input type="text" id="txtName" runat="server" />

<etcw:PageSettings runat="server" ClientClass="YourClass"
    ClientScriptPath="YourClass.js" >
    <ManagedControls>
        <etcw:ControlEntry ControlID="txtName" />
    </ManagedControls>
</etcw:PageSettings>
```

The ControlEntry elements in the aspx file are used to create a (Javascript) map between the server IDs and the client IDs. Specifically, the ControlID attribute of those elements is referenced, and is used to find the matching clientID, at the same time ASP.NET is generating these clientIDs.

Next, create a behavior class for the page that inherits from the PageBehavior class. Within your behavior class, access txtName using this.getEl("txtName"):

#### **YourClass.js:**

```
Namespace.YourClass = function Namespace$YourClass()
{
    Namespace.YourClass.initializeBase(this, [clientId]);
}

Namespace.YourClass.prototype = {
    ...
    set_text(value)
    {
        // get the HTML element with server ID "txtName"
        this.getEl("txtName").value = value;
    }
    ...
};

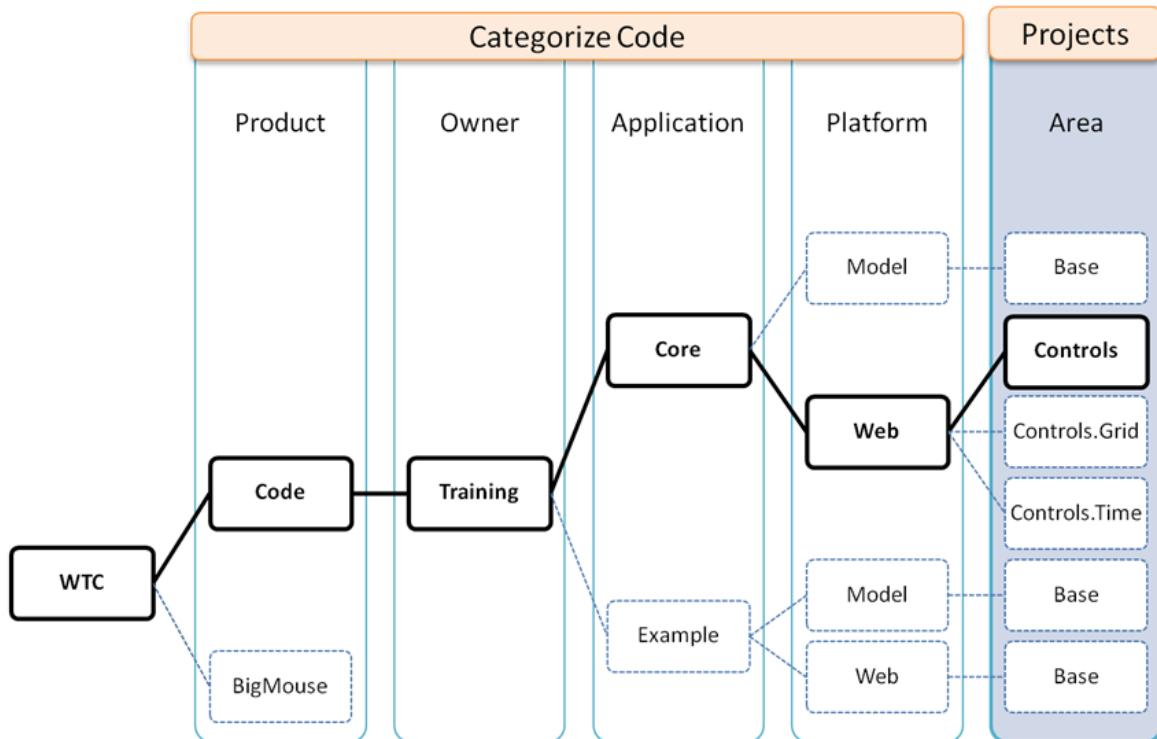
Namespace.YourClass.registerClass("Namespace.YourClass",
    Epic.Training.Core.Controls.Web.PageBehavior);
```

Once your PageBehavior class is constructed, it takes the map made by PageSettings, and uses the clientIDs to retrieve the JS Objects (DOM elements) that have those IDs. Then, it makes a map between the serverIDs (which are known to the developer) and the DOM objects (what the developer wants to be able to manipulate), bypassing the clientIDs (which are not known to the developer). In addition, it provides an API to use this map, this.getEl("serverID"), so that implementation details of the underlying map are encapsulated.

## Exercise: Subscribe to Client Events

In this exercise you will learn how to create a JavaScript *PageBehavior* class that handles client events. The goal is to handle the click event of *btnSend*.

The project containing *PageSettings* and is web-specific, product independent and is used only for training. It is part of Foundations, so the application is Core. Therefore, the folder structure will be as follows:



Folder Structure of the project containing *PageSettings*

### Part 1: Add Training.Core.Controls.Web to your solution

1. Copy the folder **Controls**
  - from: **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\projects\Code**
  - to: **C:\EpicSource\WTC\Code\Training\Core\Web**
2. Open your BigMouse solution and select:
  - **Main Menu > File > Add > Existing Project**
3. Navigate to:
  - **C:\EpicSource\WTC\Code\Training\Core\Web\Controls**
4. Select:
  - **Training.Core.Controls.Web.csproj**

5. Build your solution to create the DLL files
6. Add a reference to *Epic.Training.Core.Controls.Web* assembly in your *Pages* project
  - a. In the solution explorer, right click *References* under *Epic.Training.Example.Web.Pages* and choose *Add Reference*.
  - b. Click the **Browse** button (not the *Browse* tab), then navigate to:  
**C:\EpicSource\WTC\Code\bin**
  - c. Select the file *Epic.Training.Core.Controls.Web.dll*, then click **OK**.
7. Next you need to make sure that the two projects build in the correct order, by indicating that *Training.Example.Web.Pages* depends on *Training.Core.Controls.Web*.
  - a. In the solution explorer, right click *Training.Example.Web.Pages*
  - b. Choose *Project Dependencies*
  - c. Select *Training.Core.Controls.Web* then click **OK**.

You are now able to use *PageSettings* in your solution, which is the ASP.NET/C# component used to map server IDs to client IDs, but you still need to include *PageBehavior*, which is a JavaScript component.

*PageBehavior* could be included with the project you just added (*Training.Core.Controls.Web*) as an embedded web resource. This makes the JavaScript easier to distribute and makes for a more intuitive folder structure. However, it has several key disadvantages:

- Embedded JavaScript can be difficult to debug from Visual Studio because you cannot set dynamic breakpoints within the source file.
- The name of the JavaScript file is changed to a long, non-intuitive string when it is extracted from the assembly. This can make the script file difficult to locate and debug from the browser's developer tools.
- Embedding text within an assembly prevents it from being minimized, so it takes up extra space.
- Properly embedding a JavaScript file into an assembly requires several non-intuitive steps for the programmer, which can be confusing at first

For these reasons, HyperspaceWeb no longer embeds JavaScript files in assemblies. Instead, assemblies are kept in separate kind of project called an Assets project. Assets projects that are nested within the folder of the web application so they do not need to be pulled from other assemblies.



If you're curious how a JavaScript file can be embedded as an assembly, see:

<https://msdn.microsoft.com/en-us/library/bb398930.aspx>

In the following steps, you will add the Assets project containing PageBehavior.js to your solution.

8. Open the File Explorer to the root folder of the Pages project:
  - Solution Explorer > Training.Example.Web.Pages > right - click > Open Folder in File Explorer
9. From the file Explorer, create a new folder called **Assets**
10. Within Assets, create a new folder called **Training**
11. Copy the folder **Core**
  - from: I:\Internal\Advanced\Web Tech Camp\Big-Mouse\projects\Assets
  - to: C:\EpicSource\WTC\BigMouse\Assets\Training
12. Open your BigMouse solution and select:
  - Main Menu > File > Add > Existing Project
13. Navigate to:
  - C:\EpicSource\WTC\BigMouse\Assets\Training\Core
14. Select:
  - **Training.Core.Controls.Web.Assets.csproj**

The Scripts folder of Trianing.Core.Controls.Web.Assets contains three files:

1. Clock.js
2. CollectionGrid.js
3. PageBehavior.js

The first two will be used in a later exercise. The third will we will use in this exercise.

Note that you do not need to add a reference to or set a build dependency on your assets project, because the assets project doesn't contain any C# code that we need to reference.

## Part 2: Add a PageSettings Control to CustomerService.aspx

1. At the top of *Customer.aspx*, just below the `<%@ Page ... %>` line and just above the *asp:Content* control, register the assembly you just referenced:

```
<%@ Register Namespace="Epic.Training.Core.Controls.Web"  
    Assembly="Epic.Training.Core.Controls.Web"  
    TagPrefix="etcw" %>
```

2. At the bottom of the page, just before the end of the *asp:Content* control and just after the *ScriptManagerProxy*, add a new *PageSettings* control, and two control entries:

```
...
```

```
</asp:ScriptManagerProxy>

<!-- Customer.aspx Page Settings -->
<etcw:PageSettings runat="server">
    <ManagedControls>
        <etcw:ControlEntry ControlID="btnClear" />
        <etcw:ControlEntry ControlID="btnSend" />
    </ManagedControls>
</etcw:PageSettings>
</asp:Content>
```

3. Be sure that you included the comment in the previous step. This will help you locate the rendered version of page settings so you can understand what it is doing.
  4. Run your page, navigate to *Customer.aspx*, then view the source of the page.
  5. Search for your comment in the source. Notice what *PageSettings* rendered to.
  6. Why are the control entries for *btnClear* and *btnSend* rendered differently?
- 
- 

## Part 3: Behavior of the Customer Service Page

In this section, you will create a new JavaScript class to manage the behavior of the customer service page. This class will inherit from *PageBehavior* and receive controls listed in the managed controls from *Customer.aspx*, namely *btnSend* and *btnClear*.

1. Code snippets are pre-defined code templates that help automate common programming tasks. You should already have the snippets required for this class.
2. Open *CustomerBehavior.js*
3. Delete everything in this file so all that remains is the reference to *Shared.js* on the first line:
  - /// <reference path="Shared.js" />
4. In *CustomerBehavior.js*, use the *assetpagebehavior* snippet to auto generate the *PageBehavior* class:
  - a. On the line below the reference to *Shared.js*, type *assetpagebehavior*
  - b. Press **TAB** twice to insert the snippet
  - c. The year will be set to a default value. Update it as necessary, then press **TAB**.
  - d. Replace *mynamespace* with *Epic.Training.Example.Web.Pages*
  - e. Press **TAB**.
  - f. Replace *MyControl* with *CustomerBehavior*, then press **TAB**.
  - g. Replace *Epic\$* with *Example* then press **ENTER**
5. Take a moment to look through the class that was just created:

- There is an additional IntelliSense reference:

```
//> <reference path="~/Assets/Training/Core/Scripts/PageBehavior.js" />
```

- Some default JSHint global declarations were included as part of the header comments:

```
/*global alert, $addHandler */
```

- There is one region included in the prototype definition. It includes methods overridden from *PageBehavior* where you can tie into certain events:

```
##region overridden methods from CustomerBehavior
onDataPreload: function Example$CustomerBehavior$onDataPreload () {
    //;<summary>Called by the constructor before "onLoad".</summary>
},
...
##endregion
```



Recall that Foundations uses a specific pattern to load data asynchronously from the server. The methods *onDataPreload*, *onLoad* and *onLoaded* are part of this pattern, which will be covered in more detail later during this lesson.

- The constructor includes a call to a method named *initializeBase*:

`initializeBase(  
instance, baseArgs)`

Microsoft Ajax library equivalent of the following base call in C#:

`Constructor(args) : base(baseArgs){ ... }`

- instance**: object being constructed
- baseArgs**: Array of arguments for the base constructor (can be null)

- At the end of the file, there is a call to *registerClass*:

`registerClass(  
typeName,  
baseType)`

Microsoft Ajax equivalent of the following C# class definition:

`class TypeName : BaseType{ ... }`

- typeName**: String representation of the class being registered (including its namespace)
- baseType**: Reference to the base class. Can be null if not inheriting.

6. In *Customer.aspx*, add an attribute to the page settings that specifies *CustomerBehavior* as the client class. **This will cause its constructor to be called automatically when the page is rendered.**

```
<!-- Customer.aspx Page Settings -->
<etcw:PageSettings runat="server"
    ClientClass="Epic.Training.Example.Web.Pages.CustomerBehavior">
```

7. Add the attribute *ClientScriptPath* that points to *CustomerBehavior.js*. **This ensures that the script is loaded with the page.**
8. Remove the script manager proxy. It is no longer needed to load *CustomerBehavior.js*:

The diagram shows a comparison of two snippets of ASPX code. On the left, a red arrow labeled 'Remove' points to the original code which includes a `<asp:ScriptManagerProxy>` block. On the right, a green arrow labeled 'Add' points to the modified code where the `<asp:ScriptManagerProxy>` block has been removed, and a new `<etcw:PageSettings>` block has been added with the `ClientScriptPath` attribute set to `~/CustomerBehavior.js`.

```

    ...
    <asp:ScriptManagerProxy ID="smp" runat="server">
        <Scripts>
            <asp:ScriptReference Path="~/CustomerBehavior.js" />
        </Scripts>
    </asp:ScriptManagerProxy>

    <!-- Customer.aspx Page Settings -->
    <etcw:PageSettings runat="server"
        ClientClass="Epic.Training.Example.Web.Pages.CustomerBehavior"
        ClientScriptPath="~/CustomerBehavior.js">
        <ManagedControls>
            <etcw:ControlEntry ControlID="btnClear" />
            <etcw:ControlEntry ControlID="btnSend" />
        </ManagedControls>
    </etcw:PageSettings>
</asp:Content>

```

*Additional changes to Customer.aspx*



Be sure to remove the *ScriptManagerProxy*. If you do not, the browser will attempt to register *CustomerBehavior* before its parent class *PageBehavior* has been registered, which will result in an error.

9. Add an alert to the constructor of *CustomerBehavior* to verify that it is executed.
10. Save and run your web application, then navigate to *Customer.aspx*. Did you see the alert?
11. Once you see that *CustomerBehavior* is loading, comment out the alert.

## Part 4: Connect to the Click Event of btnSend

1. Define a new region called *event handlers* above the region marked *overridden methods from PageBehavior*.

2. Define a new event handler called `_btnSendClick` using the `instmethod` snippet. Within this method, alert the user that the button was clicked.

```
//#region event handlers
    _btnSendClick: function Example$CustomerBehavior$_btnSendClick ()
    {
        ///<summary>Click event handler for btnSend</summary>
        alert("btnSendClick called...");  

    },
//#endregion
```



You may find that the `instmethod` snippet doesn't show up in IntelliSense. This is a known bug in Visual Studio. Try using it anyway (type "instmethod" then tab twice).

3. By convention, what is the access level of this method?

■

---

The Microsoft AJAX library contains two functions that are useful for subscribing to element events: `$addHandler` and `Function.createDelegate`.

\$addHandler( element, eventName, handler)	<p>Specifies an event handler that will listen to the specified event of the given element.</p> <ul style="list-style-type: none"><li>• <b>element:</b> The element raising the event</li><li>• <b>eventName:</b> JavaScript element events. For example, use "click".</li><li>• <b>handler:</b> The function to call when the event is raised. Typically, the output of <code>Function.CreateDelegate</code> is used for this argument.</li></ul> <p>This method first validates that all the input arguments are correct, and then calls the standard JavaScript method <code>addEventListener</code> from the provided element.</p>
Function.createDelegate (instance, method)	<p>Create a delegate that will be called when an event is raised. Typically the result of this function is used as the handler argument of <code>\$addHandler</code>.</p> <ul style="list-style-type: none"><li>• <b>instance:</b> The object instance that will be referenced by the <code>this</code> keyword.</li></ul>

keyword.

- **method:** The method of the instance to call.

4. In *onLoaded*, add a handler for the click event of *btnSend*:

```
$addHandler(this.getEl("btnSend"),"click",
    Function.createDelegate(this,this.__btnSendClick));
```

In the above code:

- The *getEl* function stands for get element. Use it to access any element that you registered in the *PageSettings* using the server ID.
  - The code is placed in *onLoaded* because this is the code that can run while we are waiting on the asynchronous request for data to return. You haven't learned how to request the data yet. When you do, that request will happen in the *onLoad* method.
5. Save and run your website, then navigate to *Customer.aspx*. When you click **Send**, do you see the message?

## Part 5: Understanding Function.CreateDelegate

In this part you will run an experiment to see what happens when you accidentally omit *Function.createDelegate*, which is a common mistake.

1. Add another instance method (outside the event handlers region) called *toString*.
2. Inside *toString*, return name of the class:

```
toString: function Example$CustomerBehavior$toString() {
    ///<summary>A string representation of CustomerBehavior</summary>
    /// <returns type="String" />
    return "Epic.Training.Example.Web.Pages.CustomerBehavior";
},
```

3. Edit *\_\_btnSendClick* to indicate where the method is being called from using *this.toString*:

```
alert("__btnSendClick called from " + this.toString());
```

4. Run your application again. You should see that the click event is being handled by *CustomerBehavior*.
5. In *onLoaded*, replace *Function.createDelegate(this, this.\_\_btnSendClick)* with just *this.\_\_btnSendClick*:

```
$addHandler(this.getEl("btnSend"), "click", this.__btnSendClick);
```

6. Run your site and click **Send**.
7. Based on the experiment, does *this* still refer to the instance of *CustomerBehavior* if *Function.createDelegate* is omitted?

■

- 
8. Put *Function.createDelegate* back:

```
$addHandler(this.getEl("btnSend"), "click",  
    Function.createDelegate(this, this.__btnSendClick));
```

9. Answer the wrap-up questions.

## Part 6: Using the Event Object

Recall that more arguments can be passed to a function than are listed as parameters. Set a breakpoint in a function and then examine the *arguments* variable for details on what was actually passed at runtime. For example, an event object is passed to an event handler function as the first argument. The event handler object references the control that raised the event from its *target* property.

1. Add the parameter *event* to *\_\_btnSendClick*.
2. Include *event.target.id* in the alert message.

```
__btnSendClick: function Example$CustomerBehavior$__btnSendClick(event)  
{  
    alert("Send clicked: " + event.target.id);  
},
```

3. Run your web application, navigate to *Customer.aspx* and click **Send** to test the new code.

## Part 7: Foundations Shortcuts

For convenience and efficiency, Foundations provides shortcut functions to many commonly used methods from the Microsoft Ajax library. These shortcuts are prefixed with \$\$, and exist in the global namespace.



### Do not create your own shortcut functions in the global namespace

If there is a shortcut that you think would be helpful, communicate your need with Foundations rather than polluting the global namespace yourself.

1. The foundations shortcut to *Function.createDelegate* is `$$fcd`. Add the following code to *Shared.js*, just above the definition of *Epic.Training.Example.Web.Pages.Shared*:

```
///#region Foundations shortcuts
var $$fcd = Function.createDelegate;
//#endregion
```



Note that you will not need to do this yourself once you begin working within the HyperspaceWeb framework. `$$fcd` will already be defined for you.

2. Use the shortcut when connecting to the click event of *btnSend*:

```
onLoaded: function Example$CustomerBehavior$onLoaded()
{
    var btnSendDelegate = $$fcd(this, this.__btnSendClick);
    $addHandler(this.getEl("btnSend"), "click", btnSendDelegate);
},
```

3. Verify that **Send** still functions as expected.

## Wrap Up

---

1. Why were *btnClear* and *btnSend* rendered differently?
  -
2. What happens if you forget to use *Function.createDelegate* with *\$addHandler*?
  -
3. Why is it better to use *\$addHandler()* to subscribe to click event of *btnSend* rather than the *OnClientClick* attribute in the ASP.NET (or the onclick attribute in HTML)?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

4. Why is it better to use `$addHandler()` to subscribe to events rather than the standard JavaScript `addEventListener`?

- \_\_\_\_\_

5. How can you access the element that raised the event?

- \_\_\_\_\_
- \_\_\_\_\_

6. What is the name prefix for Foundations-written shortcut functions?

- \_\_\_\_\_

7. What is the name of the Foundations shortcut to `Function.createDelegate`?

- \_\_\_\_\_

### If you have time

1. Experiment with the JavaScript minimizer used by HyperspaceWeb. You can find the minimizer with instructions here:
  - **I:\Internal\Advanced\Web Tech Camp\Projects\BuildScripts\support**
2. If you are curious about how `PageSettings` and `PageBehavior` function, take some time to look through them.
  - Feel free to ask the instructor during lab if you are curious how they work.

# Validating Forms

Since you have access to the controls and their values on the client, one major use of JavaScript is form validation.

**E-mail**

Name:

Email:

Country:

Subject:

Message:

Urgent?

*Validate Email Form when Send is Clicked*



## Where to Validate

Should user input be validated on the client, or on the server?

- 
- 
- 

## Exercise: Validate Email Fields

In this exercise you will first validate the email fields before allowing the user to send an email.

Next, you will learn a bit about JavaScript debugging in the various browsers.

## Part 1: Validate the Form

1. Create two more fields on the email form, one for the user's name, *txtName* and one for the user's email address, *txtEmail*. Make them server controls with `runat="server"`, using standard HTML controls.
2. Make the controls available in JavaScript using page settings.
  - In other words, add a `ControlEntry` for each element that requires validation.
3. When *btnSend* is clicked look up each form element using `this.getEl` and store it into a local variable:

```
var txtName = this.getEl("txtName");
var txtEmail = this.getEl("txtEmail");
var selCountry = this.getEl("selCountry");
var txtSubject = this.getEl("txtSubject");
var areaMessage = this.getEl("areaMessage");
var chkUrgent = this.getEl("chkUrgent");
```

4. Next, pull the value stored in *txtName* and store it into a local variable using the `value` property:

```
var name = txtName.value;
```

Notice that as you type `txtName.value`, the JavaScript IntelliSense does not provide an option for the `value`. Even though you know this is an input element, Visual Studio has no way of knowing that because the type returned by `getEl` is an [HTMLElement](#), not an [HTMLInputElement](#).

1. To verify this problem, go to the definition of `this.getEl` by placing your cursor on one of the calls in `CustomerBehavior.js` and pressing **F12**, then examine the return type listed in the XML comments.
2. Scroll to the bottom of `PageBehavior.js` and examine the functions `$$asInputElement`, `$$asSelectEl` and `$$asTextAreaEl`. What are they used for?
  -
3. Update your code in `CustomerBehavior` so that IntelliSense knows the type of each form element:

```
var txtName = $$asInputElement(this.getEl("txtName"));
var txtEmail = $$asInputElement(this.getEl("txtEmail"));
var selCountry = $$asSelectEl(this.getEl("selCountry"));
var txtSubject = $$asInputElement(this.getEl("txtSubject"));
```

```
var areaMessage = $$asTextAreaEl(this.getEl("areaMessage"));
var chkUrgent = $$asInputEl(this.getEl("chkUrgent"));
```

4. Obtain the value of each element and store it in a local variable for validation.
  - You should get IntelliSense indicating that value exists.
5. Trim the result using the `trim` method so whitespace is not considered a valid value.
6. The resulting code should look similar to the following:

```
var name = txtName.value.trim();
var email = txtEmail.value.trim();
var country = selCountry.value.trim();
var subject = txtSubject.value.trim();
var message = areaMessage.value.trim();
```

7. Validate that each value is not null or empty. If so, use alert to inform the user that the missing value should be provided.
8. In the case of email, also validate that the given value is a valid email address.

You can validate the email address using a regular expression. Add the method from the following file to `Shared.js`, within the Shared object definition:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\scripts\isEmail.txt

```
Epic.Training.Example.Web.Pages.Shared = {
  alert: function Example$Shared$alert(message)
  {
    /// <summary> Output a string to the user in a popup.
    /// This is an example of a function in a shared module. </summary>
    /// <param name="message" type="String">String to display</param>
    var completeMessage = "Shared alert: " + message;
    alert(completeMessage);
  },
  isEmail: function Example$Shared$isEmail(str)
  {
    /// <summary> Verifies that the entered string is an email address </summary>
    /// <param name="str" type="String">string to check</param>
    /// <returns type="Boolean">True if the string is a valid email address</returns>
    var email = /^[\w\.-_]+(\.[\w\.-_]+)*@[\\w\.-_]+\.[\\w\.-_]+(\.[\\w\.-_]+)*[\s]*$/;
    return str.search(email) !== -1;
  }
};
```



Make sure that you place the `isEmail` method in the right place. It should go **inside** the definition of Shared, not below it.

9. To make calling `Shared` easier, you can add a reference to it in the prototype of `CustomerBehavior`:

```
//#region fields  
__shared : Epic.Training.Example.Web.Pages.Shared,  
//#endregion
```

Now you can call methods from shared like this:

```
this.__shared.isEmail(stringToTest);
```

10. To get IntelliSense help on fields, the XML documentation actually goes in the body of the constructor. Add comments for the new `__shared` field:

```
Epic.Training.Example.Web.Pages.CustomerBehavior = function  
Example$CustomerBehavior(clientId)  
{  
    ///<field name="__shared" type="Epic.Training.Example.Web.Pages.Shared">  
    ///Shortcut to the shared library</field>  
    Epic.Training.Example.Web.Pages.CustomerBehavior.initializeBase(this,  
    [clientId]);  
}
```

11. Make sure that clicking the label next to any of the controls results in the corresponding control getting focus.
  - You'll notice that this requires the use of an `asp:Label` rather than the `label` element for controls with `runat="server"`.
12. If there are no errors on the form, alert the user that the email was sent and clear the form.



A quick way to clear the form is to get a reference to the form from one of the buttons, and then call the reset method. This is equivalent to clicking a button of type reset. For example:

```
$$asInputElement(this.getEl("btnClear")).form.reset();
```

Be aware that the entire page has only one form, so this will reset all controls on the page. If this was not the behavior that you wanted, you would need to deal with each field individually.

13. The final validation code will look something like this:

```
// Validate each value
if(!name)
{
    alert("Enter your name");
    return;
}

if(!this.__shared.isEmail(email))
{
    alert("Enter a valid e-mail address");
    return;
}

if(!country)
{
    alert("Select a country");
    return;
}

if(!subject)
{
    alert("Enter a subject");
    return;
}

if(!message)
{
    alert("Enter a message");
    return;
}

alert("e-mail sent");
$$asInputElement(this.getEl("btnClear")).form.reset();
```

## Part 2: Save the Form Controls in Fields

Notice that every time **Send** is clicked, the form controls are first pulled using `getEl`. We can make the page more efficient by performing this step when the page first loads and then saving the references into class-level fields.

1. Create a field in `CustomerBehavior` named `__txtName`:

```
//#region fields  
__shared: Epic.Training.Example.Web.Pages.Shared,  
  
__txtName: null,  
  
//#endregion
```

2. Ensure you get proper IntelliSense on `__txtName` by adding a field XML comment to the constructor:

```
Epic.Training.Example.Web.Pages.CustomerBehavior = function  
Example$CustomerBehavior(clientId)  
{  
    ///<summary>Behavior for the customer service page</summary>  
    ///<field name="__shared"  
type="Epic.Training.Example.Web.Pages.Shared">  
    /// Shortcut to the shared library</field>  
    ///<field name="__txtName" type="HTMLInputElement">Name field from the email  
form</field>  
    Epic.Training.Example.Web.Pages.CustomerBehavior.initializeBase(this,  
[clientId]);  
};
```

3. In `onLoaded`, initialize `__txtName` using `getEl`:

```
this.__txtName = this.getEl("txtName");
```

Notice that `$$asInputEl` isn't needed at this point. This is because the field comment will let IntelliSense know what the type of the object is in other parts of `CustomerBehavior`.

4. In `__btnSendClick`, remove the declaration of `txtName`.
5. Everywhere `txtName` was used, replace it with `this.__txtName`
6. Repeat the previous steps for the remaining form elements
7. The final code should appear as follows:

#### Constructor:

```
Epic.Training.Example.Web.Pages.CustomerBehavior = function
```

```
Example$CustomerBehavior(clientId)
{
    /// <summary>Behavior for the customer service page</summary>
    ///<field name="__shared"
type="Epic.Training.Example.Web.Pages.Shared">
    ///Shortcut to the shared library</field>
    ///<field name="__txtName" type="HTMLInputElement">Name field from the
email form</field>
    ///<field name="__txtEmail" type="HTMLInputElement">Email address field
from the email form</field>
    ///<field name="__selCountry" type="HTMLSelectElement">Country field
from the email form</field>
    ///<field name="__txtSubject" type="HTMLInputElement">Subject field
from the email form</field>
    ///<field name="__areaMessage" type="HTMLTextAreaElement">Message field
from the email form</field>
    ///<field name="__chkUrgent" type="HTMLInputElement">Urgent field from
the email form</field>
    Epic.Training.Example.Web.Pages.CustomerBehavior.initializeBase(this,
[clientId]);
};
```

### Field declarations:

```
Epic.Training.Example.Web.Pages.CustomerBehavior.prototype = {

    //#region fields
    __shared: Epic.Training.Example.Web.Pages.Shared,
    __txtName: null,
    __txtEmail: null,
    __selCountry: null,
    __txtSubject: null,
    __areaMessage: null,
    __chkUrgent: null,
    //#endregion

    ...
}
```

### Click event handler:

```
__btnSendClick: function Example$CustomerBehavior$__btnSendClick(event)
{
    ///<summary>Click event handler for btnSend</summary>
    var name = this.__txtName.value.trim();
    var email = this.__txtEmail.value.trim();
```

```

var country = this.__selCountry.value.trim();
var subject = this.__txtSubject.value.trim();
var message = this.__areaMessage.value.trim();

// Validate each value
if(!name)
{
    alert("Enter your name");
    return;
}

if(!this.__shared.isEmail(email))
{
    alert("Enter a valid e-mail address");
    return;
}

if(!country)
{
    alert("Select a country");
    return;
}

if(!subject)
{
    alert("Enter a subject");
    return;
}

if(!message)
{
    alert("Enter a message");
    return;
}

alert("e-mail sent");
$$asInputEl(this.getEl("btnClear")).form.reset();
},

```

### The onLoaded method:

```

onLoaded: function Example$CustomerBehavior$onLoaded()
{
    /// <summary>Called by the constructor after "onLoad".</summary>
    var btnSendDelegate = $$fcd(this, this.__btnSendClick);
    $addHandler(this.getEl("btnSend"), "click", btnSendDelegate);
    // Cache references to email form elements
    this.__txtName = this.getEl("txtName");
}

```

```

this.__txtEmail = this.getEl("txtEmail");
this.__selCountry = this.getEl("selCountry");
this.__txtSubject = this.getEl("txtSubject");
this.__areaMessage = this.getEl("areaMessage");
this.__chkUrgent = this.getEl("chkUrgent");
}

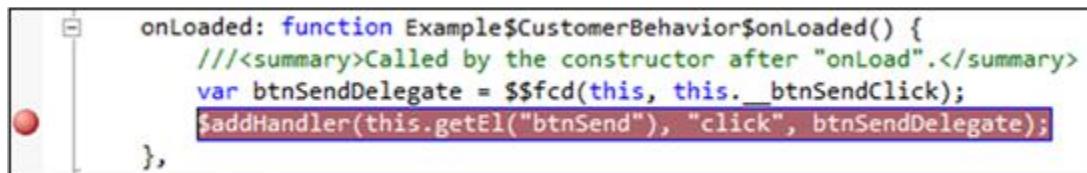
```

## Part 3: Debugging

---

Now that you have several JavaScript files to experiment with, this is a good time to practice debugging techniques.

1. Set your default browser to IE.
2. Set a dynamic breakpoint in *CustomerBehavior.js* at the `$addHandler` call in *onLoaded*.
  - This is done by clicking in the margin next to the line on which you want to set a breakpoint.



*Setting a Breakpoint in Visual Studio*

3. In the constructor in *PageBehavior.js*, set a breakpoint on the call of *onLoaded*. The call in *PageBehavior.js* is what executes the method in *CustomerBehavior.js*.
4. Run your web application and navigate to the customer service page.
5. Continue stepping through the code until you reach *Customer.aspx*. Feel free to step over (or out of) framework code, such as `$addHandler`.
6. Run the web application in Chrome. Do the breakpoints work as expected?

---

## If you have time

---

Figure out how to set the same two breakpoints in Chrome using its built-in developer tools.

## Wrap Up

---

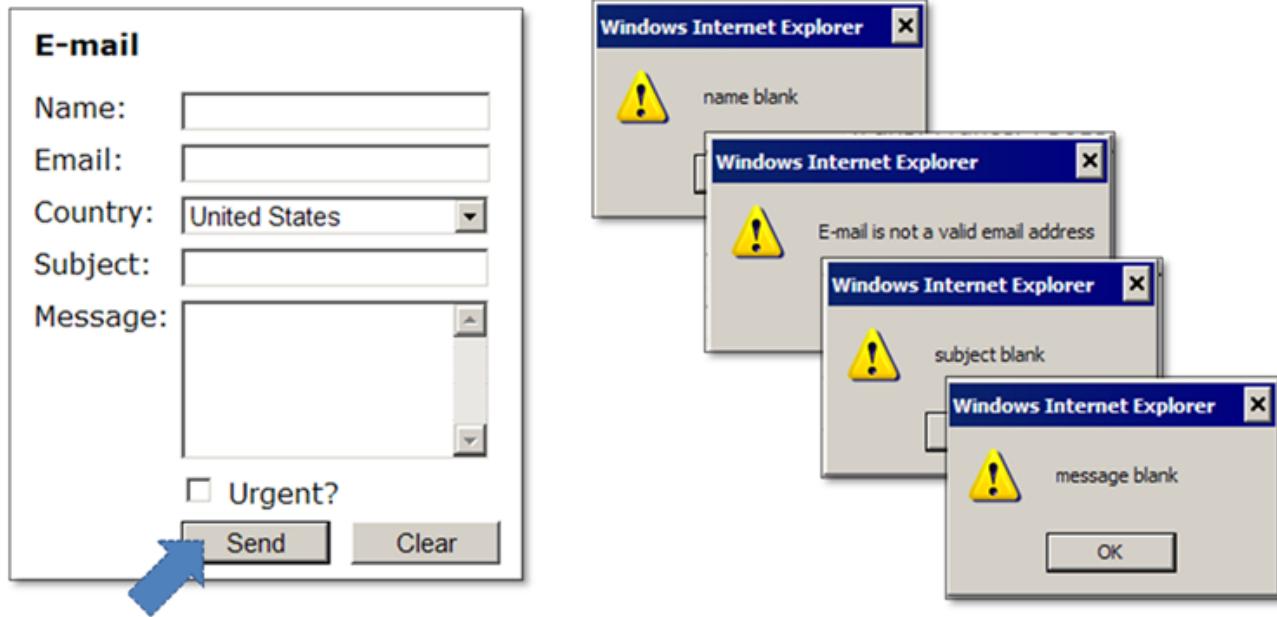
1. Why do server controls (`runat="server"`) require an `asp:Label` to receive focus when the label is clicked, while client controls may use an HTML label element?

- \_\_\_\_\_
  - \_\_\_\_\_
2. Can you integrate the Visual Studio debugger with Chrome?
- \_\_\_\_\_

# Changing Content and Layout

## Behavior after First Exercise

Currently, if there are multiple errors in the email form, only the first error will be displayed. Additionally, the errors are displayed in alert boxes, which is somewhat disruptive to the end user and not consistent across browsers.



*Alerts are not user friendly*

## Behavior after Second Exercise

A better user interface would display all errors that are currently on the form as part of the page. To make it more accessible, the errors could link back to the field containing the error:

**E-mail**

Name:

Email:

Country:

Subject:

Message:

Urgent?



Please fix the following errors before submitting:

- Enter a name
- Enter an email address
- Enter a subject
- Enter a message

*Illustration of the desired behavior*

 HyperspaceWeb will automatically validate entry fields individually (like Chrontrols in Hyperspace).

## Document Object Model (DOM)

The Document Object Model (DOM) is the JavaScript-object representation of the HTML tree. The DOM includes the combination of the content rendered from ASP.NET, as well as the current layout from CSS. Both the content and layout can be manipulated through the DOM in JavaScript. This is often referred to as Dynamic HTML (DHTML).

In the customer service page, you will use a hidden div element as a placeholder to insert errors from the email form. When errors occur, they will be added to this div as a list of hyperlinks, and then the div will be displayed. The following diagram illustrates how the div element will appear in the DOM as displayed by the IE developer tools:

<p><b>DIV Without Errors</b></p>  <pre>&lt;div class="error hidden" id="divError"/&gt;</pre>	<p><b>DIV With Errors</b></p> <p>Please fix the following errors before submitting:</p> <ul style="list-style-type: none"> <li>Enter a name</li> <li>Enter an email address</li> <li>Enter a subject</li> <li>Enter a message</li> </ul> <div style="border: 1px solid red; padding: 5px; margin-top: 10px;"> <span>CSS class <i>hidden</i> removed</span> </div> <pre> &lt;div class="error" id="divError"&gt;   &lt;strong&gt;     Text - Please fix the following errors before submitting:   &lt;/strong&gt;   &lt;ul&gt;     &lt;li&gt;&lt;a href="javascript:;" _events="[object Object]"&gt;Text - Enter a name&lt;/a&gt;&lt;/li&gt;     &lt;li&gt;&lt;a href="javascript:;" _events="[object Object]"&gt;Text - Enter an email address&lt;/a&gt;&lt;/li&gt;     &lt;li&gt;&lt;a href="javascript:;" _events="[object Object]"&gt;Text - Enter a subject&lt;/a&gt;&lt;/li&gt;     &lt;li&gt;&lt;a href="javascript:;" _events="[object Object]"&gt;Text - Enter a message&lt;/a&gt;&lt;/li&gt;   &lt;/ul&gt; </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The div with and without errors



Note that the closed notation of the div element above is only how IE is displaying the element. Remember that you should never use closed notation with HTML elements that could contain content, as it will break your page.

#### Actual markup:

- <div id="divError" class="error hidden"></div>

## Exercise: Display Error Messages

In this exercise you will enhance the email form by making it more user friendly. The goal is to display validation errors as part of the page, below the email form. The errors will be links back to the control containing the error.

In order to add the errors as specified above, you will need to be able to alter the content and layout of the page. The following functions allow you to do this:

Element properties and methods	<p>HTML elements have a variety of properties and methods that allow you to alter content on the page. Some examples are:</p> <ul style="list-style-type: none"> <li>• <b>childNodes</b> - Array of Elements contained in an element</li> <li>• <b>firstChild</b> - First element from childNodes</li> <li>• <b>appendChild(element)</b> - Adds an element to the end of childNodes</li> <li>• <b>removeChild(element)</b> - Removes specified element from</li> </ul>
--------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>childNodes</p> <ul style="list-style-type: none"> <li>• <b>setAttribute(attributeName,attributeValue)</b> - Set the value of an attribute of this element to the specified value.</li> </ul>
Document methods	<p>HTML elements have a variety of properties and methods that allow you to alter content on the page. Some examples are:</p> <ul style="list-style-type: none"> <li>• <b>document.createElement</b> - Returns a newly created but unattached html element. The argument passed is a string with the name of the element, such as "a", "div", "span", or "input".</li> <li>• <b>document.createTextNode</b> - Creates a node that is only used to display text. <ul style="list-style-type: none"> <li>▪ In general, it can be added to childNodes of elements that contain content, such as div, to insert text that would normally be between &lt;div&gt; and &lt;/div&gt;.</li> <li>▪ Text added this way is made HTML safe, meaning that &lt; is replaced by &amp;lt;, &gt; by &amp;gt;, and so on.</li> </ul> </li> </ul>
Microsoft AJAX methods	<p>Useful methods included with Microsoft AJAX library:</p> <ul style="list-style-type: none"> <li>• <b>Sys.UI.DomElement.addCssClass(element,className)</b> - Appends className to the list of classes of the given element. If the element already has the specified class, it does nothing.</li> <li>• <b>Sys.UI.DomElement.removeCssClass(element,className)</b> - Removes className from the list of classes of the given element. If the element doesn't have the specified class, it does nothing.</li> <li>• <b>Function.createCallback(method,context)</b> - Augments the given method by appending context to the end of its list of arguments.</li> </ul>

Most of these features are wrapped in foundations shortcut functions within HyperspaceWeb. Again, the purpose of Foundations shortcuts is to increase efficiency, convenience and browser compatibility. We will add some similar shortcuts during this exercise as they become relevant.

## Part 1: Prepare your Solution

---

Since you have not seen the methods from the previous section in action, the method to modify the DOM will be provided to you this time. Your job is to add it to your solution, understand how it works and use it correctly.

1. Add an empty div to *Customer.aspx*, just below the email layout table.

- <div id="divError" class="error hidden"></div>

- This div will contain error messages that arise when the user attempts to submit the email form.
2. Expose this div to *CustomerBehavior* by adding a *ControlEntry* for it in *PageSettings > ManagedControls*:
    - <etcw:ControlEntry ControlID="divError" />
  3. Add the following CSS to *ExampleMaster.css*:

```
.hidden
{
    display: none;
}
.error
{
    background-color: #FFEEEE;
    color: Red;
    border: 2px solid Red;
    padding: 5px;
}
li
{
    list-style-position: inside;
}
```

4. Copy the JavaScript from the following file and paste it into *Shared.js*, within the Shared object definition:
  - **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\scripts\displayFormErrors.txt**



Make sure you put *displayFormErrors* in the right place. It should go inside the definition of *Shared*. Be sure that you add a comma so that the object definition is syntactically correct.

5. To avoid JSHint errors, you will need to declare a global:
  - `document`
6. Examine the method *displayFormErrors*. Look for the methods described in the previous section of this exercise and see how they are being used.
7. What function is called when the generated hyperlinks are clicked?
  - \_\_\_\_\_
8. What mechanism is used to pass the control to this function?

- 
- 
9. When you feel that you understand how this function should be used, replace the alert calls with a call to *displayFormErrors*. Be sure that the errors appear when appropriate and that the errors are not displayed when the form submits without errors.

- To add an error to the details array, you can do something similar to the following:

```
var name = this.__txtName.value.trim();
...
var details=[];

if (!name)
{
    details.push(
        {message: "Name is required", control: this.__txtName }
    );
}
...
...
```

10. Attempt to modify all your validation to use the new method. If you get stuck, refer to the following example:

```
__btnSendClick: function Example$CustomerBehavior$__btnSendClick(event)
{
    //;<summary>Click event handler for btnSend</summary>
    var name = this.__txtName.value.trim();
    var email = this.__txtEmail.value.trim();
    var country = this.__selCountry.value.trim();
    var subject = this.__txtSubject.value.trim();
    var message = this.__areaMessage.value.trim();

    // Validate each value
    var details = [];
    if (!name)
    {
        details.push(
            { message: "Name is required", control: this.__txtName }
        );
    }

    if (!this.__shared.isEmail(email))
    {
        details.push(
            { message: "Enter a valid e-mail address",
...
```

```

        control: this.__txtEmail }
    );
}

if(!country)
{
    details.push(
        { message: "Select a country", control: this.__selCountry }
    );
}

if(!subject)
{
    details.push(
        { message: "Enter a subject", control: this.__txtSubject }
    );
}

if(!message)
{
    details.push(
        { message: "Enter a message", control: this.__areaMessage }
    );
}

this.__shared.displayFormErrors(
    "Fix the following errors",
    this.getEl("divError"),
    details);

if (details.length === 0)
{
    alert("e-mail sent");
    $$asInputEl(this.getEl("btnClear")).form.reset();
}
},

```

11. When **Clear** is clicked, *divError* should be hidden. Expose *btnClear* and connect an event handler to do so.

## Part 2: Shortcuts and Browser compatibility issues

---

In this section you will make this code look more like JavaScript in the HSWeb framework by adding some common shortcuts. This will also improve the cross-browser compatibility of your code.

- Foundations uses the function \$\$fcc to represent Function.createCallback. Add this shortcut to *Shared.js* now:

```
//#region Define shortcuts similar to those in HyperspaceWeb
var $$fcd = Function.createDelegate;
var $$fcc = Function.createCallback;
//#endregion
```

- Replace *Function.createCallback* with *\$\$fcc* where it is used in *displayFormErrors*.

There are several properties that can make working with the DOM a bit easier, **but you need to be careful in terms of browser support**.

innerHTML (all browsers)	Allows you to set markup directly inside an element that is capable of containing other elements. For example:  myDiv.innerHTML=" <strong>Hello!</strong> ";
innerText (IE / Chrome)	Add text to an element that can contain content. For example, you could replace this:  myDiv.appendChild(document.createTextNode("hello"));  with:  myDiv.innerText="hello";
textContent (Chrome / Firefox)	Same as innerText, but used with Firefox/Chrome

- Replace the loop that clears all child nodes by simply setting the *innerHTML* of the message div to an empty string:

```
//#region Remove all parts of previous message
// while (messageDiv.childNodes.length > 0) {
//   messageDiv.removeChild(messageDiv.firstChild);
// }
messageDiv.innerHTML = "";
//#endregion
```

- Add a new shortcut called *\$\$dom*. In HyperspaceWeb, this shortcut encapsulates a variety of DOM related functions:

- Look here for a digital copy: **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\scripts\dom.txt**

```
///#region Foundations shortcuts
var $$fcd = Function.createDelegate;
var $$fcc = Function.createCallback;
var $$dom = {
    setInnerText: function Example$Dom$setInnerText(el, text) {
        /// <summary>Sets the text content of a dom element</summary>
        /// <param name="el" type="HTMLElement">dom element</param>
        /// <param name="text" type="String">text value</param>
        if (typeof (document.body.textContent) !== "undefined") {
            el.textContent = text;
        }
        else {
            el.innerText = text;
        }
    },
    getInnerText: function Example$Dom$getInnerText(el) {
        /// <summary>Get the text content of a dom element</summary>
        /// <param name="el" type="HTMLElement">dom element</param>
        if (typeof (document.body.textContent) !== "undefined") {
            return el.textContent;
        }
        else {
            return el.innerText;
        }
    }
};
//#endregion
```

5. Use `$$dom.setInnerText` to add the error message to each link as it is created.
6. Verify that the errors show as expected in all browsers.
7. Encapsulate the CSS class methods from `Sys.UI.DomElement` into `$$dom` as well:

```
var $$dom = {
    addCssClass: Sys.UI.DomElement.addCssClass,
    removeCssClass: Sys.UI.DomElement.removeCssClass,
    ...
}
```

8. Replace all instances of `Sys.UI.DomElement.addCssClass` and `Sys.UI.DomElement.removeCssClass` in `displayFormErrors` with the corresponding shortcut from `$$dom`.
9. Test the email form to make sure that it still works.
10. Continue working on the "If you have time" steps.

## If you have time

---

1. All links should have titles that describe where the link will take the user when followed. Modify `displayFormErrors` to set the "title" attribute in new links accordingly.
2. Modify `displayFormErrors` so that controls with an error have a light red background (#FFEEEE). Be sure remove the color when the problem is corrected, or when Clear is clicked.
3. Rather than confirming that the email was sent using alert, add a new div called `divMessage` to display the confirmation message. Instead of being red, the text color should be green, and the background color should be #EEFFEE. In this case details are not necessary.
4. In the confirmation, indicate which country the office is in that will be handling the message.
5. If the message is marked as urgent, inform them that the response will come as soon as possible as part of the confirmation message.

# Creating Script Controls

## Clock Should Track Time

Currently the clock just renders to a set of HTML spans contained within a div. The next step is to write the JavaScript that will actually make the clock tick.



*Clock Keeping time with 12 and 24 hour displays*

## The ScriptControl

The clock is currently inheriting from *WebControl*, which renders only to HTML. To add the necessary behavior, we need to inherit from *ScriptControl*, which renders to both HTML and JavaScript.

### ASP.NET:

```
<etcw:Clock runat="server"
    ID="clkMain"
    IsTwentyFourHour="true"
    CssClass="clkMain" />
```

### HTML:

```
<div id="clkMain" class="clkMain">
    <span id="clkMain_spanHours">24</span>:
    <span id="clkMain_spanMinutes">00</span>:
    <span id="clkMain_spanSeconds">00</span>
    <span id="clkMain_spanAmPm"></span>
</div>
```

### JavaScript:

```
Type.registerNamespace("Epic.Training.Core.Controls.Time.Web");
Epic.Training.Core.Controls.Time.Web.Clock = function Core$Clock(...){
{
    ////<summary>JavaScript class to automate the clock control</summary>
    ...
}
```

Two things are required for a script control to function properly.

1. What script files to load, known as *Script References*

## 2. What code to execute, known as *Script Descriptors*

Inheriting from ScriptControl forces you to implement two abstract methods that supply these required pieces. The methods are called, GetScriptReferences and GetScriptDescriptors.

For the clock script, you'll need some way to track the current time. JavaScript provides built-in functions for this purpose: setTimeout and setInterval.

## Working with Timeouts

Timeouts are useful when an event should occur once after a set amount of time. The two JavaScript functions that you can use in this case are:

setTimeout	<pre>tid = setTimeout(code, delay)</pre> <ul style="list-style-type: none"><li>• <b>delay</b>: Time to wait before executing code (milliseconds)</li><li>• <b>code</b>: Event handler (use <code>\$\$fcd</code> to create)</li><li>• <b>tid</b>: Unique ID used to clear the timeout before it occurs</li></ul>
clearTimeout	<pre>clearTimeout(tid)</pre> <ul style="list-style-type: none"><li>• Cancel the timeout event corresponding to the given timeout id (<b>tid</b>).</li></ul>

## Working with Intervals

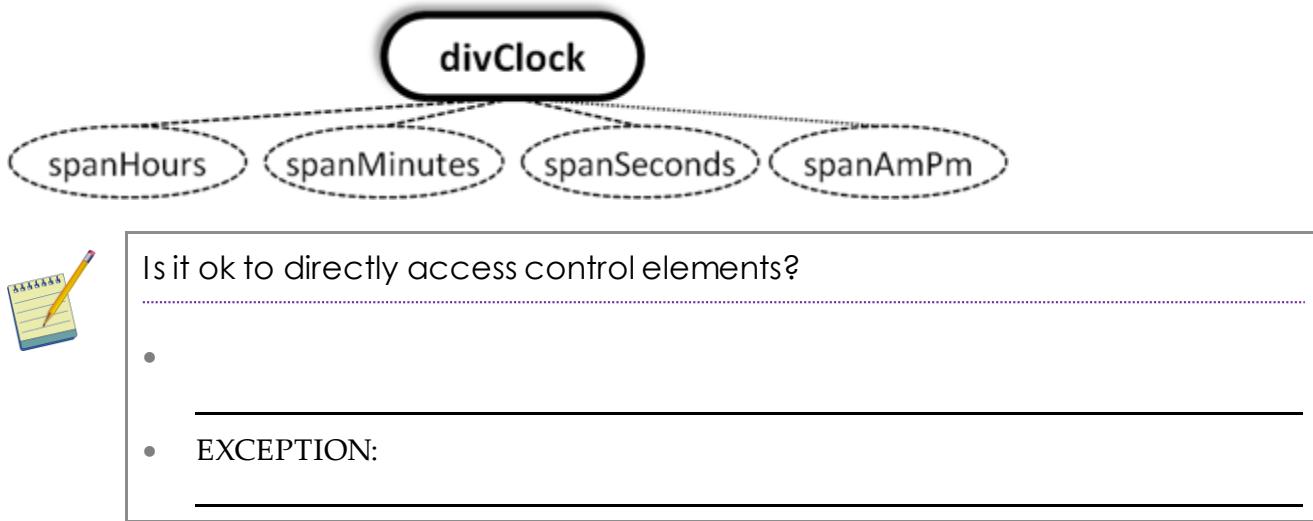
While a timeout is good for a 1-time event, an interval is more appropriate for a clock, which ticks at regular intervals. The following two functions are useful when working with intervals:

setInterval	<pre>pid = setInterval(code, period)</pre> <ul style="list-style-type: none"><li>• <b>period</b>: Period of ticks between executions of the code (milliseconds)</li><li>• <b>code</b>: Event handler (use <code>\$\$fcd</code> to create)</li><li>• <b>pid</b>: Period ID used to clear the event before it occurs</li></ul>
clearInterval	<pre>clearInterval(pid)</pre> <ul style="list-style-type: none"><li>• Cancel the interval event corresponding to the given period id (<b>pid</b>).</li></ul>

## Maintaining Encapsulation

When working with any kind of custom control, it is important to respect the encapsulation of the control. With script controls, this means you should not assume the structure of the HTML content

behind the control from outside that control's script file. it also means you should not access any part of the script that is private from outside the script file.



## Exercise: Make the Clock Keep Time

In this exercise you will add a script, *Clock.js*, so that the clock displays the current time, updating every second. You will also add a feature to toggle between 12 and 24 hour display by clicking on the clock.

A shell of *Clock.js* has already been provided to you in the Assets project.

### Part 1: Change Clock to a ScriptControl

Currently the Clock class is a *WebControl*, meaning that it renders to HTML only. In order to keep track of time efficiently, we need to change it to a *ScriptControl*.

1. Add the .NET reference: *System.Web.Extensions* to *Epic.Training.Core.Controls.Time.Web* to *Training.Core.Controls.Time.Web*
2. Open *Clock.cs*
3. Indicate that *Clock* inherits from *ScriptControl* rather than *WebControl*.
  - Keep the *INamingContainer* interface.
4. To implement the additional methods of *ScriptControl*:
  - Right click (or use the **CTRL + .** shortcut) on the reference to *ScriptControl* that indicates that *Clock* inherits from it
  - Select **Implement Abstract Class**
5. Examine the *Clock* constructor. Notice that the base class call is no longer valid. This is because there isn't a constructor of *ScriptControl* that allows you to specify the tag.

6. Remove `:base("div")` from the constructor
7. By default the control will render as a span, but you want it to render as a div. To do so, override the `HtmlTextWriterTag` property:

```
protected override HtmlTextWriterTag TagKey
{
    get
    {
        return HtmlTextWriterTag.Div;
    }
}
```

8. Scroll down to the bottom of `Clock.cs`. You will see two additional functions. The first is `GetScriptDescriptors`:

```
protected override IEnumerable<ScriptDescriptor> GetScriptDescriptors ()
{
    throw new NotImplementedException();
}
```

A `ScriptDescriptor` is a class that returns the JavaScript code to run when this `ScriptControl` is rendered. In this case, you want to do two things:

- Create an instance of the JavaScript clock from `Clock.js`
- Add the clock to the object registry, so the `PageBehavior` can find it.

Both of these tasks are done for you by `Epic.Training.Core.Controls.Web.ManagedDescriptor`, which resides in `Epic.Training.Core.Controls.Web`.

9. Add a reference to `Epic.Training.Core.Controls.Web` (using the Browse button) to your `Epic.Training.Core.Controls.Time.Web` project.
10. Set a corresponding project dependency.
11. Add the following code to `GetScriptDescriptors` so that an instance of the JavaScript `Clock` class will be instantiated:

```
protected override IEnumerable<ScriptDescriptor> GetScriptDescriptors ()
{
    yield return new ManagedDescriptor(
        "Epic.Training.Core.Controls.Time.Web.Clock", // JavaScript class
        this, // Corresponding server control
        ManagedDescriptorType.Control, // Has UI: Control, No UI: Component
        "" + this._hours.ClientID + "", // Client ID of hours span
        "" + this._minutes.ClientID + "", // Client ID of minutes span
        "" + this._seconds.ClientID + "", // Client ID of seconds span
        null);
```

```

        "" + this._amPm.ClientID + "'' // Client ID of AM/PM span
    );
}

```

Notice that `GetScriptDescriptors` returns `IEnumerable<ScriptDescriptor>`, rather than just `ScriptDescriptor`. This means that the results will likely be iterated over (for example, using `foreach`), and also means that you could return multiple `ScriptDescriptors` if necessary. The correct syntax for returning a `ScriptDescriptor` in this context is to use `yield return`, as indicated above.

The `ManagedDescriptor` you created above produces the JavaScript code that will be sent to the client. It's constructor takes the following arguments:

constructor	The constructor to call from JavaScript to create an object that handles the behavior of the control. In the case of the clock, it will be the object that ticks every second and updates the clock's display.
serverControl	The ASP.NET control that produces the HTML that the JavaScript object will be anchored to. In the case of a clock, the ASP.NET control renders to a div that contains a span for hours, minutes, seconds and AM/PM. The JavaScript Clock object will anchor to the div.
type	<p>There are two types of Managed Descriptors:</p> <ol style="list-style-type: none"> <li><b>Component</b> - A component-type managed descriptor creates a JavaScript class that <b>is not</b> anchored to an HTML element in the DOM. This is because components typically do not have a visual piece.</li> <li><b>Control</b> - A control-type managed descriptor creates a JavaScript class that <b>is</b> anchored to an HTML element in the DOM. This is because controls represent widgets that the user can interact with.</li> </ol>
args	An array of additional arguments to pass to the JavaScript constructor.

3. Modify `GetScriptReferences` in `Clock.cs` so that both `Clock.js` and `PageBehavior.js` are added to the `ScriptManager` automatically whenever a clock control is rendered:

```

protected override IEnumerable<ScriptReference> GetScriptReferences()
{
    yield return new
    ScriptReference("~/Assets/Training/Core/Scripts/PageBehavior.js");
    yield return new

```

```
ScriptReference("~/Assets/Training/Core/Scripts/Clock.js");  
}
```

**The above order is important.**

*PageBehavior.js* must be loaded before *Clock.js* because the clock will automatically add itself to the object registry created by *PageBehavior.js*. If that registry doesn't exist, you will encounter errors.

4. Set a breakpoint within the JavaScript constructor of *Clock*.
5. Run your site in IE to verify that the clock is being constructed.
6. Once you have verified that the clock is loading, remove the breakpoint.



Notice that the delegate for the interval (assigned to the `_del` field) is created using `Function.createDelegate`, rather than `$$fcd`. This is because the Foundations shortcuts that you added are in the *Pages* project, not within an assembly accessible by *Controls.Time*.

The elegant solution would be to recognize that the `$$fcd`, `$$fcc` and `$$dom` shortcuts are actually useful for any web application, and are not product dependent. Therefore, they should be moved to a new assembly that is an even lower level than *Controls* or *Controls.Time*, for example, *Epic.Training.Core.Web*. In this case, the functional area would be *Base* (which appears in the folder path but not the namespace) since all functional areas on the web platform could make use of these shortcuts. This similar to how the shortcuts are structured in the HS Web framework.

Since this process is very involved with rather little benefit in terms of the in-class exercise, it will not be done here. Feel free to try it out if you choose.

## Part 2: Setup the clock to tell time

1. Examine the URL listed in the `_update` method of *Clock.js* (CTRL+ Click to open the page in Visual Studio) to remind yourself about the JavaScript *Date* class.
2. Within `_update`, create a new instance of today's date using
  - `var today = new Date();`
3. Modify the `_update` method so that the time is set correctly. Be sure to take into account 12 and 24 hour time
  - You know which it is by examining `this._isTwentyFourHour`
  - The spans are available under `this._<spanHours / spanMinutes / spanSeconds / spanAmPm>`

- Pieces of the time can be retrieved using the `format(pattern)` instance method of `today`. Some useful patterns are:
  - "hh": 12h clock hours, padded
  - "HH": 24h clock hours, padded
  - "mm": minutes, padded
  - "ss": seconds, padded
  - "tt": AM/PM for a 12h clock
- For example, to set the hours of a 12h clock, use:
  - `this.__spanHours.innerHTML = today.format("hh");`



The `format` method is a Microsoft Ajax library extension of the `Date` class and is not part of standard JavaScript.

4. Attempt the previous step on your own first. If you get stuck, refer to the following sample solution:

```
__update: function Core$Clock$__update()
{
  ///<summary>Called every second to update the stored and displayed time
  values.</summary>
  // *** Add code here to set the clock to the current time ***
  // *** Be sure to check this.__isTwentyFourHour to set it correctly ***
  // *** For details on getting the current time, Check:
  // *** http://wiki/main/Foundations/Training/WTC/Reference/JavaScript/Basics/Date\_Class

  //EXERCISE: Set value of each span to display the current time

  var today = new Date();
  this.__spanMinutes.innerHTML = today.format("mm");
  this.__spanSeconds.innerHTML = today.format("ss");
  if (this.__isTwentyFourHour)
  {
    this.__spanHours.innerHTML = today.format("HH");
    this.__spanAmPm.innerHTML = "";
  }
  else
  {
    this.__spanHours.innerHTML = today.format("hh");
    this.__spanAmPm.innerHTML = today.format("tt");
  }
}
```

```
}

// Setup interval if it isn't setup already
if (!this.__pid) {
    this.__pid = setInterval(this.__del, 1000);
}

}
```

## Part 3: Passing server-control properties to the client

The C#.NET Clock class has a property *IsTwentyFourHour* and you need to send that value to the client. Fortunately, the *ManagedDescriptor* constructor is setup so that passing additional arguments is relatively easy.

1. Examine *ManagedDescriptor.cs* to determine how to pass additional arguments to the clock constructor through the constructor of *ManagedDescriptor*.
2. Adjust the Clock constructor in *Clock.js* so that it accepts an additional parameter, *isTwentyFourHour*, and assign that value to *this.\_\_isTwentyFourHour*.
3. Add the XML documentation to the clock constructor in *Clock.js* corresponding to the new parameter.
  - There is a snippet for this: *param*
4. In *Clock.cs*, pass the additional argument to the *ManagedDescriptor* constructor. Pass **true** if *IsTwentyFourHour* is **true**, or **false** if it is not.



### Remember that JavaScript is case sensitive.

You cannot simply pass *IsTwentyFourHour.ToString()*, because this will return "True" or "False" instead of "true" or "false". Instead, try:

```
IsTwentyFourHour.ToString().ToLower()
```

5. Verify that changing the *IsTwentyFourHour* attribute in your markup affects the display of the clock on the client.
6. If you get stuck, refer to the following solution:

#### In *Clock.cs*:

```
protected override IEnumerable<ScriptDescriptor> GetScriptDescriptors ()
{
    yield return new ManagedDescriptor(
        "Epic.Training.Core.Controls.Time.Web.Clock", // JavaScript class
```

```

    this, // Corresponding server control
    ManagedDescriptorType.Control, // Has UI: Control, No UI: Component
    """ + this._hours.ClientID + "", // Client ID of hours span
    """ + this._minutes.ClientID + "", // Client ID of minutes span
    """ + this._seconds.ClientID + "", // Client ID of seconds span
    """ + this._amPm.ClientID + "", // Client ID of AM/PM span
    this.IsTwentyFourHour.ToString().ToLower() // If the clock is a 12h or 24h
);
}
}

```

### In Clock.js:

```

Epic.Training.Core.Controls.Time.Web.Clock = function Core$Clock(element,
clientID, hoursClientID, minutesClientID, secondsClientID, amPmClientID,
isTwentyFourHour) {

    ...
    ///<param name="isTwentyFourHour" type="Boolean">True for a 24h clock,
    /// false for a 12h clock</param>
    ...

    // Calls constructor of Epic.Training.Controls.ControlHandler,
    // which sets up Elements and Objects.

    Epic.Training.Core.Controls.Time.Web.Clock.initializeBase(this,
        [element]);

    // Get parts of clock from element
    // note $get is an MS Ajax shortcut to document.getElementById
    this.__isTwentyFourHour = isTwentyFourHour;
    this.__spanHours = $get(hoursClientID, element);
    this.__spanMinutes = $get(minutesClientID, element);
    this.__spanSeconds = $get(secondsClientID, element);
    this.__spanAmPm = $get(amPmClientID, element);

    // Set timeout delegate
    this.__del = Function.createDelegate(this, this.__update);

    // Start the periodic update process
    this.__update();
};

}

```

## Part 4: Accessing custom controls on the page

In this part of the exercise, you will learn how to access a custom control from the page that contains it. The goal is for the clock to switch between 12 and 24h time when clicked.

Currently the master page is not setup to include *PageSettings*. Use *Customer.aspx* as a reference for how to accomplish each of these steps.

1. Create a new JS file called *ExampleMasterBehavior.js*
2. Create a JavaScript class called *ExampleMasterBehavior*,
  - Remember to use the *assetpagebehavior* code snippet
  - It should be part of the *Epic.Training.Example.Web.Pages* namespace
  - Use *ExampleMasterBehavior* as the class name
  - Don't forget to add an IntelliSense reference to *Shared.js*.
3. In *ExampleMaster.Master*, Register the *Epic.Training.Core.Controls.Web* assembly and namespace, so the *PageSettings* tag is available:

```
<%@ Register Namespace="Epic.Training.Core.Controls.Web"  
Assembly="Epic.Training.Core.Controls.Web"  
TagPrefix="etcw" %>
```

4. Add a *PageSettings* to the master page.
  - a. Set the *ClientClass* attribute to *Epic.Training.Example.Web.Pages.ExampleMasterBehavior*
  - b. Set the *ClientScriptPath* attribute to *~/ExampleMasterBehavior.js*.



You don't need to add *clkMain* as a managed control, because it uses the *ManagedDescriptor* to render itself, which adds it to the managed controls automatically.

5. Add a new public get and set methods to the JavaScript Clock class to alter *\_isTwentyFourHours*:
  - a. *get\_isTwentyFourHour()*
  - b. *set\_isTwentyFourHour(value)*
6. In addition to setting *this.\_isTwentyFourHour* appropriately in *set\_isTwentyFourHour(value)*, you should also:
  - a. Clear the current interval stored in *this.\_pid*
  - b. Set *this.\_pid* to *null*
  - c. Call *this.\_update()*

These steps will ensure that the clock display switches as soon as the hour format changes, rather than when the next timeout occurs (up to one second later).

```
//#region properties

get_isTwentyFourHour: function Core$Clock$get_isTwentyFourHour() {
    /// <value>(public) This is a 24h clock</value>
    /// <returns type="Boolean">
    /// True for a 24h clock, false for a 12h clock</returns>
    return this.__isTwentyFourHour;
},

set_isTwentyFourHour: function Core$Clock$set_isTwentyFourHour(value) {
    /// <value>(public) This is a 24h clock</value>
    /// <param name="value" type="Boolean">
    /// Set to true for a 24h clock, false for a 12h clock.</param>
    this.__isTwentyFourHour = value;
    clearInterval(this.__pid);
    this.__pid = null;
    this.__update();
},
//#endregion
```



The <value> XML comment is used with JavaScript get and set property methods, rather than <summary>.

For more information, see:

<http://msdn.microsoft.com/en-us/library/hh542724.aspx>

7. In ExampleMasterBehavior, add a method called `__clkMainClick`. For now, just alert that the clock was clicked.
  - Place it in a region for event handlers
8. Attach an event to the element of `clkMain` in `onLoaded`. You can access the element using `get_element()`, as shown below:

```
var clockDelegate = $$fcd(this, this.__clkMainClick);
$addHandler(this.getCtrl("clkMain").get_element(),"click", clockDelegate);
```

9. Run your site and verify that clicking the clock displays the alert.
10. If it works, comment out the alert.

## Part 5: Casting for IntelliSense

- Within `_clkMainClick`, try toggling the type of clock by using `get_isTwentyFourHour()` and `set_isTwentyFourHour()`. Notice that you don't receive IntelliSense for these methods:

```
var isTwentyFourHour = this.getCtl("clkMain").get_isTwentyFourHour();
this.getCtl("clkMain").set_isTwentyFourHour(!isTwentyFourHour);
```

- Run your site and try clicking the clock. It should work, even though you didn't receive IntelliSense. The reason that IntelliSense is not working is that the data type indicated for `getCtl` is `Sys.UI.Control`, not `Epic.Training.Core.Controls.Time.Web.Clock`.
- To get better IntelliSense support, add a new field to your class called `_clkMain`, which includes the data type of the clock in the field XML comment:

```
Epic.Training.Example.Web.Pages.ExampleMasterBehavior = function
Example$ExampleMasterBehavior(clientId)
{
    ///<field name="_clkMain" type="Epic.Training.Core.Controls.Time.Web.Clock">

Epic.Training.Example.Web.Pages.ExampleMasterBehavior.initializeBase(this
, [clientId]);
};

Epic.Training.Example.Web.Pages.ExampleMasterBehavior.prototype = {
    //#region fields
    _clkMain: null,
    //#endregion

    ...
}
```

- Next, within `ExampleMasterBehavior.js` add an IntelliSense reference to `Clock.js`:

```
/// <reference path("~/Shared.js" />
/// <reference path "~/Assets/Training/Core/Scripts/PageBehavior.js" />
/// <reference path "~/Assets/Training/Core/Scripts/Clock.js" />
```

- Retype the code in `_clkMainClick`, this time using `this._clkMain`. You should get IntelliSense this time:

```
##region Event Handlers
_clkMainClick: function Example$ExampleMasterBehavior$_clkMainClick(event) {
    ///<summary>Click event handler for clkMain</summary>
    var isTwentyFourHour = this._clkMain.get_isTwentyFourHour();
    this._clkMain.set_isTwentyFourHour(
},
//#endregion
```

**set\_isTwentyFourHour(Boolean value)**  
 (public) Set the type of clock  
**value:** Set to true for a 24h clock, false for a 12h clock.

*Accurate IntelliSense is now available for the Clock Class*

6. Test your site again. You will find that it doesn't actually work. What is the problem?

■

- 
7. Change `onLoaded` so that `_clkMain` is populated before the click event is attached to it:

```
onLoaded: function Example$ExampleMasterBehavior$onLoaded() {  
    //<summary>Called by the constructor after "onLoad".</summary>  
    this._clkMain = this.getCtl("clkMain");  
    var clockDelegate = $$fcd(this, this._clkMainClick);  
    $addHandler(this._clkMain.get_element(), "click", clockDelegate);  
},
```

8. Run your site to verify that it still works as expected.

## Wrap Up

---

1. What is allowing you to pass additional arguments to the constructor of `ManagedDescriptor`?

■

# Loading Data on the Client

Rather than hard-coding table content, we want the page to be more dynamic. In this section the inventory table will be populated on the client, rather than being hard coded.

## Populate the Inventory at Run Time

- Now table is statically defined in markup
- Goal:** Populate at runtime

The diagram illustrates the transition from a static table definition to a dynamic, populated table. On the left, a 'Catalog' table is shown with a single row for 'Blue' cheese, containing columns for Name, Image, Price, and Details. An arrow points to the right, leading to a second 'Catalog' table which now contains three rows: 'Blue', 'Cheddar', and 'Swiss'. Each row includes an image of the cheese slice and a 'Details' link.

Name	Image	Price	Details
Blue		90.00	<a href="#">View</a>

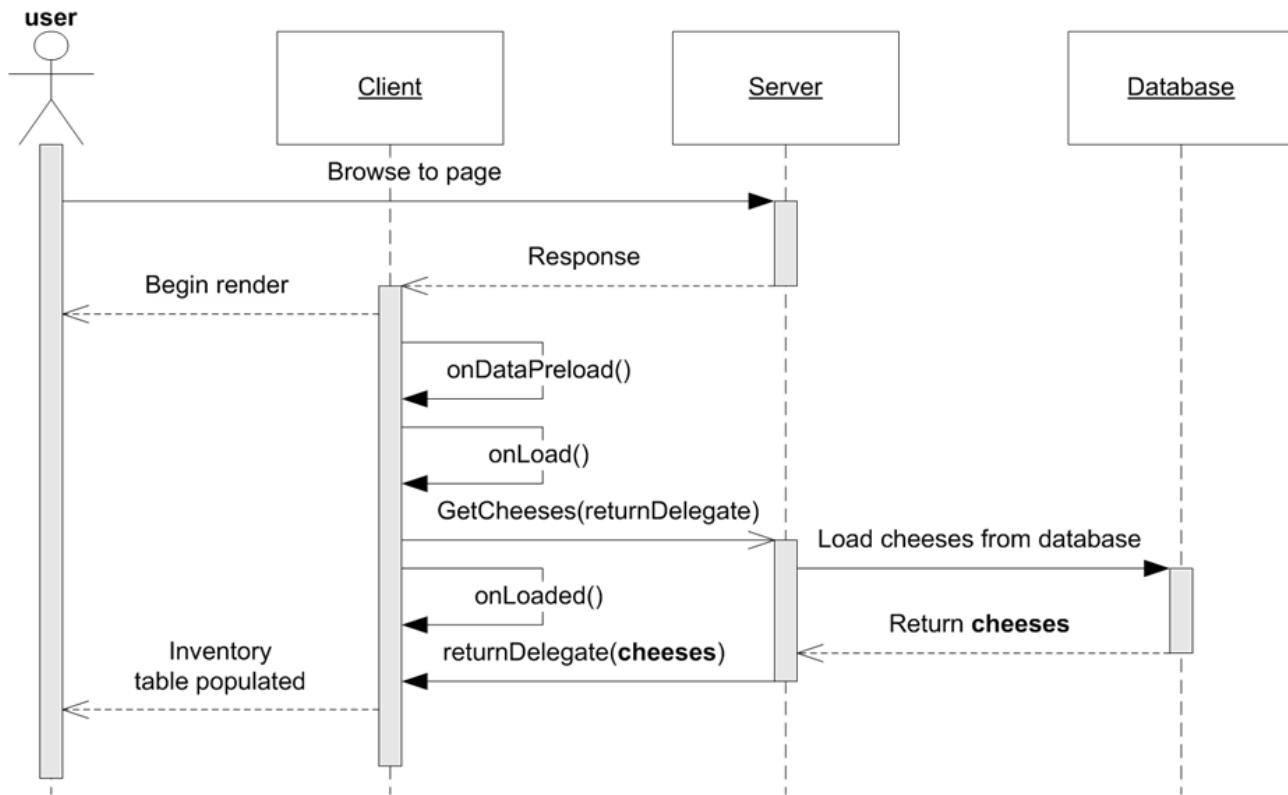
  

Name	Image	Price	Options
Blue		80	<a href="#">Details</a>
Cheddar		76	<a href="#">Details</a>
Swiss		57	<a href="#">Details</a>

*Example of how the inventory will look*

## Load Pattern

As mentioned in the first lesson, the load pattern used by Foundations is structured to take advantage of the concurrent processing potential of the client, server and database.



### Recommended Load Pattern

onDataPreload	<ul style="list-style-type: none"> <li>Opportunity to do work before data request is sent.</li> </ul>
onLoad	<ul style="list-style-type: none"> <li>Send asynchronous request(s) for data</li> </ul>
onLoaded	<ul style="list-style-type: none"> <li>Page is loaded</li> <li>Data is not loaded yet (asynchronous request is still in flight)</li> <li>Opportunity to do work while waiting for data (concurrency)           <ul style="list-style-type: none"> <li>Initialize private fields</li> <li>Create delegates</li> <li>Connect events</li> </ul> </li> </ul>
returnDelegate	<ul style="list-style-type: none"> <li>Typically named <i>onDataLoaded</i></li> <li>Called by response to asynchronous request for data.</li> <li>Data returned is available in first parameter</li> </ul>

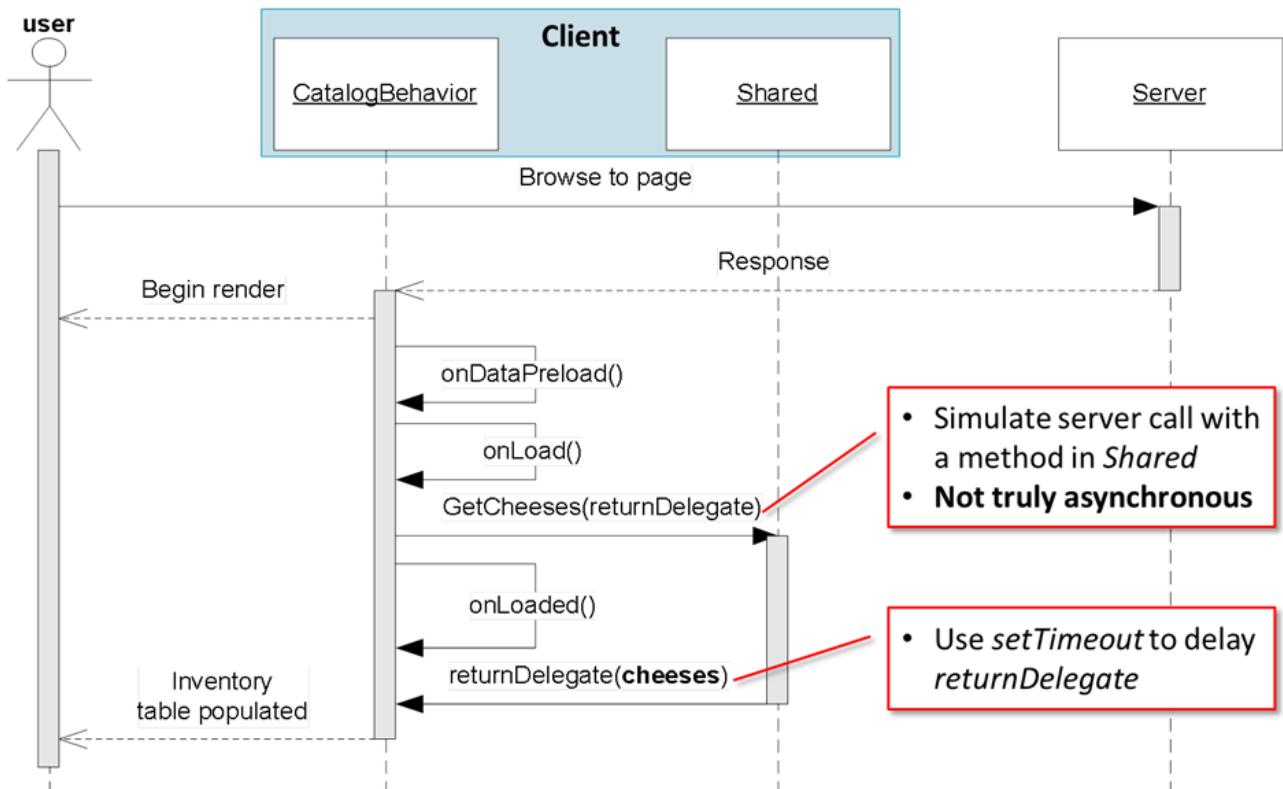


### For now, this load pattern is simulated

We have not covered how to make asynchronous calls to web services, so this load pattern cannot be fully realized until a later lesson.

## Simulated Load Pattern

Rather than making a call to the server, we will place methods in *Shared* to simulate calls to the server. The main difference in terms of how you code is that **the call is not actually asynchronous**. However, the load delegate is still delayed using `setTimeout`, to make it seem like a server call is under way.



*Illustration of the simulated load pattern*

## Using the CollectionGrid to Edit Data

Your employer wants to be able to edit cheese inventory items. Fortunately, the *CollectionGrid* control includes several advanced features that make this possible.

### Advanced Features of the Collection Grid



Catalog		
Name	Image	Options
Blue		Details
Cheddar		Details
Swiss		Details

Name	Image	Description	Options
Blue		Strong blue cheese flavor running throughout	Edit Details
Cheddar	<input type="text" value="Cheddar"/>	A very mild cheddar flavor.	<input type="button" value="Done"/>
Swiss		Mellow flavor with many holes	Edit Details
<a href="#">Add Cheese</a>			

Options to edit cheeses are required

- Cheeses should be editable
- Option to add additional cheeses
- *CollectionGrid* supplies framework to support these options



#### Remember

The *CollectionGrid* is a control supplied for training. The actual Grid control in HyperspaceWeb is similar, but has more options.

### Parameter Validation

You should validate all parameters to public functions and methods. This ensures that programmers using your function do so in the way it was intended to be used. This will also mitigate the loosely typed nature of JavaScript by imposing a data type on the input.

The Ajax library makes it relatively easy to implement validation in a way that won't slow down production (minimized) code. You will find examples of this in *CollectionGrid.js*. For example:

```
addTableOptionHandler: function
Core$CollectionGrid$addTableOptionHandler(option, handler) {
    ///<summary>Add an event handler that will be called when the
    /// indicated table option is clicked. </summary>
    ///<param name="option" type="String">The name of the option</param>
```

```
///<param name="handler" type="Function">Handler created using $$fcd
///  that will be called when the table option link is
///  clicked.</param>

//#if DEBUG
var e = Function.validateParameters(arguments, [
  { name: "option", type: String },
  { name: "handler", type: Function }
]);
if (e) { throw e; }
//#endif
```

Anything placed between `//#if DEBUG` and `//#endif` will be stripped when the JavaScript file is minimized



Use parameter validation on all public functions and methods

## Exercise: Populate the Inventory Table

In this exercise you will replace the static inventory table with a dynamic one using the simulated load pattern.

### Part 1: Get the sample data

1. In order to simulate a web-service call, you will need to include the following in your *Shared* static class:

<code>__sampleData</code>	This is a private array that contains cheese objects.  Note that the object properties are in Pascal case. This is because the objects will eventually generated automatically based on C# classes, which use Pascal case for their properties.
<code>GetCheeses</code>	<code>GetCheeses (returnDelegate)</code>  Function to return the private property <code>__sampleData</code> .  Note that the function doesn't return the list directly. Instead, it calls the <code>returnDelegate</code> via a timeout of 1 second. This is similar to how web-service calls to the server will work.

	When the data returns from the server, the return delegate will be called by the web service, passing the resulting data as an argument.
--	------------------------------------------------------------------------------------------------------------------------------------------



Notice that *GetCheeses* is in Pascal case instead of camel case, as is typical for JavaScript methods. This is because the web service method name will eventually be defined in C#.

This difference in casing will make it easy to tell when a method call involves a trip to the server.

2. Copy the method and data described above from:

- **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\scripts\getCheeses.txt**

3. Paste them into the top of *Shared*.

4. Your result should look like the following:

```
//#region Shared static class
Epic.Training.Example.Web.Pages.Shared = {

    //#region Sample data and methods to simulate web-service call
    __sampleData: [
        { Name: "Blue", Item: "110",
            ImagePathSmall: "./images/Blue-Thumb.png",
            ImagePathLarge: "./images/Blue-Full.png", Weight: "25",
            Description: "Strong blue cheese flavor running throughout",
            Price: "80"
        },
        { Name: "Cheddar", Item: "220",
            ImagePathSmall: "./images/Cheddar-Thumb.png",
            ImagePathLarge: "./images/Cheddar-Full.png", Weight: "35",
            Description: "A very mild cheddar flavor.", Price: "76"
        },
        { Name: "Swiss", Item: "320",
            ImagePathSmall: "./images/Swiss-Thumb.png",
            ImagePathLarge: "./images/Swiss-Full.png", Weight: "67",
            Description: "Mellow flavor with many holes", Price: "57"
        }
    ],
    GetCheeses: function Example$Shared$GetCheeses(returnDelegate) {
        ///<summary> A function that returns sample data in a similar
        /// format to what a web service would return.</summary>
        ///<param name="returnDelegate" type="Function">
        /// Function that will be called when the data is returned.</param>

        // Pass sample data as an argument to the return delegate
        var toCall = $$fcc(returnDelegate, this.__sampleData);
    }
}
```

```
// Call the return delegate, just like an asynchronous call would  
setTimeOut(toCall, 1000);  
,  
//#endregion  
  
...
```

## Part 2: Get the CollectionGrid ScriptControl

Within HyperspaceWeb, Foundations provides a Grid control. Since we do not yet have access to this framework, a sample control called the CollectionGrid has been provided.

1. Copy the folder *Controls.Grid* from:
  - **I:\Internal\Advanced\Web Tech Camp\Big-Mouse\projects\Code**
2. Paste the folder into the following part of your Code branch:
  - **C:\EpicSource\WTC\Code\Training\Core\Web**
3. Now you should now have three functional areas listed in that folder:
  - Controls
  - Controls.Grid
  - Controls.Time
4. Add *Training.Core.Controls.Grid.Web* to your solution as an existing project
5. Setup *Training.Core.Controls.Grid.Web* so that it has a project dependency on *Training.Core.Controls.Web*.
6. Build your solution.
7. In *Training.Example.Web.Pages*, add a reference to and project dependency on *Training.Core.Controls.Grid.Web*
8. Register the *Epic.Training.Core.Controls.Grid.Web* assembly in *Catalog.aspx*, just beneath <%@ Page ... %>:

```
<%@ Register Assembly="Epic.Training.Core.Controls.Grid.Web"  
Namespace="Epic.Training.Core.Controls.Grid.Web"  
TagPrefix="etcw" %>
```

9. Comment out the current inventory table in *Catalog.aspx*.
10. Recompile your solution.
11. Add the following control:

```
<etcw:CollectionGrid runat="server">
```

```

ID="gridInventory" ShowHeaders="true">
    <ColumnDefinitions>
        <etcw:Column Header="Name" BoundProperty="Name" />
        <etcw:Column Header="Image" BoundProperty="ImagePathSmall" />
    </ColumnDefinitions>
    <RowOptions>
        <etcw:Option Name="Details"
            ToolTip="View or edit details about this cheese" />
    </RowOptions>
</etcw:CollectionGrid>

```

12. Save and run your page.
13. Navigate to the inventory page. You should see an empty table with appropriate column headers.
14. Setup *Inventory/Catalog.aspx* to include a *PageSettings* control:
  - a. Create a JavaScript file and class named *CatalogBehavior* that will be used to manage controls on the catalog page (remember to use the *assetpagebehavior* snippet).
  - b. The namespace should be *Epic.Training.Example.Web.Pages.Inventory*
  - c. Add a *PageSettings* control to the catalog page that points to the behavior class (*ClientClass* attribute) and the .js file containing the behavior class (*ClientScriptPath* attribute).
    - Don't forget to register *Epic.Training.Core.Controls.Web* within Catalog.ascx

Remember that it is not necessary to add *gridInventory* as a control entry, because it is rendered using the *ManagedDescriptor* class, which adds the table to the managed controls automatically.

15. For IntelliSense support, add references to Shared.js and CollectionGrid.js to CatalogBehavior.js:

```

/// <reference path("~/Shared.js")/>
/// <reference path "~/Assets/Training/Core/Scripts/CollectionGrid.js" />
/// <reference path "~/Assets/Training/Core/Scripts/PageBehavior.js" />

```



Do not to include whitespace between the reference elements. This will break IntelliSense.

16. To get better IntelliSense support for grids inside *Inventory/CatalogBehavior.js*, create a private field named *\_gridInventory*. That is initialized to null.
17. Add a field XML comment to the constructor that specifies the type as:  
*Epic.Training.Core.Controls.Grid.Web.CollectionGrid*

18. In `onLoaded`, initialize `this.__gridInventory` using `this.getCtl("gridInventory")`.
19. Add a private instance member `_shared` that points to `Epic.Training.Example.Web.Pages.Shared`:

```
//#region fields  
_shared : Epic.Training.Example.Web.Pages.Shared,  
//#endregion
```

20. Place a breakpoint in the constructor of `CatalogBehavior`, then save and run your site and navigate to the inventory page.
21. Once you verify that `CatalogBehavior` is loading as expected, remove the breakpoint.

### Part 3: Create the Cheese JavaScript Class

The objects that are loaded with the `GetCheese` method are generic JavaScript objects that are not associated with any particular class. This is similar to how they will come from the server once web services are introduced in a later lesson. The problem here is that you can't add methods and other functionality to the cheese class in JavaScript, because there is no cheese class for the object.

In this section, you will create a JavaScript class to represent a cheese. After you get the cheeses from `Shared` (what you will do now) or the server (what you will switch to later), you will create a new cheese for each entry in the list, then add that cheese to the collection grid on the catalog page.

The `Cheese` class should exist in an assets project with the following properties:

- Owner = Training
  - Application = Example
1. Add a new class library project to your solution. The code properties are as follows:
    - a. **Name:** Example
    - b. **Location:** C:\EpicSource\WTC\BigMouse\Assets\Training
  2. Rename the project to `Training.Example.Web.Assets`
  3. Rename the default namespace and assembly name to:
    - a. `Epic.Training.Example.Web`
  4. Delete the default file `Class1.cs`
  5. Change the output path to `..\..\..\bin\` for all configurations.
  6. Add a new folder named `Scripts` to `Training.Example.Web.Assets`
  7. Add a new JavaScript file named `Cheese.js`
  8. Use the `ajaxclass` snippet to create the `Epic.Training.Example.Web.Cheese` class in `Cheese.js`.

```
/// <reference name="MicrosoftAjax.js" />
```

```

//*****
// Copyright <YEAR> Epic Systems Corporation
//*****

Type.registerNamespace("Epic.Training.Example.Web");
Epic.Training.Example.Web.Cheese = function
Example$Cheese(referenceObject)
{
    /// <summary>Class representing a Cheese</summary>
    var key;
    Epic.Training.Example.Web.Cheese.initializeBase(this);
    // If the reference object has been passed, copy
    // properties over that are defined in the prototype of this class
    if (referenceObject) // Check for null/undefined
    {
        for (key in referenceObject) // Iterate over properties
        {
            if (referenceObject.hasOwnProperty(key) && typeof (this[key]) !==
                "undefined")
            {
                this[key] = referenceObject[key]; // Copy the value over
            }
        }
    }
};

Epic.Training.Example.Web.Cheese.prototype = {
    //#region fields
    //#endregion
};
Epic.Training.Example.Web.Cheese.registerClass(
    "Epic.Training.Example.Web.Cheese");

```

9. On ExampleMaster.Master, add a script reference to the script manager so that cheese is loaded.

```
<asp:ScriptReference
    Path="~/Assets/Training/Example/Scripts/Cheese.js" />
```

**! Put it in the correct location**

Remember that an *asp:ScriptReference* must be placed inside the *Scripts* element of either a *ScriptManager* or *ScriptManagerProxy* control.

10. Add an IntelliSense reference to *CatalogBehavior.js* for the new cheese class:

```
/// <reference path("~/Assets/Training/Example/Scripts/Cheese.js") />
```

11. Add the following public fields to the cheese class. Note that they are in pascal case, because the cheese class will eventually be sent from the server (where the cheese is defined in C#).
- Be sure to document the fields with the appropriate XML comments.

Field	Data Type	Initial Value	Description
Name	String	"New Cheese"	Name of the cheese
Item	Number	-1	ID of the cheese
ImagePathSmall	String	"./images/GenericCheese-Thumb.png"	Thumbnail image path
ImagePathLarge	String	""	Full Image Path
Weight	Number	0	Weight in pounds
Description	String	"Fresh!"	Description of the cheese
Price	Number	0	Cost in dollars

```
/// <reference name="MicrosoftAjax.js" />
// ****
// Copyright <YEAR> Epic Systems Corporation
// ****
Type.registerNamespace("Epic.Training.Example.Web");
Epic.Training.Example.Web.Cheese = function
Example$Cheese(referenceObject)
{
    /// <summary>Class representing a Cheese</summary>
    /// <field name="Name" type="String">(public)Name of the cheese</field>
    /// <field name="Item" type="Number">(public)ID of the cheese</field>
    /// <field name="ImagePathSmall" type="String">(public)Thumbnail image path</field>
    /// <field name="ImagePathLarge" type="String">(public)Full image path</field>
    /// <field name="Weight" type="Number">(public)Weight in pounds</field>
    /// <field name="Description" type="String">(public)Description of the cheese</field>
```

```

///<field name="Price" type="Number">(public)Cost in dollars</field>

var key;
Epic.Training.Example.Web.Cheese.initializeBase(this);
// If the reference object has been passed, copy
// properties over that are defined in the prototype of this class
if (referenceObject) // Check for null/undefined
{
    for (key in referenceObject) // Iterate over properties
    {
        if (referenceObject.hasOwnProperty(key) && typeof (this[key]) !==
"undefined")
        {
            this[key] = referenceObject[key]; // Copy the value over
        }
    }
}
};

Epic.Training.Example.Web.Cheese.prototype = {
//#region fields

Name: "New Cheese",
Item: -1,
ImagePathSmall: "./images/GenericCheese-Thumb.png",
ImagePathLarge: "",
Weight: 0,
Description: "Fresh!",
Price: 0

//#endregion
};

Epic.Training.Example.Web.Cheese.registerClass(
"Epic.Training.Example.Web.Cheese");

```

12. Examine the code that was placed in the constructor from the class snippet. Notice that it is iterating through the reference object and copying over property values that have matching fields in this class.

```

// If the reference object has been passed, copy
// properties over that are defined in the prototype of this class
if (referenceObject) { // Check for null/undefined
    for (key in referenceObject) { // Iterate over properties
        if (referenceObject.hasOwnProperty(key) &&
            typeof(this[key]) !== "undefined") {
            this[key] = referenceObject[key]; // Copy the value over
    }
}

```

```
    }
}
}
```

Note that this copy technique is not "deep". If the fields of your class include a collection, then the *reference* to the collection is copied. This means that if you remove entries from the collection in the copied object, then they would be removed from the corresponding collection in all duplicate instances as well (and vice versa).

If your intention during the copy is to create a distinct collection (for example, an array) in the field, then you would need to create a new array and copy each element over in a loop:

```
this.TheCollectionField = [];
var collectionToCopy = referenceObject.TheCollectionField;
for (var ln=0; ln < collectionToCopy.length; ln++)
{
    this.TheCollectionField[ln] = collectionToCopy[ln];
}
```

This isn't a problem for the in-class example, because the cheese doesn't contain any collections, but if you were to duplicate this for other scenarios then it might end up causing problems.

## Part 4: Populate the inventory list

1. In *CatalogBehavior*, create a method called *onDataLoaded* that takes one parameter, *cheeses* using the *instmethod* snippet.
  - The parameter *cheeses* will be the array of cheese objects that you added to *Shared.js*.
2. Write a loop that will:
  - Create a new cheese object for each cheese in the array
  - Add the new cheese object to grid inventory

```
//#region event handlers
onDataLoaded: function Example$CatalogBehavior$onDataLoaded(cheeses)
{
    ///<summary>Event handler called when data is
    ///  ready to be displayed</summary>
    ///<param name="cheeses" type="Array">An array
    ///  of Cheese objects</param>
    var ln, newCheese;
    for (ln = 0; ln < cheeses.length; ln++) {
        newCheese = new Epic.Training.Example.Web.Cheese(cheeses[ln]);
        this.__gridInventory.addRow(newCheese);
    }
}
```

```
},
//#endregion
```

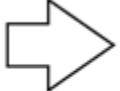
- In *onLoad*, call *GetCheeses*, and pass a delegate pointing to *onDataLoaded* (be sure to update the JSHint comments as necessary):

```
onLoad: function ExampleCatalogBehavior$onLoad() {
    ///<summary>Called by the constructor after "onDataPreload" and before
    /// "onLoaded"</summary>
    var loadHandler = $$fcd(this, this.onDataLoaded);
    this.__shared.GetCheeses(loadHandler);
},
```

- Save and run your site, then navigate to *Catalog.aspx*.
  - At this point, the table should populate correctly, but images are not being displayed.
  - Also, the *Details* link doesn't point to the details page
- To display the images, change the type of column definition for the images from *etcw:Column* to *etcw:ImageColumn* in *Catalog.aspx*:

```
<etcw:ImageColumn Header="Image" BoundProperty="ImagePathSmall"
    AlternateText="Thumbnail of cheese" />
```

- Add the *CssClass* attribute with a value of "dataTable" to the grid so that the same styles used for the contact information on the customer service page are applied to *gridInventory*.
- Save and run your page. It should appear similar to the following:



Catalog		
Name	Image	Options
Blue	./images/Blue-Thumb.png	<a href="#">Details</a>
Cheddar	./images/Cheddar-Thumb.png	<a href="#">Details</a>
Swiss	./images/Swiss-Thumb.png	<a href="#">Details</a>

Catalog		
Name	Image	Options
Blue		<a href="#">Details</a>
Cheddar		<a href="#">Details</a>
Swiss		<a href="#">Details</a>

New look of *gridInventory*

## Exercise: Make the Cheeses Editable

It is not sufficient to just list the cheeses currently in the inventory. Your employer would also like to be able to edit existing cheeses and add new cheeses. Fortunately the *CollectionGrid* supports the addition of editable columns.

1. In *CatalogBehavior*, add a new private field called *\_curEditCheese* that is initially *null*. This field represents the cheese that is currently being edited.
  - *\_curEditCheese: null,*

```
Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior = function
Example$CatalogBehavior(clientId)
{
  ///<field name="__shared"
  ///      type="Epic.Training.Example.Web.Shared">
  /// (private) Shortcut to the shared library</field>
  /// <field name="__gridInventory"
  ///      type="Epic.Training.Core.Controls.Grid.Web.CollectionGrid">
  /// (private) Grid to display the inventory</field>
  ///<field name="__curEditCheese" type="Epic.Training.Example.Web.Cheese">
  ///(private)The cheese currently being edited</field>

Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.initializeBase(
  this, [clientId]);
};

Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.prototype = {
  //#region fields
  __gridInventory: null,
  __shared: Epic.Training.Example.Web.Shared,
  __curEditCheese: null,
  //endregion
}
```

2. Create a private method called *\_validateAndSaveCheese* that will eventually be used to save modified cheeses to the server:

```
//#region private methods
__validateAndSaveCheese: function Example$CatalogBehavior$__validateAndSaveCheese ()
{
  /// <summary>Save cheese to the database then
  /// release edit mode</summary>
  /// <returns type="Boolean" >"true" if successful,
  /// "false" otherwise</returns>
  if (this.__curEditCheese !== null) {
    // TODO: Eventually call web service here to save cheese.
    // TODO: Validate the cheese
    this.__gridInventory.set_rowReadOnly(
      this.__curEditCheese, true);
    this.__curEditCheese = null;
  }
  return true;
}
```

```
},
//#endregion
```

3. Create a new event handler in *CatalogBehavior* called *\_gridInventoryDetailsClick*. Have this method navigate to the details page, including the cheese item number in the query string

```
_gridInventoryDetailsClick: function
Example$CatalogBehavior$__gridInventoryDetailsClick(event, cheese)
{
    ///<summary>(private) Click event handler of the
    /// gridInventory row option Details</summary>
    /// <param name="event" type="Object">
    ///   The click event argument</param>
    /// <param name="cheese" type="Epic.Training.Example.Web.Cheese">
    ///   Cheese object of the clicked row</param>
    if (this._validateAndSaveCheese()) {
        document.location.href = "./Details.aspx?item=" + cheese.Item;
    }
},
```

4. In *onLoaded*, add a delegate to the *Details* row option, after initializing *this.\_gridInventory*:

```
this._gridInventory = this.getCtl("gridInventory");

var detailsDelegate = $$fcd(this, this._gridInventoryDetailsClick);
this._gridInventory.addRowOptionHandler("Details", detailsDelegate);
```

5. Run the web application, navigate to the inventory page and test the details link of each cheese. The details page should remain the same for each, other than the item number displayed in the address bar. For example, *Swiss* should show:
- *http://localhost:<port#>/Inventory/Details.aspx?item=320*
6. The collection grid supports in-cell editing of data using standard HTML form elements. For example, in *Catalog.aspx*:
- a. Change the name column definition to an *etcw>EditColumn*,
  - b. Add the attribute *Type="Text"*
7. Run your site and navigate to the inventory page. Notice that the name of each cheese is now in an input element of type text.
8. Add a new column (after the image column) for *Description*. This column should be of type *TextArea*:

```
<etcw:EditColumn Header="Description" BoundProperty="Description"
Type="TextArea" Rows="2" Cols="30" />
```

9. To be able to select the image, use an edit column of type Select and provide options for the various images that are available:

```
<etcw:EditColumn Header="Image" BoundProperty="ImagePathSmall"
Type="Select" DisplayType="Image" AlternateText="Thumbnail of cheese" >
<SelectOptions>
    <etcw:SelectOption DisplayText="American"
        Value="./images/American-Thumb.png" />
    <etcw:SelectOption DisplayText="Blue"
        Value="./images/Blue-Thumb.png" />
    <etcw:SelectOption DisplayText="Cheddar"
        Value="./images/Cheddar-Thumb.png" />
    <etcw:SelectOption DisplayText="Gouda"
        Value="./images/Gouda-Thumb.png" />
    <etcw:SelectOption DisplayText="Mozzarella"
        Value="./images/Mozzarella-Thumb.png" />
    <etcw:SelectOption DisplayText="Swiss"
        Value="./images/Swiss-Thumb.png" />
    <etcw:SelectOption DisplayText="Other"
        Value="./images/GenericCheese-Thumb.png" />
</SelectOptions>
</etcw:EditColumn>
```

The *DisplayType* attribute indicates how the data is presented when the row is in read-only mode. The default is *Text*.

10. Run your application and navigate to the inventory page. Notice that because all rows are in edit mode, you cannot see any of the thumbnails.
11. Set the *ReadOnly* attribute to true for the entire grid:

```
<etcw:CollectionGrid ID="gridInventory" runat="server"
ShowHeaders="true" CssClass="dataTable"
ReadOnly="true">
```

12. Next, add two new row options, *Edit* and *Done*. These will allow you to enable a single row at a time once the correct JavaScript is attached to them. Also add a *RowOptionDisplayStyle* to the Details option:

```
<RowOptions>
    <etcw:Option Name="Edit" ToolTip="Edit this cheese in the table"
```

```

    RowOptionDisplayMode="IsReadOnly" />
<etcw:Option Name="Done" ToolTip="Done editing this cheese"
    RowOptionDisplayMode="NotReadOnly" />
<etcw:Option Name="Details" RowOptionDisplayMode="IsReadOnly"
    ToolTip="View or edit details about this cheese" />
</RowOptions>

```

The attribute *RowOptionsDisplayMode* allows you to control when an option is available for a particular row. The choices are *IsReadOnly*, *NotReadOnly* and *Both* (the default).

13. Add the ability to insert new cheeses into the table. To do so, you will use a *TableOption*:

```

...
<TableOptions>
    <etcw:Option Name="Add Cheese"
        ToolTip="Add a new cheese to the inventory" />
</TableOptions>
</etcw:CollectionGrid>

```

14. Run your site and view the inventory page. Verify that you see two options on each row, **Edit** and **Details**, and an option at the end of the table to add a cheese.
15. Within the private methods region, add a method called `_editCheese` that accepts a cheese object. This method will validate and save the previous cheese and then change the row of the new cheese to edit mode.

```

//#region private methods
_editCheese: function Example$CatalogBehavior$__editCheese(cheese)
{
    ///<summary>(private) Set the cheese being edited to a
    /// different cheese</summary>
    ///<param name="cheese" type="Epic.Training.Example.Web.Cheese">
    /// The new cheese to edit</param>
    if (this.__curEditCheese !== cheese) {
        if (this.__validateAndSaveCheese()) {
            this.__gridInventory.set_rowReadOnly(cheese, false);
            this.__curEditCheese = cheese; //Set the new cheese being edited
            this.__gridInventory.select(cheese); //Set focus
        }
    }
},
...

```

16. Add event handlers for *Edit* and *Done* row options and the *Add Cheese* table option within the event handlers region. Initially, just show an alert:

```
__gridInventoryEditClick: function
Example$CatalogBehavior$__gridInventoryEditClick(event, cheese)
{
    alert("Edit clicked");
},
__gridInventoryDoneClick: function
Example$CatalogBehavior$__gridInventoryDoneClick(event, cheese)
{
    alert("Done clicked");
},
__gridInventoryAddCheeseClick: function
Example$CatalogBehavior$__gridInventoryAddCheeseClick(event)
{
    alert("Add clicked");
},
```

17. In *onLoaded*, connect the events to the various options:

```
onLoaded: function Example$CatalogBehavior$onLoaded() {
    ///<summary>Called by the constructor after "onLoad".</summary>
    this.__gridInventory = this.getCtl("gridInventory");
    var detailsDelegate = $$fcd(this, this.__gridInventoryDetailsClick);
    this.__gridInventory.addRowOptionHandler("Details", detailsDelegate);

    var editDelegate = $$fcd(this, this.__gridInventoryEditClick);
    this.__gridInventory.addRowOptionHandler("Edit", editDelegate);

    var doneDelegate = $$fcd(this, this.__gridInventoryDoneClick);
    this.__gridInventory.addRowOptionHandler("Done", doneDelegate);

    var addDelegate = $$fcd(this, this.__gridInventoryAddCheeseClick);
    this.__gridInventory.addTableOptionHandler("Add Cheese", addDelegate);
},
```



Event handlers for row options need to be connected before data is inserted into the table, otherwise the row-option links will not have the corresponding event handlers connected to their click events.

Since `onDataLoaded` is delayed until after `onLoaded` by the call to `setTimeout` in the simulated web service call, this order is guaranteed.

18. Run your site and verify that the event handlers are correctly attached to the new options of the table.
  - You will not be able to verify that *Done* works until the edit mode actually switches.
19. In `_gridInventoryEditClick`, edit the selected cheese:

```
_gridInventoryEditClick:
function Example$CatalogBehavior$__gridInventoryEditClick(event, cheese)
{
    ///<summary>(private) Handle the click event of the
    /// Edit row option</summary>
    /// <param name="event" type="Object">The click event</param>
    /// <param name="cheese" type="Epic.Training.Example.Web.Cheese">
    /// Cheese corresponding to the clicked row</param>
    this.__editCheese(cheese);
},
```

20. Run your site and verify that the edit mode switches. You will also be able to verify that the *Done* event handler is attached.
21. Change `_gridInventoryDoneClick` so that the selected cheese row is set to read only:

```
_gridInventoryDoneClick: function
Example$CatalogBehavior$__gridInventoryDoneClick(event, cheese) {
    /// <summary>(private) Handle the click event of the Done
    /// row option</summary>
    /// <param name="event" type="Object">The click even argument</param>
    /// <param name="cheese" type="Epic.Training.Example.Web.Cheese">
    /// Cheese object of the clicked row</param>
    if (this.__curEditCheese === cheese) {
        this.__validateAndSaveCheese();
        this.__gridInventory.select(cheese);
    }
    else {
        throw new Error(
            "No cheese is currently being edited. Done should be hidden.");
    }
},
```

22. Run your site and verify that *Edit* and *Done* toggle between edit modes.
23. Change `_gridInventoryAddCheeseClick` to insert a new Cheese into the grid:

```
_gridInventoryAddCheeseClick:  
function Example$CatalogBehavior$ _gridInventoryAddCheeseClick(event) {  
    ///<summary>Handle the click event of the Add Cheese table  
    /// option in gridInventory</summary>  
    ///<param name="event" type="Object">The click event object</param>  
    if (this._validateAndSaveCheese()) {  
        var newCheese = new Epic.Training.Example.Web.Cheese();  
        this._gridInventory.addRow(newCheese);  
        this._editCheese(newCheese);  
    }  
},
```

24. Verify that you are able to add new cheeses.
25. Continue working on the if you have time steps.

### If you have time

---

You may have noticed that when switching between edit and read-only mode, the table shifts. You can use styling to avoid this.

1. Add a new style sheet named *Catalog.css* to the Inventory folder. Make sure that this style sheet only loads for *Catalog.aspx* (Examine the *Details* page for an example of how this is done).
2. Add the following styles to *Catalog.css*. Be sure to apply column-specific classes to the appropriate column of *gridInventory*:

```
/* Classes for the name column of gridInventory */  
  
.invTable  
{  
    width: 100%;  
}  
  
.invNameCol  
{  
    width:150px;  
}  
.invNameCol input  
{  
    width:95%;  
}  
/* Classes for the description column of gridInventory */  
.invDescriptionCol  
{
```

```
width:225px;
}
.invDescriptionCol textarea
{
    width:95%;
}
/* Classes for the image column of gridInventory */
.invImageCol
{
    width: 100px;
    text-align: center;
}
.invImageCol select
{
    width: 95%;
}
/* Class automatically added to the row-options column */
.cgRowOptions
{
    width:120px;
}
/* Class automatically added to all rows bound to an object */
.cgData
{
    min-height: 60px;
}
```

# Lesson 5: Implementing Business Logic

<b>The Big Picture.....</b>	<b>5•3</b>
Traditional Paradigm (Post-Back) .....	5•3
New Paradigm (Web-Service).....	5•4
Asynchronous Programming.....	5•5
Response Order.....	5•6
Avoid Overlapping Web Service Calls.....	5•6
Exploring Real-World Scenarios.....	5•7
Chart Review .....	5•7
Combining Web Services.....	5•7
Not Combining Web Services.....	5•8
Avoiding Unnecessary Web Service Calls.....	5•8
<b>Communicating with the Server.....</b>	<b>5•9</b>
Load the Full Cheese Inventory .....	5•9
Creating Business Objects.....	5•9
Exercise: Load Inventory via Web Service.....	5•11
Part 1: Get the data file and Business Objects.....	5•11
Part 2: Create the Web Service .....	5•13
Part 3: Prepare the Cheese class for web-service serialization.....	5•17
Part 4: Populate the table from the server .....	5•18
Part 5: Overlapping Web Services .....	5•21
Part 6: Save new and modified cheeses.....	5•24
Part 7: Loading the Details Page.....	5•30
Exercise: Send the E-mail.....	5•34
Part 1: Create the Email Class .....	5•34
Part 2: Create the EmailList Class .....	5•35
Part 3: Create the Web Service .....	5•37
If you have time .....	5•40
Wrap Up.....	5•41
Troubleshooting Web Services.....	5•42
<b>Communicating via Post Back.....</b>	<b>5•43</b>
Post Back .....	5•43
Server Events.....	5•43
Exercise: Exploring Server Events and Post Back.....	5•44
Part 1: Working with Server Events.....	5•45

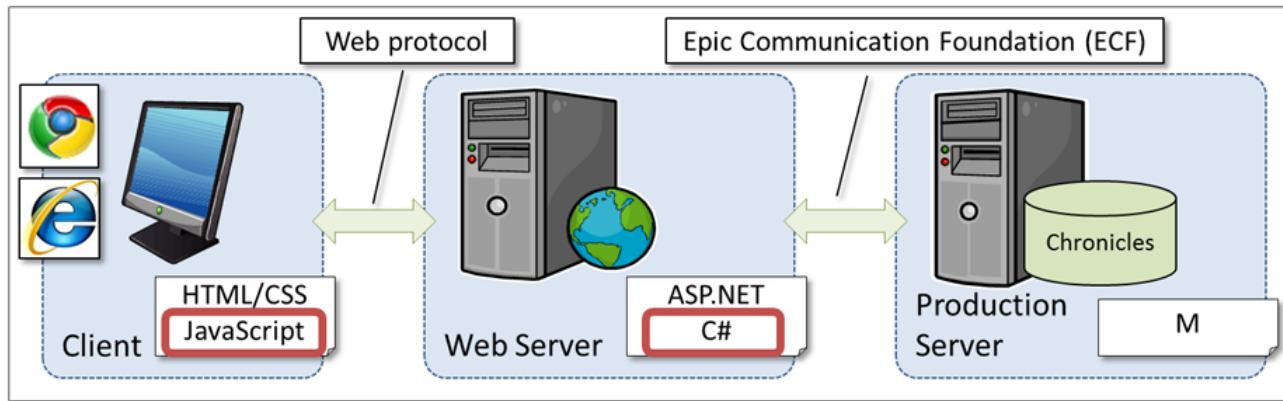
Part 2: Observe Network Traffic .....	5•46
Part 3: Reduce Post-Back Traffic.....	5•48
Part 4: Reduce Traffic from Shipping-Cost Calculation .....	5•49
If you have time.....	5•51
Wrap Up.....	5•51
<b>Implementing User Security.....</b>	<b>5•53</b>
Options Vary by User .....	5•53
Authentication .....	5•53
Authorization.....	5•54
Exercise: Implement User Security.....	5•54
Part 1: Setup Forms Authentication .....	5•54
Part 2: Make the details page editable .....	5•57
Part 3: Create a Log-out Link.....	5•58
Part 4: The Session Variable.....	5•59
Part 5: Create the CustomRoleProvider Class.....	5•61
Part 6: Activate & Use the RoleProvider .....	5•61
If you have time.....	5•63
Wrap Up.....	5•64
<b>Managing State.....</b>	<b>5•66</b>
State in VB vs. Web .....	5•66
Web Gardens and Farms.....	5•68
Exercise: Maintaining State .....	5•69
Part 1: Using session-specific state on the client .....	5•69
Part 2: Using session-specific state on the server .....	5•72
Part 3: Using application-wide state on the server .....	5•72
Wrap Up.....	5•74

# Implementing Business Logic

## The Big Picture

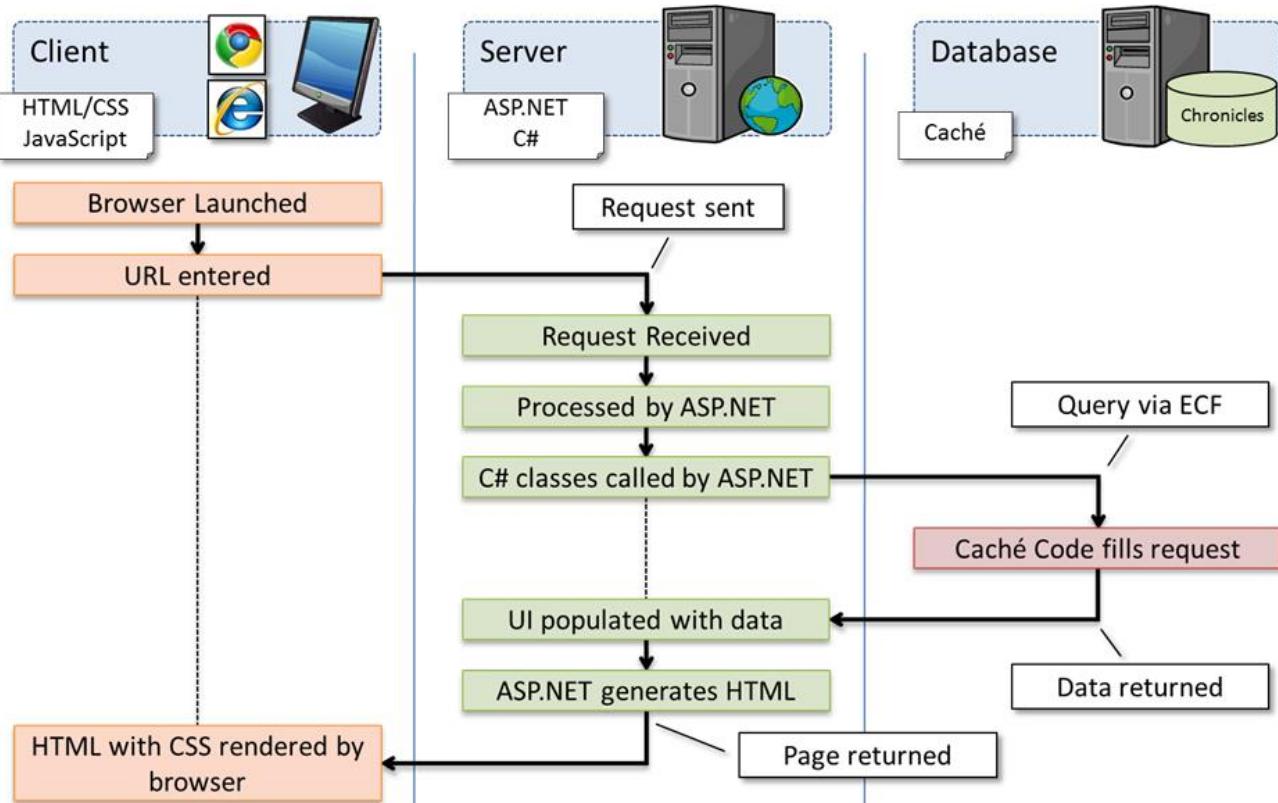
Business objects can be written for the client or the server, but only the server-side is fully secure. Additionally, it is the server that has direct access to the database.

In this lesson, you will learn about asynchronous server communication to link the client and server together, and will have an opportunity to create several server business objects to prepare for communication with the database. At this point, database communication will be simulated with XML files on the server.



*Location of Business Objects*

## Traditional Paradigm (Post-Back)



Recall the traditional paradigm used for communicating with the web server. It is based on post-back, which occurs when the current page posts information back to itself in order to do something on the server. Things that it might do include:

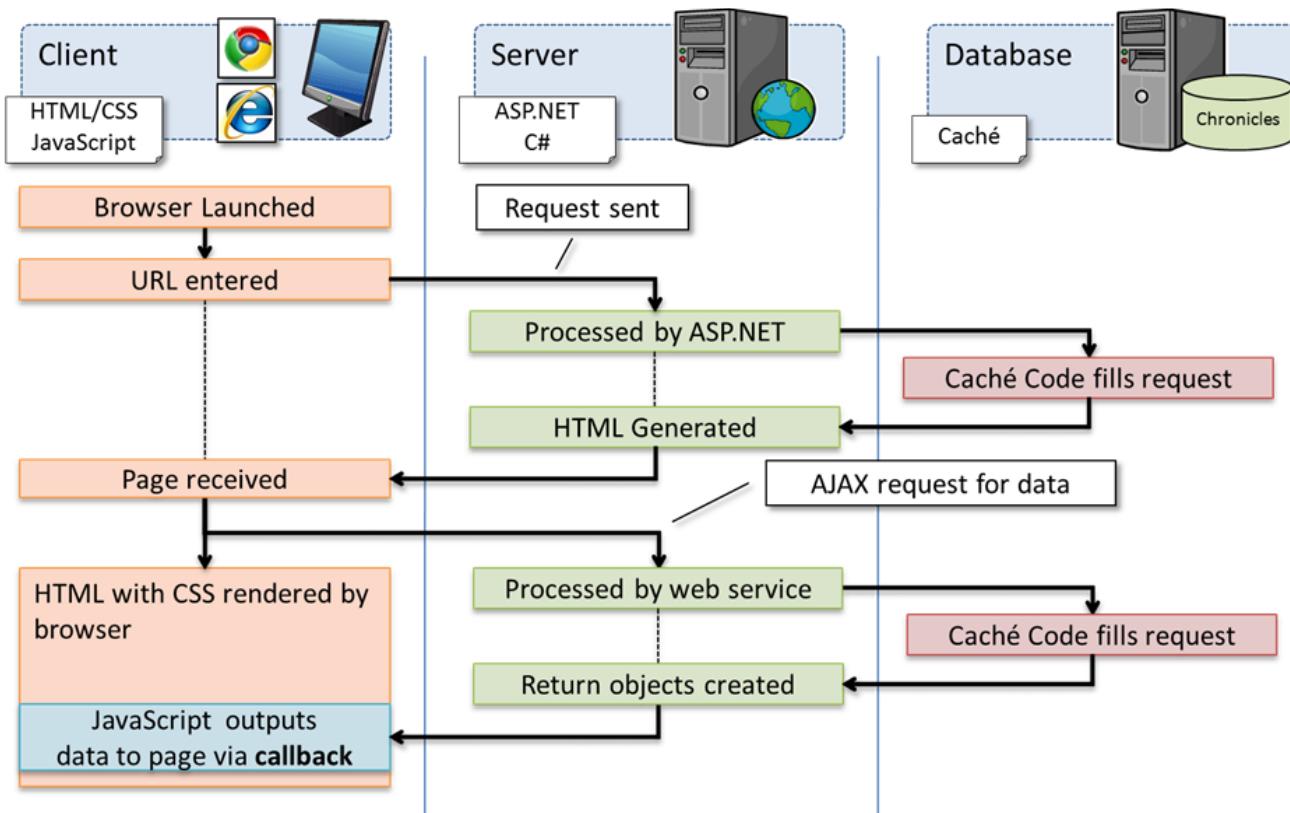
- Process a server event wired up in ASP.NET
- Load data from the Database or an XML file
- Access secure information

A key disadvantage of the traditional paradigm is that post-back is synchronous, so the client (and by extension, the user) must wait until the request resolves until another action can be taken.

Another disadvantage is increased network traffic and server workload resulting from the entire page being re-rendered.

## New Paradigm (Web-Service)

---



Next recall the new web-service paradigm. Although a complete load occurs on the initial request, subsequent communication with the web server uses asynchronous web-service calls instead of post back. This has the following benefits:

- The client can continue working while waiting on the response.
- The entire page is not reprocessed, which reduces server workload
- The process page is not sent back to the client, which reduces network traffic.

The disadvantages are more processing on the client (in JavaScript). The tradeoff seems reasonable since even modest client machines tend to have unutilized processing power in the traditional paradigm.

## Asynchronous Programming

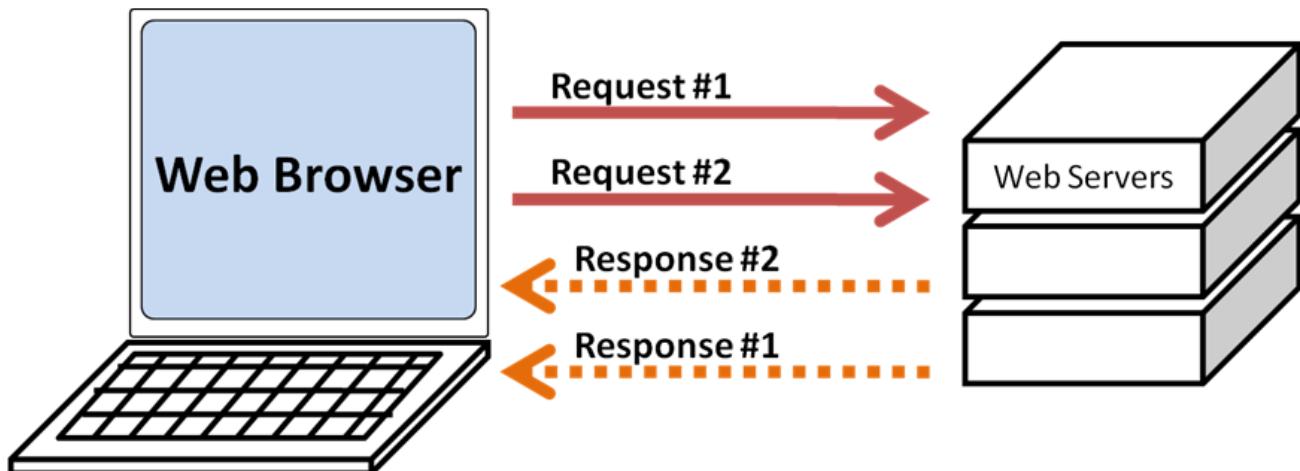
Examine the following pseudo-code segment. Based on the comments, determine if the code *doSomethingElse* will work properly. If not, modify the code so that it executes the way that it should.

```
onLoad()
{
    WebServiceCall(callOnReturn, callOnError);
    doSomething(); //Independent of web service
```

```
doSomethingElse(); //Must run after web service returns  
}  
  
callOnReturn(result)  
{  
    loadWebServiceResult(result);  
  
}
```

## Response Order

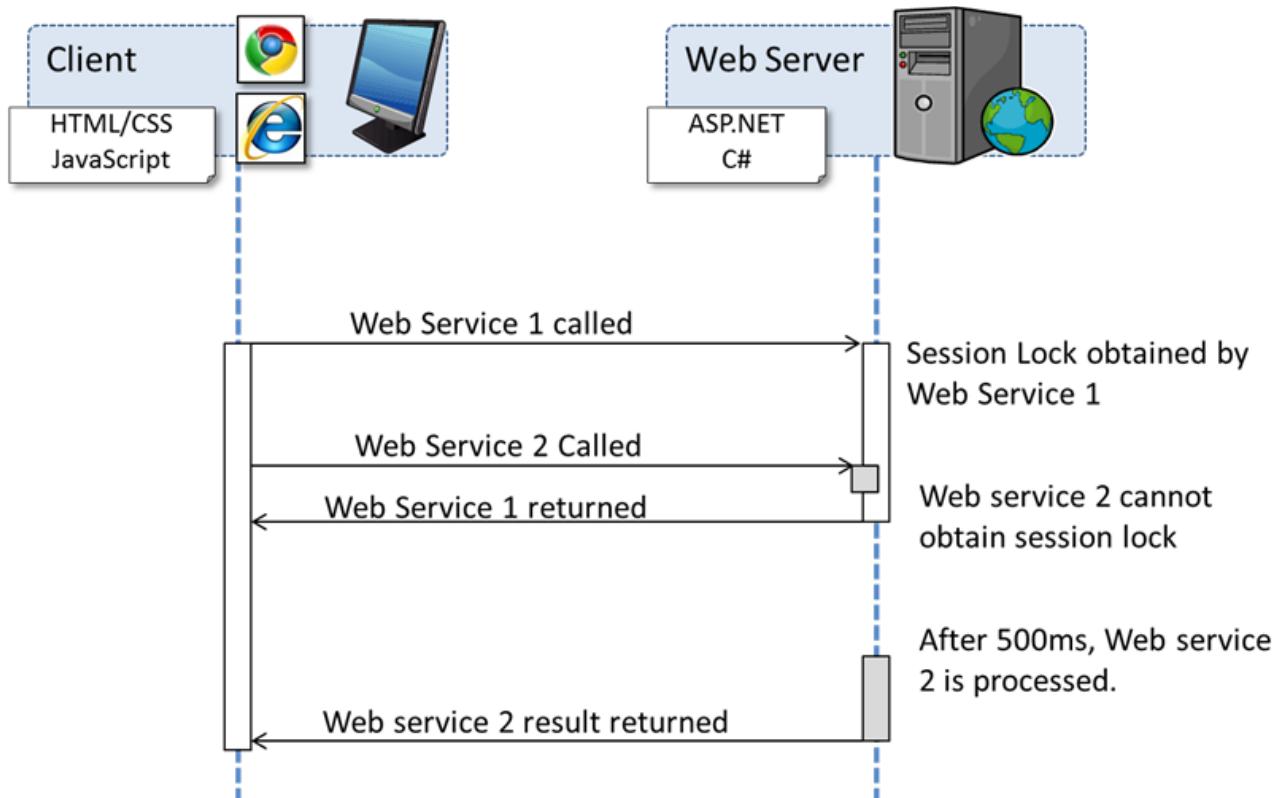
Asynchronous requests sent to the server may not be responded to in the order they are sent, depending on network conditions and how the message was routed.



*Asynchronous requests can be responded to out of order*

## Avoid Overlapping Web Service Calls

When web service calls invoked within the same session (i.e., user connection) overlap, the call that arrives second will automatically be delayed by 500ms. This is because the call that arrives first will obtain a lock on the session, which forces the later call to wait.



**Avoid overlapping web service calls to prevent session lock contention.**

If you need to call multiple web services, it might be better to call the second web service after the first one finishes. If possible, it would be even better to combine the two web services into one.

## Exploring Real-World Scenarios

In this section, we'll explore some web service factoring problems that occurred during Epic development. In some cases, it made sense to refactor web services to combine them. In other cases, doing so would break encapsulation of distinct components, so the decision was made to explicitly keep the web services as separate calls, even though they could be invoked simultaneously.

### Chart Review

The web version of Chart review has several interesting scenarios related to effective web service factoring.

### Combining Web Services

Previously, when a row was selected, the availability of some of the toolbar buttons (like bookmark and launch encounter) was looked up via a web service. At the same time, if the preview pane was

open, selecting a row will also trigger another web service to load the preview report. The optimization was to piggyback looking up toolbar-button availability with the web service that renders the report.

## Not Combining Web Services

Due to component encapsulation and complex code paths, it isn't always a good idea to combine web services. One example is the Refresh workflow from Chart Review.

When the **Refresh** button is clicked, one of two different web services is called, either ReloadData or EnumerateRows. The return handler for both web services calls filters.reload() if the filters control has already been loaded. The filters control in turn calls the ReloadFilters web service to make sure the filters control is reset properly.

Because the Filters control is separate from Chart Review, we don't want to break encapsulation by creating the combination web services ReloadDataAndFilters and EnumerateRowsAndReloadFilters, just to make things slightly faster. Additionally, it would have been tricky to combine the calls because of the multiple code paths.

## Avoiding Unnecessary Web Service Calls

In Chart Review, duplicate web service calls were invoked when default filters were loaded as the page was launched. The UpdateDescription web service, which produces a text description of the currently applied filters, is first called when the default filter is loaded. Before the filter's model is given an opportunity to set the loaded filter choices into the data model, the FilterControl attempts to apply the default filters, which triggered an additional UpdateDescription call requesting the same data. Since the code paths that generated the repeated calls to UpdateDescription were separate, it wasn't obvious that something bad was happening before examining Hyperspace Instrumentation data. In fact, both callers do need to call UpdateDescription in some cases, just not this one.

The fix to this was to create a flag for this case to indicate that the default filters are being loaded. This way the second UpdateDescription call is skipped if the flag is already set.

# Communicating with the Server

In this section, you will get the business objects needed to load cheeses from an XML file and send them to the client in response to an asynchronous request.

## Load the Full Cheese Inventory



Catalog			
Name	Image	Description	Options
Blue		Strong blue cheese flavor running throughout	<a href="#">Edit Details</a>
Cheddar		A very mild cheddar flavor.	<a href="#">Edit Details</a>
Swiss		Mellow flavor with many holes	<a href="#">Edit Details</a>
<a href="#">Add Cheese</a>			

Catalog			
Name	Image	Description	Options
Blue		Strong blue cheese flavor running throughout	<a href="#">Edit Details</a>
Cheddar		A very mild cheddar flavor.	<a href="#">Edit Details</a>
Swiss		Mellow flavor with many holes	<a href="#">Edit Details</a>
American		Medium cheddar flavor	<a href="#">Edit Details</a>
Feta		Creamy white cheese	<a href="#">Edit Details</a>
Gouda		Rich, nutty flavor	<a href="#">Edit Details</a>
Mozzarella		White cheese with small holes	<a href="#">Edit Details</a>
<a href="#">Add Cheese</a>			

The gridInventory after loading all cheeses

Ultimately, all data of this kind will be loaded from Chronicles. For now it will be loaded from an XML file.

## Creating Business Objects

For now, XML serialization will be used to save and load cheese objects to a flat file. XML serialization uses C# attributes to indicate which properties are included, and how they are serialized, either as *elements* or *attributes*. The following XML serialization attributes can be added to properties in your business objects.

XmlAttribute	Use this attribute on a property to indicate that it is an XML attribute. The value of the attribute name will be the name of the XML attribute. The value of the type attribute will be the type of the XML attribute.
XmlRoot	Use this attribute on a class to indicate that it is the root of an XML file. The element name will match the class name. The namespace value corresponds to the value of the <code>xmlns</code> (XML Namespace) attribute. For example, this:  <code>[XmlRoot (Namespace="Epic.Training.Example")]</code>

```
public class CheeseList {...}
```

Will render as:

```
<CheeseList xmlns="Epic.Training.Example" ... >
  ...
</CheeseList>
```

#### XmlAttribute

Indicates that the property of a class will render as the attribute of an element in an XML file. For example:

```
public class Cheese
{
  [XmlAttribute]
  public string Name { get; set; }

  [XmlAttribute]
  public int Item { get; set; }
  ...
}
```

Will render as:

```
<Cheese Name="..." Item="..." ... />
```

*XmlAttribute* is appropriate for simple properties that only have one value. For collections, use *XmlElement*.

#### XmlElement

If no XML attribute is applied to a property, *XmlElement* is used by default.

This attribute Indicates that the property of a class will serialize to a nested XML element. For example:

```
[XmlRoot(Namespace="Epic.Training.Example")]
public class CheeseList
{
  ...
  [XmlElement("Cheese")]
  public List<Cheese> Cheeses { get; set; }
  ...
}
```

Will serialize as the following if there are two cheeses (220-Cheddar and 320-Swiss) in the list:

```
<CheeseList xmlns="Epic.Training.Example">
    <Cheese Item="220" Name="Cheddar" .../>
    <Cheese Item="320" Name="Swiss" .../>
</CheeseList>
```

The argument to `XmlElement` indicates the name of the element used. By default the property name is used, but for collections it makes sense to change the name to a singular form, because the element will be repeated to indicate multiple objects in the collection.

`XmlAttribute`

Use to indicate that a property should not be serialized to XML. By default, a property is serialized as an element. For example:

```
[XmlAttribute]
public PropertyToIgnore {get; set;}
```



Normally, the XML namespace is a uniform resource identifier (URI). The two URI alternatives are a uniform resource locator (URL) or uniform resource name (URN). Both alternatives uniquely identify some entity (in this case the namespace of an XML file), but the use of a URL implies that the resource is available at that location, while the URN does not imply resource availability. Think of the URL as someone's address and the URN as that person's name. You can go to address to find the person, but you can't go to their name. The in-class example will use a URN to identify the namespace of XML files. The syntax of a URN is:

```
urn:<NID>:<NSS>
```

The namespace identifier (NID) indicates the type of the namespace, while the namespace-specific string (NSS) uniquely identifies the entity within the namespace type. To be 100% compliant with this standard, the URN used for the XML namespace for the in-class example should be something like `urn:.NET:Epic.Training.Example` to indicate that we are using a .NET namespace to identify a corresponding XML namespace, and the .NET namespace is `Epic.Training.Example`. However, for brevity, the in-class example uses only the namespace-specific string (`Epic.Training.Example`).

## Exercise: Load Inventory via Web Service

The goal of this activity is to create a web service to load the cheese list from a file and save changes to the list.

### Part 1: Get the data file and Business Objects

1. Add the following as existing items to the `Training.Example.Web.Pages > App_Data` folder:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\App\_Data\Cheeses.xml
- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\App\_Data\Cheeses.xsd



An XML Schema Definition (XSD) is similar to a DTD in that it defines the structure of an XML file, but is considered to be more flexible.

For more information, see: <http://www.w3schools.com/schema/default.asp>

2. Open *Cheeses.xml* and try adding a new cheese of your own. Notice that because the *Cheeses.xsd* is providing the XML structure, you will receive IntelliSense help on the available elements and attributes. For example, you could enter:

```
<Cheese Name="My New Cheese" Item="999"
ImagePathSmall="./images/GenericCheese-Thumb.png"
ImagePathLarge="./images/GenericCheese-Full.png" Weight="16"
Description="Only the BEST cheese ever..." Price="100" />
```

3. Create the folder:

- C:\EpicSource\WTC\Code\Training\Core\Model\

4. Copy the folder:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\projects\Code\Xml

5. Paste the folder into the new folder that you just created:

- C:\EpicSource\WTC\Code\Training\Core\Model\

This is the *Training.Core.Xml* project. It contains a class called *XmlLoader* that simplifies the use of *System.Xml.Serialization*. Feel free to use it in your lab project as well as the class example.

6. Create the folder:

- C:\EpicSource\WTC\Code\Training\Example

7. Copy the folder:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\projects\Code\Model

8. Paste the folder into the new folder you just created:

- C:\EpicSource\WTC\Code\Training\Example\

This is the *Training.Example* project. It contains business objects for the *Big Mouse Cheese Factory* example. Unlike the project *Training.Example.Web*, which you already had in your solution, these business objects are written in C#, so they are not dependent on the web platform.



Projects that contain *Example* in the namespace should not be reused in your *Quizzer* application.

9. Add both existing projects to your solution:
  - Code\Training\Core\Model\Xml\Training.Core.Xml.csproj
  - Code\Training\Example\Model\Base\Training.Example.csproj
10. Build your solution so the projects that you just added generate assemblies that you can reference.
11. Include a reference to the assemblies of the following projects in your *Pages* project:
  - *Epic.Training.Example*
  - *Epic.Training.Core.Xml*
12. Ensure that the *Pages* project also depends on the two that you just added so everything builds in the correct order.

## Part 2: Create the Web Service

---

In this part you will create the web service used to load cheese information from the server.

1. Examine *Web.config* from *Training.Example.Web.Pages*. It should be relatively simple:

```
<configuration>

    <system.web>
        <compilation debug="true" targetFramework="..." />
        <authentication mode="Windows" />
    </system.web>

    <system.webServer>
        <modules runAllManagedModulesForAllRequests="true"/>
    </system.webServer>

</configuration>
```

2. In *Training.Example.Web.Pages > Inventory*, add a new item of the type *AJAX-enabled WCF Service* called *CatalogBehavior.svc*:
  - *Inventory > Right click > add > new item > Web > WCF Service (Ajax -enabled) > Name: CatalogBehavior.svc*
3. Notice the additional configuration in *Web.config*:

```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior
        name="Epic.Training.Example.Web.Pages.Inventory.CatalogBehaviorAspNetAjaxBehavior">
        <enableWebScript />
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
    multipleSiteBindingsEnabled="true" />
  <services>
    <service name="Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior">
      <endpoint address=""
        behaviorConfiguration=
          "Epic.Training.Example.Web.Pages.Inventory.CatalogBehaviorAspNetAjaxBehavior"
        binding="webHttpBinding"
        contract="Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior" />
    </service>
  </services>
</system.serviceModel>
```

#### ! Renaming your service will not update Web.config

If you name the service incorrectly and need to rename it later, don't forget to update *Web.config* as well, as this doesn't happen automatically.

4. Open *CatalogBehavior.svc.cs*.
5. Set the *Namespace* argument of the *ServiceContract* class attribute to *Epic.Training.Example.Web.Pages.Inventory.Services*:

```
[ServiceContract (Name space="Epic.Training.Example.Web.Pages.Inventory.Services") ]
```

When a web service is made available in JavaScript, it is added as a class in the namespace specified in the *ServiceContract* attribute.

6. Open *Global.asax*.
  - This is where you can store web-specific properties that must be shared between pages.
7. Change the namespace from *Pages* to *Epic.Training.Example.Web.Pages*.
8. Right click on *Global.asax* in the solution explorer and select **View Markup**

9. Change the *Inherits* attribute from *Pages.Global* to *Epic.Training.Example.Web.Pages.Global*.
10. In *Global.asax.cs*, add a private, static, readonly, string field named *s\_AppDataPath*:

```
/// <summary>
/// The root location of all XML files
/// </summary>
private static readonly string s_AppDataPath =
HostingEnvironment.MapPath("~/App_Data/");
```

*HostingEnvironment.MapPath* is the method used by ASP.NET to resolve the ~ character.

11. Add a second private static readonly string field for the root cheese XML file name:

```
/// <summary>
/// Root file name of all cheese XML files
/// </summary>
private static readonly string s_CheeseFileNameRoot = s_AppDataPath +
"Cheeses";
```

12. Add a public string property named *CheeseFileName* that returns the correct XML file name based on the user's country:

```
/// <summary>
/// Root of cheese XML files
/// </summary>
public static string CheeseFileName {
    get
    {
        // Eventually you will get the country from the Session cache
        // based on the user's login information
        string country = "";
        return s_CheeseFileNameRoot + country + ".xml";
    }
}
```

13. Stop your development server, if it is running.



Any time you make changes to static variables, you should restart the development server so they can be reinitialized.

14. In *CatalogBehavior.svc.cs*, change *DoWork()* to *GetCheeses()*.
15. Have *GetCheeses* return a list of *Cheese* objects using the *XmlLoader* class:

```

/// <summary>Get the list of cheeses</summary>
/// <returns>Generic List of type Cheese</returns>
[OperationContract]
public List<Cheese> GetCheeses()
{
    List<Cheese> cheeses = null;
    try
    { // Ensure list is not modified while being read
        CheeseList.CheeseLock.EnterReadLock();
        String filename = Global.CheeseFileName;
        CheeseList cl = XmlLoader<CheeseList>.LoadItems(filename);
        cheeses = cl.Cheeses;
    }
    catch (Exception e)
    {
        throw new Exception(e.Message, e);
    }
    finally
    { // Done reading the list
        CheeseList.CheeseLock.ExitReadLock();
    }
    return cheeses; // Return the list
}

```

To handle concurrent accesses, *CheeseList* implements a *CheeseLock* property of the type *ReaderWriterLockSlim*. You will get a chance to define one of your own in the next exercise.

#### 16. Add a new service method called *AddOrReplaceCheese*:

```

/// <summary> Add a new cheese to the list for a given country,
/// or replace an existing cheese.</summary>
/// <param name="newCheese">The new cheese</param>
/// <returns>The item number of the cheese</returns>
[OperationContract]
public int AddOrReplaceCheese(Cheese newCheese)
{
    try
    { // Ensure 1 process can modify list at a time
        CheeseList.CheeseLock.EnterWriteLock();
        String filename = Global.CheeseFileName;
        CheeseList cl = XmlLoader<CheeseList>.LoadItems(filename);
        cl.AddOrReplaceCheese(newCheese);
        XmlLoader<CheeseList>.ReplaceList(filename, cl);
    }
    catch (Exception e)
    {

```

```
        throw new Exception(e.Message, e);
    }
    finally
    {
        CheeseList.CheeseLock.ExitWriteLock();
    }
    // Return the new item number so the client can be updated
    return newCheese.Item;
}
```

17. Why do you suppose locks are always released in the finally block?

■

---

18. Remove and sort usings:

- Edit > IntelliSense > Organize Usings > Remove and Sort

### Part 3: Prepare the Cheese class for web-service serialization

The Cheese class is setup to serialize to XML, but not to JavaScript via JavaScript Object Notation (JSON). The following changes will enable the cheese class for JSON serialization.

1. Verify that the .NET reference *System.Runtime.Serialization* is included in the *Training.Example* project.
  - This is the assembly required to use the attributes *[DataContract]* and *[DataMember]*.  
DataContract marks a class that you want to send to the client, and DataMember marks properties in the class that you want to send.
  - If there are any properties that you do not want to be sent to/from the client, use *[IgnoreDataMember]*
2. Open *Cheese.cs*.
3. Add the attribute *[DataContract]* to the *Cheese* class.
4. Add the attribute *[DataMember]* to each of the public properties of *Cheese*.
5. When finished, the code should look similar to the following:

```
using System;
using System.Xml.Serialization;
using System.Runtime.Serialization;

namespace Epic.Training.Example
{
    /// <summary>
```

```
/// Class used to represent cheese objects
/// </summary>
[DataContract]
public class Cheese
{
    [XmlAttribute,DataMember]
    public string Name { get; set; }

    [XmlAttribute,DataMember]
    public int Item { get; set; }

    [XmlAttribute,DataMember]
    public string ImagePathSmall { get; set; }

    [XmlAttribute,DataMember]
    public string ImagePathLarge { get; set; }

    [XmlAttribute,DataMember]
    public string Description { get; set; }

    [XmlAttribute,DataMember]
    public double Weight { get; set; }

    [XmlAttribute,DataMember]
    public double Price { get; set; }
}
```

## Part 4: Populate the table from the server

In this part you will setup *Catalog.aspx* to include *CatalogBehavior.svc*, and call it from *CatalogBehavior.js*.

1. Open *Inventory/Catalog.aspx*.
2. Add a *ScriptManagerProxy* just above the *PageSettings*.
3. Add a service reference in the script manager proxy on *Catalog.aspx*:

```
<asp:ScriptManagerProxy runat="server">
    <Services>
        <asp:ServiceReference Path="~/Inventory/CatalogBehavior.svc" />
    </Services>
</asp:ScriptManagerProxy>
```



Make sure that you include the service reference inside the *Services* element, not the *Scripts* element.



Now that the service is sent to the client, it is possible to view the JavaScript that was automatically generated for the service. If you want to see this JavaScript, do the following:

1. Save and run your web application.
2. Navigate to Catalog.aspx
3. Change the page name from **Catalog.aspx** to **CatalogBehavior.svc/js**
  - This will download the JavaScript file.
4. Open the JavaScript file in your favorite text editor. You will see a class representing the web service:

```
Type.registerNamespace('Epic.Training.Example.Web.Pages.Inventory.Services');

Epic.Training.Example.Web.Pages.Inventory.Services.CatalogBehavior=function() {
    ...
}

Epic.Training.Example.Web.Pages.Inventory.Services.CatalogBehavior.prototype={
    ...
    AddOrReplaceCheese:function (newCheese, succeededCallback,
        failedCallback, userContext) {
        ...
    }
    ...
};

if (typeof(Epic.Training.Example.Cheese) === 'undefined') {

    Epic.Training.Example.Cheese=gtc("Cheese:http://schemas.datacontract.org/
        2004/07/Epic.Training.Example");

    Epic.Training.Example.Cheese.registerClass('Epic.Training.Example.Cheese'
    );
}
```

5. To get IntelliSense help regarding the service, add a reference to *CatalogBehavior.svc* in *CatalogBehavior.js*:

```
///<reference path="CatalogBehavior.svc" />
```



- Ensure that the IntelliSense references appear at the top of the JS file and there are no newlines between them. Otherwise your IntelliSense will be broken.
- Also make sure that references to .svc files are listed last

6. To get easy access to the service add the field:

```
Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior = function
Example$CatalogBehavior(clientId)

{
    ...
    ///<field name="__service"
    type="Epic.Training.Example.Web.Pages.Inventory.Services.CatalogBehavior">
        /// The web service corresponding to this JavaScript class
    ///</field>

    Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.initializeBase(
        this, [clientId]);
}

Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.prototype = {
    //#region fields
    ...
    __service: null,
    //#endregion
}
```

7. Initialize the field to the service reference in *onDataPreload*:

```
onDataPreload: function Example$CatalogBehavior$onDataPreload() {
    ///<summary>Called by the constructor before "onLoad".</summary>
    this.__service =
        Epic.Training.Example.Web.Pages.Inventory.Services.CatalogBehavior;
},
```

8. Update IntelliSense (Ctrl + Shift + J).

9. In *onLoad*, comment out the *GetCheeses* call from *Shared*, and replace it with a call to *this.\_service.GetCheeses()*.

```
onLoad: function Example$CatalogBehavior$onLoad ()  
{  
    var loadHandler = $$fcd(this, this.onDataLoaded);  
    //this._shared.GetCheeses(loadHandler);  
    this._service.GetCheeses(loadHandler);  
},
```

10. Build and run your solution.
11. Navigate to the inventory page
12. You should see the full list of cheeses loaded.

## Part 5: Overlapping Web Services

Recall that overlapping web service calls can result in session lock contention. In this section you'll perform a simple experiment to illustrate this scenario.

1. In *CatalogBehavior.svc.cs*, add a simple web-service method:

```
[OperationContract]  
public void MyServiceMethod() { Thread.Sleep(200); }
```

2. Add a button to initiate the test, just above the collection grid containing the cheeses:

```
<button id="elServiceTest" type="button">Run service test</button>
```



Setting *type="button"* will prevent the element from posting back when clicked.

3. Add the new control to the *PageSettings > Managed Controls*:

```
<etcw:PageSettings runat="server"  
    ClientScriptPath="~/Inventory/CatalogBehavior.js"  
  
    ClientClass="Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior">  
  
    <ManagedControls>  
        <etcw:ControlEntry ControlID="elServiceTest" />
```

```
</ManagedControls>  
</etcw:PageSettings>
```

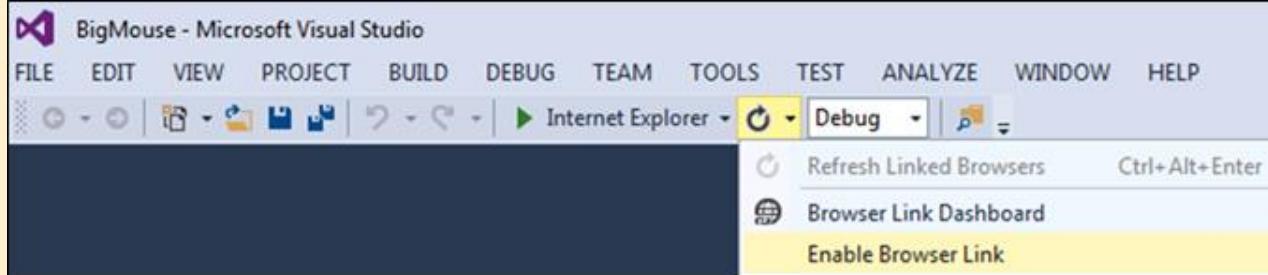
1. Connect a click event to elServiceTest in onLoad:

```
$addHandler(this.getEl("elServiceTest"), "click", $$fcd(this,  
this.__serviceTest));
```

2. Add the button-click event handler. In this handler, call MyServiceMethod twice:

```
_serviceTest: function Example$CatalogBehavior$__serviceTest()  
{  
    ///<summary>Run the web service overlap test</summary>  
    this.__service.MyServiceMethod();  
    this.__service.MyServiceMethod();  
},
```

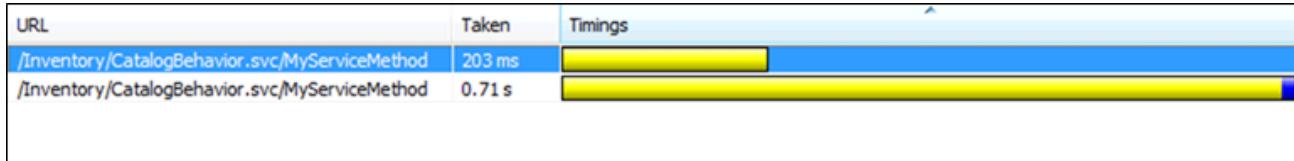
! If you are using Visual Studio 2013 or higher, then you may want to disable the browser link feature before continuing:



Otherwise, you'll notice unrelated network traffic.

3. Start the web application
4. Click **F12** to open the IE developer tools
5. Click the **Network** tab
6. Click **Start Capturing**
7. Navigate to the catalog page. Notice how IE is capturing network traffic.
8. Click **Clear** on the right side of the developer tools to clear the currently captured network traffic.
9. On the catalog page, click **Run service test**
10. Click **Stop Capturing** on the developer tools.

11. The result should look something like this:



*Overlapping web service calls, call 1 arrives first*

or this:



*Overlapping web service calls, call 2 arrives first*



Why does the later service call take 700 ms rather than just 200 ms?

---

---

---

12. Another interesting experiment is what happens when a large number of web services overlap.

- Change \_\_serviceTest so that MyServiceMethod is called 10 times instead of just 2, and then measure the network traffic.

13. Your measurements should look something like the following screenshot. Note that it could vary depending on which web services reach the server first:



*Results from 10 overlapping web services*



How many web services can be in flight at one time?

Look carefully at the results from the previous experiment. Notice that when a certain number of web services have been sent to the server, all subsequent web services are not sent until at least one returns.

Based on the results, how many services can be in flight at one time?

- \_\_\_\_\_



### Never call more than one web service per user action

As you observed from the previous experiments, sending overlapping web services can greatly increase the time required for the result to come back from the server, often in unpredictable ways. For this reason, Epic recommends not sending more than one asynchronous call per user action in the system.

14. To prevent the **Run service test** button from cluttering your UI, add the CSS class **hidden** to it in markup.

## Part 6: Save new and modified cheeses

Currently changes to the Cheese list are discarded. In this exercise you'll make it so changes are saved to the XML file.

1. Stop the web application.
2. Create a method that will take the item number returned from the server after a cheese is saved and assign it to the cheese that was saved:

```
_cheeseSaved: function Example$CatalogBehavior$__cheeseSaved(item, cheese)
{
    /// <summary>Called when the cheese save is a success. Sets the Item
    /// number of the cheese to the new server-assigned number.</summary>
    /// <param name="item" type="Number">
    /// Item number assigned by the server</param>
    /// <param name="cheese" type="Epic.Training.Example.Web.Cheese">
    /// The cheese that was just saved</param>

    cheese.Item = item;
},
```

This will ensure that the record ID (i.e., item) of a new cheese on the client matches that assigned by the web server.

1. Create a new method that will inform the user if there is an error saving the cheese:

```
_saveError: function Example$CatalogBehavior$_saveError(error, cheese)
{
    /// <summary>Display a message if there was a problem saving the cheese
    /// </summary>
    /// <param name="error" type="Object">The error returned from the
    /// server</param>
    /// <param name="cheese" type="Epic.Training.Example.Web.Cheese">
    /// The cheese that could not be saved</param>

    alert("There was a problem saving the following cheese: \n\n\t"
        + cheese.toString() + "\n\n"
        + error.get_message());

    this._editCheese(cheese);
},
```

2. Add private fields to *CatalogBehavior.prototype* to store the save delegates. Be sure to add the corresponding XML comments in the constructor:

Add to CatalogBehavior.prototype:

```
Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior =
    function Example$CatalogBehavior(clientId)
{
    ...
    /// <field name="__saveDelegate" type="Function">(private) Called when the cheese is
    /// saved</field>
    /// <field name="__saveErrorDelegate" type="Function">(private) Called when the cheese
    /// cannot be saved</field>
```

```
Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.initializeBase(
this, [clientId]);
};
```

```
Epic.Training.Example.Web.Pages.Inventory.CatalogBehavior.prototype = {
    //region fields
    ...
    __saveDelegate: null,
    __saveErrorDelegate: null,
    //endregion
```

3. In *onLoaded*, populate the delegates:

```
onLoaded: function Example$CatalogBehavior$onLoaded () {
{
    ///<summary>Called by the constructor after "onLoad".</summary>
    ...
    this.__saveDelegate = $$fcd(this, this.__cheeseSaved);
    this.__saveErrorDelegate = $$fcd(this, this.__saveError);
},
}
```

4. Modify *\_\_validateAndSaveCheese* to save the cheese currently being edited:

```
__validateAndSaveCheese: function Example$CatalogBehavior$__validateAndSaveCheese () {
{
    /// <summary>Save cheese to the database then
    /// release edit mode</summary>
    /// <returns type="Boolean" >"true" if successful,
    /// "false" otherwise</returns>
    if (this.__curEditCheese !== null) {
        // TODO: Validate the cheese
        this.__service.AddOrReplaceCheese(
            this.__curEditCheese,
            this.__saveDelegate,
            this.__saveErrorDelegate,
            this.__curEditCheese);

        this.__gridInventory.set_rowReadOnly(this.__curEditCheese, true);
        this.__curEditCheese = null;
    }
    return true;
},
}
```

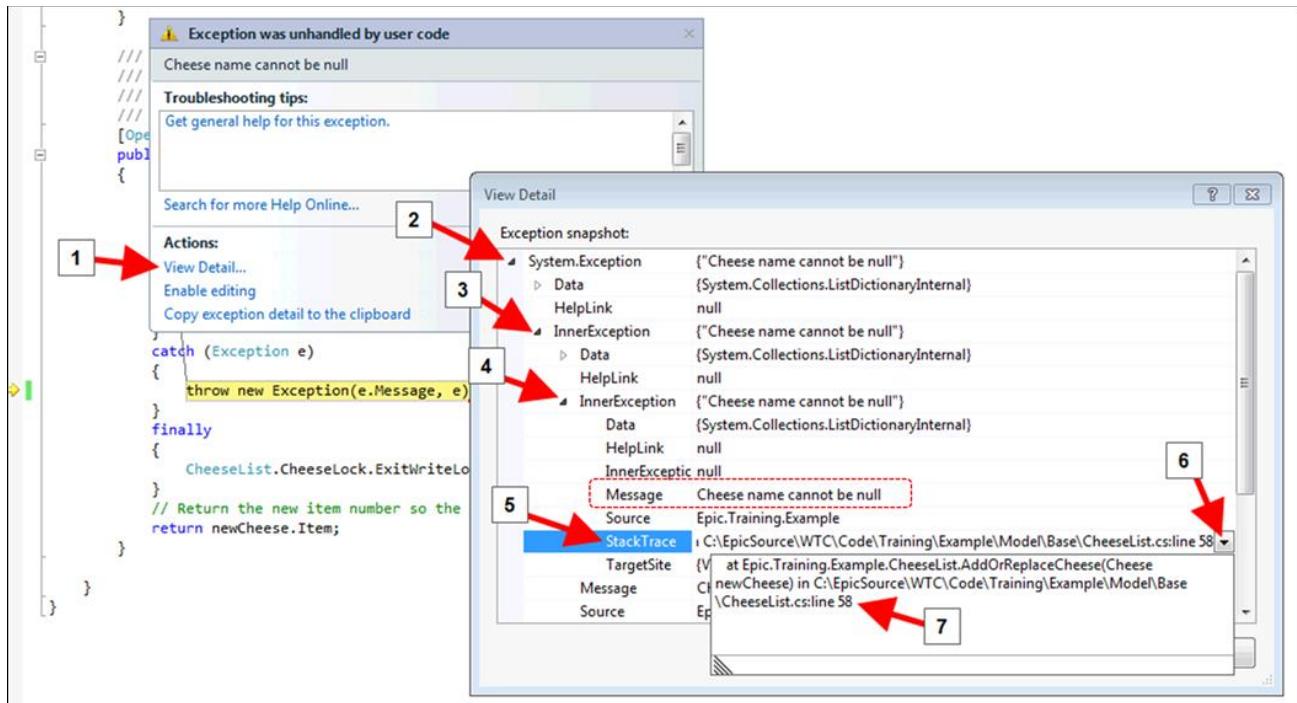
When calling an AJAX-Enabled WCF web service method marked with the *[OperationContract]* attribute, there are always certain arguments available in addition to those that you specified in the C# definition. The arguments are as follows:

Specified arguments	(optional) The first arguments match the parameters specified in C#. If there are no parameters in the data method definition, then the first argument will be the success delegate.
Success Delegate	(optional) This is the method that will be called if the service returns

	successfully. Whatever object you return will be the first argument passed to this method.
Failure Delegate	(optional) This is the method that will be called if the service throws an exception. The error object will be the first argument passed to the failure delegate.
Context	(optional) Whatever value you pass to this argument will end up being passed as the second argument to either the success or failure delegate.  If the server method that you call returns void and completes successfully, then the context will still be the second argument of the success delegate. In this case the first argument is null.

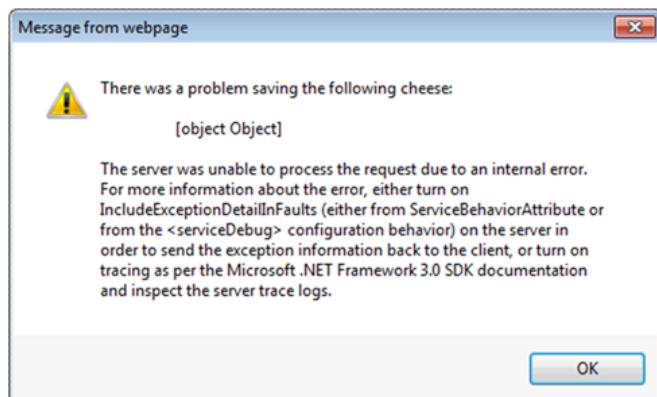
In `_validateAndSaveCheese`, you are passing `this._curEditCheese` twice. The first time is to make it available in the C# method `AddOrReplaceCheese`. The second time is to pass it back as the second argument of the success or failure delegates.

5. Run the web application and try modifying cheeses and adding new cheeses. The XML file should update as expected.
6. Add a new cheese and set the name to an empty string. An exception should be thrown on the server.
  - If visual studio doesn't break on the server exception, ensure that the "Enable Just My Code" setting is checked:  
**Tools > Options > Debugging > Enable Just My Code**
7. Click **View Detail** and follow the `InnerException` until it is null to find the root of the problem:



Follow inner exceptions of server errors to find the problem source

8. Click OK.
9. Press F5.
10. You should receive the following error on the client:



Client error resulting from a server exception

Notice that this error is not particularly useful. First of all, `[object Object]`, which is the default `toString` from `Object`, doesn't tell you much about the cheese that caused the error. Second, the actual server error has been intentionally masked. This is for security reasons, because hackers attacking the server can use error information to refine their attack if it is too detailed.

In this case, the error information will not help the hacker much, so it is ok to pass that information back to the client.

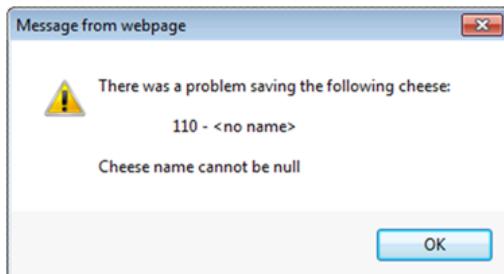
11. Since `[object Object]` is not a meaningful representation of a cheese, override the `toString` method of the `Cheese` JavaScript Class:

```
//#region public methods
toString: function Example$Cheese$toString() {
    ///<summary>Get the string representaiton of this cheese</summary>
    ///<returns type="String" />
    var result = this.Item.toString() + " - ";
    if (this.Name.length === 0) {
        result += "<no name>";
    }
    else {
        result += this.Name;
    }
    return result;
},
//#endregion
```

12. Unmask errors thrown from `CatalogBehavior.svc` by adding the following attribute to the `CatalogBehavior` class:

```
[ServiceBehavior(IncludeExceptionDetailInFaults=true)]
```

13. Try removing the name from a cheese again. This time you should see useful details on the client:



*Server errors reflected on the client in a meaningful way*

14. Although you could simply rely on the server to catch errors, that technique is not efficient. A better technique is to validate the cheese on the client before trying to save it. To do so, modify `_validateAndSaveCheese` to do the following if the name is empty:
- Alert the user that a cheese name must be provided
  - return false
15. Test your changes to make sure they work as expected.

```
_validateAndSaveCheese: function
Example$CatalogBehavior$__validateAndSaveCheese()
```

```
{  
    /// <summary>Save cheese to the database then  
    /// release edit mode</summary>  
    /// <returns type="Boolean" >"true" if successful,  
    /// "false" otherwise</returns>  
    if (this.__curEditCheese != null)  
    {  
  
        if (this.__curEditCheese.Name)  
        {  
  
            this.__service.AddOrReplaceCheese(  
                this.__curEditCheese,  
                this.__saveDelegate,  
                this.__saveErrorDelegate,  
                this.__curEditCheese);  
  
            this.__gridInventory.set_rowReadOnly(this.__curEditCheese, true);  
            this.__curEditCheese = null;  
  
        }  
        else  
        {  
  
            alert("Provide a name for the cheese");  
            return false;  
  
        }  
    }  
    return true;  
},
```

## Part 7: Loading the Details Page

In this step you will change the details page so that it loads the selected cheese details. You could write a web service to do this, but it is also possible to get this information on the server while the page is loading.

The screenshot shows a web application interface. On the left is a table titled "Catalog" with columns "Name" and "Image". The table lists various cheeses: Blue, Cheddar, Swiss, American, Feta, Limburger, Gouda, Mozzarella, and My New Cheese. Each cheese entry includes a small image, a brief description, and a "Edit Details" link. An arrow points from the "Edit Details" link for the Gouda entry to a modal dialog box on the right. The dialog box has a title "Details" and contains a large image of a wheel of Gouda cheese. To the right of the image are several input fields: "Item:" with value "792", "Name:" with value "Gouda", "Weight:" with value "25 lb", and "Price:" with value "67.5". Below these is a "Description:" text area containing the text "Hard on the outside, soft on the inside". At the bottom of the dialog are three buttons: "Edit", "Accept", and "Cancel".

Name	Image	Description	
Blue		Strong blue cheese.	
Cheddar		A very mild cheese.	
Swiss		Mellow flavor.	
American		Standard American cheese.	
Feta		Very creamy, tangy.	
Limburger		Very stinky.	<a href="#">Edit Details</a>
Gouda		Hard on the outside, soft on the inside.	<a href="#">Edit Details</a>
Mozzarella		Melts easily, great on Pizza.	<a href="#">Edit Details</a>
My New Cheese		Only the BEST cheese ever...	<a href="#">Edit Details</a>
<a href="#">Add Cheese</a>			

*Loading details specific to the requested cheese*



Keep in mind that what you are about to do breaks from the standard pattern of making a separate asynchronous call to load the data. By loading data into the page during the initial request, you are not taking advantage of concurrently rendering the page while waiting for the data to load.

This technique should only be used when the amount of data being loaded is very small and will somehow make the end-user experience better.

1. Open *Details.aspx.cs*
2. The query string is stored in `Request.QueryString["item"]`. Retrieve this value and parse it to an integer within *Page\_Load*.

```
protected void Page_Load(object sender, EventArgs e)
{
    int item;
    if(Int32.TryParse(Request.QueryString["item"],out item))
    {
        // Code if item parsed correctly
    }
}
```

The request object contains all information associated with the HTTP request, which includes the query string. The query string will include the parts past the "?" in the URL. For example, if the URL is:

```
http://address/page.aspx?key1="val1"&key2="val2"
```

Then the *Request.Query* collection will contain two keys, *key1* and *key2*, with values "*val1*" and "*val2*".

3. Call *GetCheeses* from *CatalogBehavior.svc.cs* to load the list, then iterate through the cheeses to find the matching cheese object:

```
Cheese matchingCheese = null;  
CatalogBehavior behavior = new CatalogBehavior();  
List<Cheese> cheeseList = behavior.GetCheeses();  
foreach (Cheese cheese in cheeseList)  
{  
    if (cheese.Item == item)  
    {  
        matchingCheese = cheese;  
        break;  
    }  
}
```

 The above code is considered poor style, since it is explicitly calling code that should be specific to *CatalogBehavior* from within *DetailsBehavior*:

```
CatalogBehavior behavior = new CatalogBehavior();  
List<Cheese> cheeseList = behavior.GetCheeses();
```

A more elegant solution would be to factor *GetCheeses* out of both behavior classes and place it within the business logic. For example, as a static method in the *CheeseList* class.

If you have time, feel free to refactor the example. Also, if you see this in actual code, be sure to mention it during PQA.

4. Set *imgFull.ImageUrl* to the *ImagePathLarge* property of the cheese object.
5. Set the *Text* property of the various text boxes to the corresponding property from the cheese object.
6. Save and run your project.
7. Navigate to the inventory page.
8. Click details for one of the rows. Are the details displayed as expected?

9. What happens if you navigate directly to the page, omitting *item* from the query string (delete the ?*item*=# part of the URL)?

■

- 
10. Change *Page\_Load* of *Details.aspx.cs* so that if *item* is not in the query string, the page redirects the user to *Default.aspx*, using *Response.Redirect("~/Default.aspx")*;
11. Also redirect to *Default.aspx* if the cheese returned is null.
12. The final code should appear similar to the following:

```
protected void Page_Load(object sender, EventArgs e)
{
    int item;
    if (Int32.TryParse(Request.QueryString["item"], out item))
    {
        Cheese matchingCheese = null;
        CatalogBehavior behavior = new CatalogBehavior();
        List<Cheese> cheeseList = behavior.GetCheeses();
        foreach (Cheese cheese in cheeseList)
        {
            if (cheese.Item == item)
            {
                matchingCheese = cheese;
                break;
            }
        }
        if (matchingCheese != null)
        {
            imgFull.ImageUrl = matchingCheese.ImagePathLarge;
            txtDescription.Text = matchingCheese.Description;
            txtItem.Text = matchingCheese.Item.ToString();
            txtName.Text = matchingCheese.Name;
            txtPrice.Text = matchingCheese.Price.ToString();
            txtWeight.Text = matchingCheese.Weight.ToString();
        }
        else
        {
            Response.Redirect("~/Default.aspx");
        }
    }
    else
    {
        Response.Redirect("~/Default.aspx");
    }
}
```

{}

13. Try navigating to the details page. Does the correct cheese appear?
14. Try navigating to the details page without the item query string. Are you redirected to the home page?
15. Try entering a bogus item number into the query string. What happens?

## Exercise: Send the E-mail

---

In the previous exercise, you used existing business objects to load the cheese inventory asynchronously. In this exercise you will create additional business objects from scratch. The goal is to save emails from the customer service page to an XML file.

### Part 1: Create the Email Class

---

1. Create a public C# Email class in the *Training.Example* project.
2. Include a public property (both get and set) for each field available on the email form of the customer service page, including:
  - Name
  - EmailAddress
  - Country
  - Subject
  - Message
  - Urgent.
  - Include an additional property *Sent* that stores the *DateTime* that the email was sent.

Property Name	Data Type
Name	string
EmailAddress	string
Country	string
Subject	string
Message	string

Property Name	Data Type
Urgent	bool
Sent	DateTime

3. Remove unused "using" statements
  - Edit > IntelliSense > Organize Usings > Remove and Sort
4. Add the *DataContract* attribute to the class so that it can be serialized to JSON.

```
[DataContract]
public class Email
```

5. Add *DataMember* attributes (for JSON) and *XmlAttribute* attributes (for the data file) to each property.
  - For example, the *Name* property would appear as follows:

```
[XmlAttribute, DataMember]
public string Name { get; set; }
```

## Part 2: Create the EmailList Class

1. Add a new XML Schema to the App\_Data directory named Emails.xsd.
2. Replace the auto-generated code with the following code to indicate the allowed structure of the file. You can copy this code from:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\EmailSchema.txt

```
<?xml version="1.0" encoding="utf-8"?>
<schema id="Emails"
  targetNamespace="Epic.Training.Example.Emails"
  elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
<element name="EmailList">
  <complexType>
    <sequence>
      <element name="Email" maxOccurs="unbounded" minOccurs="0">
        <complexType>
          <attribute name="Name" type="string" use="required" />
          <attribute name="EmailAddress" type="string" use="required" />
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
</schema>
```

```

<attribute name="Country" type="string" use="required" />
<attribute name="Subject" type="string" use="required" />
<attribute name="Message" type="string" use="required" />
<attribute name="Urgent" type="boolean" use="optional" />
<attribute name="Sent" type="dateTime" use="required" />
</complexType>
</element>
</sequence>
</complexType>
</element>
</schema>

```

3. Add a new XML file to the *App\_Data* directory named *Emails.xml*.

4. Add the following tags to *Emails.xml*:

```

<?xml version="1.0"?>
<EmailList xmlns="Epic.Training.Example.Emails">
</EmailList>

```

5. Try adding a sample email, to make sure that the XSD is properly formed (be sure to save the file when you are done). You should receive IntelliSense help on the elements/attributes:

```

<Email Country="US" Message="This is a test message"
      EmailAddress="trainee@epic.com"
      Name="A WTC Trainee" Sent="2010-07-14T12:13:14Z"
      Subject="Test message" Urgent="false" />

```

6. Create a public *EmailList* class in the *Epic.Training.Example* project.

7. Add a public property *Emails* that is a list of emails.

- Be sure that you are using *System.Collections.Generic*.

8. Make *EmailList* the XML root (corresponding to <*EmailList*></*EmailList*>):

```

[XmlRoot(Namespace="Epic.Training.Example.Emails")]
public class EmailList

```

9. Add an attribute to *Emails* so it is serialized as an XML element.

```

[XmlElement("Email")]
public List<Email> Emails { get; set; }

```

10. Create a default constructor that takes no arguments. In this constructor, initialize Emails to an empty list.
11. Create a public void method called *Send*. It takes an *Email* as a parameter.
  - Set the *Sent* property of the email to the current system time, *DateTime.Now*.
  - Add the email to the end of the list
12. Create a public static *ReaderWriterLockSlim* property to be used when accessing the list. Also add a static constructor to initialize it:

```
/// <summary>
/// Used to manage access to the email list.
/// </summary>
[XmlIgnore]
public static ReaderWriterLockSlim EmailLock { get; private set; }

/// <summary>
/// Static constructor used to initialize the EmailLock property
/// </summary>
static EmailList()
{
    EmailLock = new ReaderWriterLockSlim();
}
```

13. Clean and build your project.
14. Remove unused usings:
  - Edit > IntelliSense > Organize Usings > Remove and Sort

### Part 3: Create the Web Service

---

1. In *Global.ascx.cs*, add a public static string property to *Global* named *EmailFileName*:

```
public static string EmailFileName { get; private set; }
```

2. In *Application\_Start*, initialize the property so that it points to *Emails.xml*:

```
/// <summary>
/// Executed when web application first starts
/// </summary>
/// <param name="sender">Sender of the event</param>
/// <param name="e">Event arguments</param>
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
    EmailFileName = s_AppDataPath + "Emails.xml";
```

{}

3. Create a new AJAX-Enabled WCF Service called *CustomerBehavior.svc* in the root folder of *Training.Example.Web.Pages*.
4. Set the *ServiceContract* namespace:

```
[ServiceContract (Namespace = "Epic.Training.Example.Web.Pages.Services") ]
```

5. Change *DoWork* to *SendEmail*. Have it accept an *Email* as a parameter.
6. Obtain and release a write lock from *EmailList* within *SendEmail*. Be sure to use appropriate error handling and release your lock in a finally block.
7. Use *XmlLoader* to load the email list using *Global.EmailFileName*.
8. Call *Send* from the loaded email list, passing the email as an argument.
9. Use *XmlLoader* to save the list back to *Global.EmailFileName*.
10. Build your project and fix any errors you find.
11. Add a *ScriptManagerProxy* just above the *PageSettings* on the *Customer.aspx* page.
12. Include *CustomerBehavior.svc* in the script manager proxy of *Customer.aspx*.
  - Be sure to place it in the *Services* element, not the *Scripts* element.
13. Include an IntelliSense reference to *CustomerBehavior.svc* in *CustomerBehavior.js*
14. Add the *\_service* field to the *CustomerBehavior* JavaScript class' prototype, with the appropriate IntelliSense comments:

```
...
//<field
name="__service" type="Epic.Training.Example.Web.Pages.Services.CustomerB
ehavior">
/// The service object for this Behavior class
//</field>

...
__service: null,
...

onDataPreload: function() {
    this.__service = Epic.Training.Example.Web.Pages.Services.CustomerBehavior;
},
```

...

15. Create a method called `_emailSent` in the private methods region that will be called when the web service returns. Within the method, clear the email form and alert the user that the email was sent.
16. Create another private method called `_emailSendError` that is called if the service returns with an error. Alert the user that an error occurred.
17. Create delegate fields for success (`_sendEmailDelegate`) and failure (`_sendEmailErrorDelegate`) to send the email. Populate the fields in `onLoaded`.
18. in `_btnSendClick`, create a new generic JavaScript object with property names matching those of the C# Email class.
  - Assign the appropriate values to each property.
  - For example:

```
var newEmail = {  
    Name: name,  
    EmailAddress: email,  
    Country: country,  
    Message: message,  
    Subject: subject,  
    Urgent: this.getEl("chkUrgent") .checked  
};
```

19. Call the web service, passing the new email message and the delegates created above.

```
this._service.SendEmail(newEmail, this._sendEmailDelegate,  
this._sendEmailErrorDelegate);
```

20. Save and run your site, then try to send an email. Fix any errors that you find.



**A JavaScript class is not required for every C# class that you send to or get from the server.**

Notice you did not actually create a class for the new email object. Because JavaScript is loosely typed, you can get away with sending generic objects to the server, as long as all the required properties are in place.

That said, it makes sense to create strongly typed client classes if they are commonly used and shared among teams, because:

- The documentation of these classes will be extracted from the XML comments
- Additional methods and features can be added to the JavaScript class

## If you have time

---

If you feel that you still need more practice with the coding patterns, try creating a new page to display all emails that have been sent.

1. Create a JavaScript class to represent an email.
  - a. Place it in the *Training.Example.Web* project
  - b. Use the class snippet to quickly generate the class
  - c. Use the existing *Cheese* JavaScript class as an example
  - d. Set the properties of the new file so that it is an *Embedded Resource* rather than *Content*
  - e. Add an attribute to *Training.Example.Web > Properties > AssemblyInfo.cs* so that the new JavaScript file is correctly named.
  - f. To ensure that *Email.js* is available throughout the website, add a script reference to the script manager on the master page.
2. Instead of sending the generic email object, create a new instance of the *Epic.Training.Example.Web.Email* class.
  - The argument is the loosely-typed object of email properties.
3. Send the strongly-typed email object to the web service.
4. Test your web application to ensure that sent emails are still added to the XML file.
5. Create a new page called *Inbox.aspx* in the root of your *Pages* project. Be sure to use the master page.
6. Place a link to *Inbox.aspx* in the menu div on your master page.
7. Add a *PageSettings* to *Inbox.aspx*.
8. Add a new JavaScript class and corresponding js file called *InboxBehavior*.
9. Create a new AJAX-Enabled web service named *InboxBehavior.svc*
10. Create a web service method to load the email list.

- Obtain a read lock while loading the list, and be sure to use appropriate error handling.
11. Within *InboxBehavior.js*, load and display the email messages. Be sure to add any necessary controls to *Inbox.aspx*.
- Don't forget to add *InboxBehavior.svc* to a script manager proxy in *Inbox.aspx*
  - Add corresponding IntelliSense reference in *InboxBehavior.js*.
  - Add the `_service` field for easy access to the web services.

## Wrap Up

---

1. In *CatalogBehavior.js*, why were the `_saveDelegate` and `_saveErrorDelegate` stored private fields, while `loadHandler` was just placed in a temporary local variable?
  - \_\_\_\_\_
  - \_\_\_\_\_
2. Why is it always a good idea to validate on the client before sending data to the server?
  - \_\_\_\_\_
  - \_\_\_\_\_
3. If you perform client validation, do you still need to perform server validation?
  - \_\_\_\_\_
  - \_\_\_\_\_
4. Why do you need to be careful about using query strings to communicate with the server?
  - \_\_\_\_\_
  - \_\_\_\_\_
5. How was the data loaded differently in the details page compared to the catalog page?
  - \_\_\_\_\_
  - \_\_\_\_\_
6. When would you use the details method instead of the catalog method?
  - \_\_\_\_\_
  - \_\_\_\_\_

7. Why is the code used to release locks done in a finally block?

■

---

## Troubleshooting Web Services

---

There are several common mistakes that may cause web services to fail, that is, appear as null when JavaScript is running:

- Verify that the .svc file has been added to the script manager.
- Web service methods do not support overloading. Provide a unique name for each method in a service.
- Properties must be public and must have both get and set if marked with [DataMember].
  - Bad Example:

```
[DataMember]
public string SampleBadProperty
{
    get
    {
        return "test";
    }
}
```

- Web services are fragile:
  - It is better to delete and recreate the service than to rename it or change its namespace
  - To avoid this problem, verify that the default namespace is correct.
- If you delete, rename or change the namespace of a web service, modify Web.config to match.
- Auto-formatting a page that includes a service using (using **CTRL+K, CTRL+D**) will cause the service to break
  - To fix this, edit the service file (add a space somewhere) and then save it again.

# Communicating via Post Back

Traditional ASP.NET web applications make heavy use of the post-back technique to communicate with the server. Remember that *post* is one form method used to send information back to the server and *post-back* is when a page uses post to send information back to itself on the server.

## Post Back

The problem with this technique is that (traditionally) the entire page needs to be refreshed when post-back occurs. For example, when clicking through the wizard control on *Order.aspx*, the whole page refreshes every time **Next** or **Previous** is clicked.

The screenshot shows a wizard control titled "Place an Order" with four panels:

- Order Information:** Contains dropdowns for "Select a cheese" (Blue) and "Select quantity" (1), and a "Next" button.
- Shipping Address:** Contains fields for Name, Street Address, City, State, Zip, Phone, and Email, with "Previous" and "Next" buttons at the bottom.
- Shipping Method:** Contains a dropdown for "Shipping Method" with "Ground" selected, and "Second day" and "Next day" as other options. It also has a "Calculate" button and a label "Shipping cost is: [redacted]". Navigation buttons "Previous" and "Next" are at the bottom.
- Billing Information:** Contains dropdowns for "Type of card" (Visa, Master Card, American Express), "Card #:", and "Expires on:", and "Place Order" and "Previous" buttons at the bottom.

*Clicking Next or Previous causes a post-back*

If only a small change occurs as a result of the post back, this is needlessly wasteful in terms of network traffic as well as client and server processing time. ASP.NET can mitigate this by only updating a subset of the page, but as you will soon discover, it still isn't as efficient as the web-service technique introduced in the previous section.

Disadvantages of post-back technique	Advantages of post-back technique
<ul style="list-style-type: none"><li>• More work for server</li><li>• More work rendering on the client</li><li>• More network traffic</li></ul>	<ul style="list-style-type: none"><li>• </li><li>• </li><li>• </li></ul>

## Server Events

One reason that post-back is traditionally used is that it bypasses the need to write client code (i.e., JavaScript). By posting back to the server, events such as button clicks can be handled using server code. Changes to the page can also take place on the server, and then the entire page is sent back to

the client. For obvious reasons, it is easier on the developer if you only need to develop in one language (in this case C#).

An example of this that you will implement in the following exercise is the *OnPreRender* event of the *asp:WizardStep*. The handler of this event can be used to populate the list of available cheeses in the cheese drop-down control:

The diagram illustrates the interaction between an ASPX page, its code-behind file, and a browser interface.

- Orders.aspx:** Shows an *asp:WizardStep* control with an *OnPreRender* event handler assigned to *stpSelect\_preRender*.
- Orders.aspx.cs:** Shows the corresponding C# code for the *stpSelect\_preRender* event handler. It clears the dropdown list, retrieves a list of cheeses from a catalog behavior, and adds items to the dropdown based on their names and item strings.
- Browser Screenshot:** Shows a "Place an Order" page with an "Order Information" section. A dropdown menu labeled "Select a cheese:" shows a list of cheese names. The "American" cheese is selected.
- A red dashed box highlights the *OnPreRender* assignment in the ASPX markup, and a red dashed arrow points from this box to the corresponding event declaration in the code-behind file.

## Exercise: Exploring Server Events and Post Back

In this exercise you will practice handling post-back server events. Then you will observe and compare the network traffic generated by post back and web services. The goal is to illustrate how efficient web services are compared to post-back techniques.



**You should avoid using postback in new web applications**

If you end up working on existing web applications that make extensive use of postback then you may end up using it yourself, but avoid it for new development whenever possible.

## Part 1: Working with Server Events

1. Modify the wizard step with id *stpSelect* in *Order.aspx* so that it fires *stpSelect\_preRender* on the *PreRender* event:

```
<asp:WizardStep ID="stpSelect" OnPreRender="stpSelect_preRender" . . .
```

2. Implement *stpSelect\_preRender* in *Order.aspx.cs* as shown:

```
protected void stpSelect_preRender(object sender, EventArgs e)
{
    int idx = ddCheese.SelectedIndex;
    ddCheese.Items.Clear();
    List<Cheese> cheeses = new CatalogBehavior().GetCheeses();

    foreach (Cheese c in cheeses)
    {
        ListItem li = new ListItem(c.Name, c.Item.ToString());
        ddCheese.Items.Add(li);
    }
    ddCheese.SelectedIndex = idx;
}
```

3. Run your web application and navigate to *Order.aspx*.
4. Verify that the cheese drop down is populated as expected.
5. In the Shipping Method step of the order page, wire a server-side click event to the *Calculate* button (*OnClick* event).

```
<asp:Button Text="Calculate" ID="btnCalcShip" runat="server"
OnClick="BtnCalcShip_click" />
```

6. Add the corresponding method to *Order.aspx.cs*:

```
protected void BtnCalcShip_click(object sender, EventArgs e)
{
    int item = Int32.Parse(ddCheese.SelectedValue);
    int quantity = Int32.Parse(ddQuantity.SelectedValue);
    List<Cheese> cheeses = new CatalogBehavior().GetCheeses();
    Cheese selCheese = null;
    foreach (Cheese aCheese in cheeses)
    { // Search for the matching cheese
        if (aCheese.Item.ToString() == ddCheese.SelectedValue)
        {
```

```

        selCheese = aCheese;
        break;
    }
}
double pricePerPound = 5.0;
}

```

7. Within *BtnCalcShip\_click*, calculate the shipping cost and write it to *lblShipCost.Text*. The cost is computed as:
- Quantity \* Weight per unit \* price per pound \* shipping method modifier/100
  - The modifier is 1.0 for ground, 1.5 for 2day and 2.0 for 1day.

```

double shippingMethodModifier = 1.0;

switch (lstShipping.SelectedValue)
{
    case "2day":
        shippingMethodModifier = 1.5;
        break;

    case "1day":
        shippingMethodModifier = 2.0;
        break;
}

double cost = selCheese.Weight * pricePerPound * shippingMethodModifier / 100.0;
lblShipCost.Text = "Shipping cost is: " + cost.ToString();

```

8. Run the website and verify that the cost is being calculated correctly.

## Part 2: Observe Network Traffic

---

Fiddler is a tool that allows you to monitor page requests and responses as they occur, as well as alter their content. If you have not done so already, install it now from [www.fiddler2.com](http://www.fiddler2.com). Then, watch the Fiddler video linked here:

<http://brainbow.epic.com/Learning/Profile.aspx?csn=35544>

1. Launch Fiddler
2. Turn off IPv6 in Fiddler:
  - Tools > Fiddler Options > uncheck **Enable IPv6** (if available)
3. Start your web application and navigate to the order page.

4. Notice that the traffic created from loading the page appears. Highlight all the new traffic created by the order page (click the first relevant entry, hold the shift key, and then click the last). Look at the right side of the screen under statistics. What is the total number of bytes sent and received?

■

- 
5. To see *what* is being sent, click the **Inspectors** tab. The top half of the right section of Fiddler represents the request information, while the bottom half represents the response. Select **Raw** for both of these.
6. Since web traffic is compressed by default, you likely won't be able to read the details at first. To read the message, click the area labeled **Click here to transform:**



*Area indicating that the message is compressed*

The screenshot shows the Fiddler application interface. In the top left, there's a table of network traffic with one item listed. The top right contains various tabs: Statistics, Inspectors (which is selected), AutoResponder, Composer, Filters, Log, and Timeline. Below the tabs are several sub-tabs: Headers, TextView, WebForms, HexView, Auth, Cookies, Raw, JSON, and XML. A red box highlights the 'Request' tab, and another red box highlights the 'Request Content' area. In the main pane, the request details are shown, including the URL (GET http://localhost:1130/Inventory/Order.aspx) and various headers like Accept, User-Agent, and Host. A red arrow points from the 'Request Content' box to the raw request text. In the bottom right pane, the response details are shown, including the status code (HTTP/1.1 200 OK), headers (Cache-Control, Content-Type, Vary, Server, X-AspNet-Version, X-SourceFiles, X-Powered-By, Date, Content-Length), and the response body (HTML code for the Order.aspx page). A red box highlights the 'Response Content' area, and a red arrow points from it to the raw response text.

*Request and response details from navigating to Orders.aspx*

7. What is being returned?

■

8. Navigate to the shipping method step. How many bytes are transferred for that step?

■

---

9. Use the Inspectors tab to examine what is being returned. What is it?

■

---

10. Click Calculate. How many bytes are transferred and what is returned?

■

---

### Part 3: Reduce Post-Back Traffic

---

ASP.NET provides an AJAX extension that can reduce network traffic generated by post back events. The control is called an Update Panel. Instead of reloading the entire page, only controls in the update panel are refreshed.



Even though the update panel reduces network traffic, it doesn't reduce server load. This is because the entire page is rendered on the server, even though only the portion in the update panel is sent back to the client.

1. On *Order.aspx* add an update panel just above the wizard. It can be found in the toolbox under AJAX Extensions. Give it the ID *upWizard*.
2. Move the closing tag to just below the end of the wizard, so that the entire wizard is on the update panel.
3. Controls cannot be directly added to an update panel. Instead, they must be between the `<ContentTemplate>` `</ContentTemplate>` tags. Add these tags just above and below the wizard.
4. Run your website again and measure the traffic generated by navigating to the shipping method step of the wizard. What is being transferred and how large is it?

■

---

■

---

5. How does the transition between wizard steps appear different than when the update panel wasn't used?
  -

---

## Part 4: Reduce Traffic from Shipping-Cost Calculation

---

Currently, when **Calculate** is clicked, the entire wizard step is refreshed. This is a waste since only *lblShipCost* changes.

1. Add a new update panel containing a *ContentTemplate* just above as *lblShipCost* and just below *btnCalcShip*. Call the update panel *upShipCost*.
2. Move *lblShipCost* into the *ContentTemplate* of the update panel. The result should resemble the following:

```
<asp:Button Text="Calculate"
    ID="btnCalcShip" runat="server"
    OnClick="BtnCalcShip_Click" />
<asp:UpdatePanel ID="upShipCost" runat="server">
    <ContentTemplate>
        <asp:Label ID="lblShipCost"
            Text="Shipping cost is: " runat="server" />
    </ContentTemplate>
</asp:UpdatePanel>
```

3. You need to be more precise with what triggers a refresh of the respective update panels. Inside *upShipCost*, above the content template, add *<Triggers> </Triggers>* tags. Within triggers, specify that the click event of *btnCalcShip* triggers the update.

```
<Triggers>
    <asp:AsyncPostBackTrigger ControlID="btnCalcShip"
        EventName="Click" />
</Triggers>
```

4. Make sure that the click event of *btnCalcShip* doesn't automatically refresh *upWizard*. Add the following attributes to *upWizard*:

```
ChildrenAsTriggers="false" UpdateMode="Conditional"
```

5. We only want the *upWizard* to post back when the active wizard step changes. Have it listen to the *ActiveStepChanged* event with the following trigger:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="wzdOrder" EventName="ActiveStepChanged" />
</Triggers>
```

6. Measure the traffic of calculating the shipping cost. What is the size of the network traffic and what exactly is being returned?:
- \_\_\_\_\_
  - \_\_\_\_\_

7. Navigate to the customer service page and measure the network traffic required to send an email using an asynchronous call. What is sent (top half of inspectors) and how large is it?
- \_\_\_\_\_
  - \_\_\_\_\_

8. To make it easier to parse the request and response of sending the email, change from **Raw** to **JSON**. The result should look similar to the following:

The screenshot shows the Fiddler application interface with two main sections. The left section displays a single captured request to 'localhost:1130 /CustomerBehavior.svc/SendEmail'. The right section, titled 'Inspectors', shows the raw request and response in JSON format.

**Request (Raw):**

```
POST /CustomerBehavior.svc/SendEmail HTTP/1.1
Content-Type: application/json; charset=utf-8
Host: localhost:1130
Content-Length: 143

{"toSend": {"Country": "US", "EmailAddress": "wtcStudent@epic.com", "Message": "This is a test", "Name": "WTC Student", "Sent": "Date(1397249258026)", "Subject": "A test email", "Urgent": true}}
```

**Request (JSON):**

```
{"toSend": {"Country": "US", "EmailAddress": "wtcStudent@epic.com", "Message": "This is a test", "Name": "WTC Student", "Sent": "Date(1397249258026)", "Subject": "A test email", "Urgent": true}}
```

**Response (Raw):**

```
d=(null)
```

**Response (JSON):**

```
d=(null)
```

*JSON view in fiddler*

9. In terms of the order of magnitude of the network traffic, how does this compare to the relatively simple task of calculating the shipping cost?
- \_\_\_\_\_



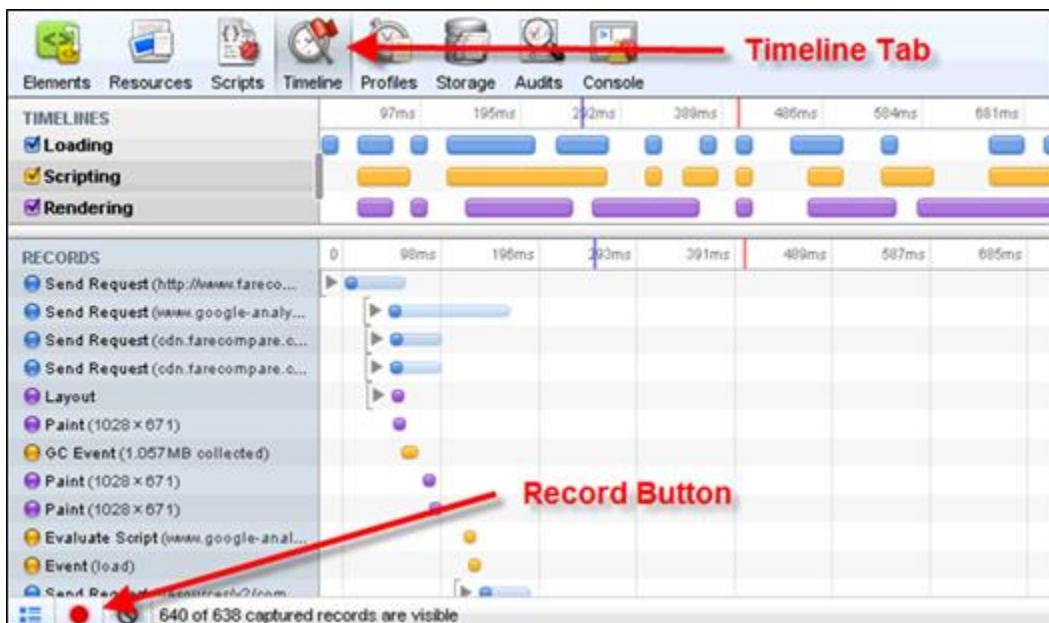
Note that this isn't an apples-to-apples comparison, but the magnitude of the data sent and received should give you an idea of how much more efficient web services are than update panels.

Also keep in mind that post back requires the entire page to re-render on the server, while a web services doesn't reload the page at all.

## If you have time

For performance testing on the client, every browser includes a profiler that you can use. Try using Chrome to record a timeline of various pages in the example site loading.

1. Run your site in Chrome
2. Press **CTRL + SHIFT + J** to open the developer tools
3. Switch to the timeline tab
4. Press the **Record** button
5. Refresh the page



An example of the Chrome profiler

## Wrap Up

1. What was the network traffic for each of the following:
  - Navigating wizard, no update panel: \_\_\_\_\_
  - Navigating wizard, with update panel: \_\_\_\_\_
  - Calculate shipping cost, with panel: \_\_\_\_\_

- Sending an email (web service): \_\_\_\_\_
2. Which method should you use in most cases, web services or post-back with update panels?
- \_\_\_\_\_

# Implementing User Security

ASP.NET has a built-in framework for handling user security. In this section, you will use that framework with the in-class example.

## Options Vary by User

The Big-Mouse cheese factory wants their web application to support three kinds of users: anonymous users (i.e., potential customers), customers with an account, and administrators who have the ability to edit inventory information.

Each role has the following authorization rules:

- All users can access the home, log-in and customer-service pages
- Authenticated customers can access all pages, but cannot edit inventory information
- Administrators can access all pages, and can additionally edit inventory information

## Authentication

There are two types of authentication available in ASP.NET. The default mode is *Windows* authentication, where ASP.NET takes the authentication from the OS. The second is *Forms* authentication, which forces the user to authenticate for every new session. The method being used is specified in the *Web.config* file:

```
<authentication mode="Forms">
    <forms loginUrl="Login.aspx" />
</authentication>
```

In the code above, Forms authentication is used and the login control on Login.aspx is used to authenticate the user.

With standard forms authentication, user credentials are included in the *web.config* file.

However, when working with systems that keep user information elsewhere (e.g., Chronicles), it is necessary to customize how forms authentication works. This is done using the abstract .NET framework classes *MembershipProvider* and *RoleProvider*.

MembershipProvider

Defines the contract that ASP.NET implements to provide membership services using custom membership providers.

For more details on creating custom membership providers beyond what

	is covered in this lesson, see: <ul style="list-style-type: none"><li>• <a href="https://msdn.microsoft.com/en-us/library/f1kyba5e.aspx">https://msdn.microsoft.com/en-us/library/f1kyba5e.aspx</a></li></ul>
RoleProvider	Defines the contract that ASP.NET implements to provide role-management services using custom role providers.  For more details on creating custom role providers beyond what is covered in this lesson, see: <ul style="list-style-type: none"><li>• <a href="https://msdn.microsoft.com/en-us/library/8fw7xh74.aspx">https://msdn.microsoft.com/en-us/library/8fw7xh74.aspx</a></li></ul>



For more details on how standard forms authentication works, see:

- <https://msdn.microsoft.com/en-us/library/7t6b43z4.aspx>

## Authorization

Authorization for individuals or groups of users can be set at either the file or folder level, though the folder-level is preferred because it is easier to maintain. This setting is also done in the *Web.config*.

For example, the following code prevents unauthenticated users from accessing anything in the Inventory folder:

```
<location path="Inventory">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
```

## Exercise: Implement User Security

In this activity you will work with Forms security in ASP.NET.

### Part 1: Setup Forms Authentication

1. Add the following files to the *App\_Data* folder of your *Pages* project:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\App\_Data\Users.xml

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\App\_Data\Users.xsd
  - Take a moment to examine the content of *Users.xml*, specifically, the user names and passwords available.
2. Take a moment to examine the following files, which are already in *Training.Example*:
- *User.cs*
  - *UserList.cs*



The file *User.cs* contains a class named *BigMouseUser*. This is to prevent a name conflict with the .NET class named *User*. Technically the name conflict can be resolved by fully specifying the namespace, but it is more convenient to just avoid the conflict.

3. Add the property *UserFileName* to *Global.asax.cs*:

```
/// <summary>
/// File where users are stored
/// </summary>
public static string UserFileName { get; private set; }
```

4. Initialize *UserFileName* in *Application\_Start*. Additionally, pass it to the *FileName* static property of *UserList*:

```
UserFileName = s_AppDataPath + "Users.xml";
UserList.FileName = UserFileName;
```

5. Stop the development server if it is running.
6. Add a new project to your solution for web-specific security logic targeting the *Example* application:
- **Project Template:** Visual C# > Class Library
  - **Name:** Security
  - **Path:** C:\EpicSource\WTC\Code\Training\Example\Web\
7. Rename the project *Training.Example.Security.Web* and set the assembly name and default namespace to *Epic.Training.Example.Security.Web*.
8. Set the build directory of all configurations to ..\..\..\bin\
9. Delete *Class1.cs*
10. Add the following references to *Training.Example.Security.Web*:
- *System.Configuration* (.NET tab)
  - *System.Web.ApplicationServices* (.NET tab)

- *Epic.Training.Example* (browse tab)
  - *Epic.Training.Core.Xml* (browse tab)
11. Set a project dependency on *Training.Example* and *Training.Core.Xml*
  12. Add the following existing item to the root of *Training.Example.Security.Web*:
    - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\cs\ CustomMembershipProvider.cs
  13. Build the *Training.Example.Security.Web* project so the binaries are available for reference.
  14. To activate forms authentication using the *CustomMembershipProvider* class, make the following changes to *Web.config* (in your *Pages* project):

```
<system.web>
  ...
  <authentication mode="Forms">
    <forms loginUrl="Login.aspx" />
  </authentication>

  <membership defaultProvider="CustomMembershipProvider">
    <providers>
      <clear />
      <add name="CustomMembershipProvider"
           type="Epic.Training.Example.Security.Web.CustomMembershipProvider" />
    </providers>
  </membership>
</system.web>

<location path="Inventory">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
```

15. Add a reference in *Training.Example.Web.Pages* to *Epic.Training.Example.Security.Web*.
  - Also add a project dependency.
16. Run your site and navigate to *Catalog.aspx*.
  - You should be redirected to *Login.aspx* automatically.
17. Sign in using admin for the username and password.
  - You should be redirected to *Catalog.aspx*.

## Part 2: Make the details page editable

One key difference between Customers and Administrators is that administrators will be able to edit a cheese on the details page. In this part, you will activate the details page for editing. Later you will restrict this option to the admin role.

1. Add an existing item to *Inventory*:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\scripts\DetailsBehavior.js

2. Register the *PageSettings* control in *Details.aspx*:

```
<%@ Register TagPrefix="etcw"
    Namespace="Epic.Training.Core.Controls.Web"
    Assembly="Epic.Training.Core.Controls.Web" %>
```

3. Add a new AJAX-Enabled WCF Service named *DetailsBehavior*

4. Setup *DetailsBehavior.svc.cs* so that it will work with *DetailsBehavior.js*. Also, add a void web service method named *SaveCheese* that calls *AddOrReplaceCheese* from *CatalogBehavior*:

```
using System.ServiceModel;
using System.ServiceModel.Activation;
namespace Epic.Training.Example.Web.Pages.Inventory
{
    [ServiceContract(Namespace =
"Epic.Training.Example.Web.Pages.Inventory.Services")]
    [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    public class DetailsBehavior
    {
        [OperationContract]
        public void SaveCheese(Cheese existingCheese)
        {
            new CatalogBehavior().AddOrReplaceCheese(existingCheese);
            return;
        }
    }
}
```

5. In *Details.aspx*, register the *PageSettings* control

- Pull the code from *Catalog.aspx*

6. Add a page settings and script manager proxy to the bottom of *Details.aspx*, within the content control. You can copy this code segment from:

- I:\Internal\Advanced\Web Tech Camp\Big-Mouse\text\DetailsScript.txt

```
<asp:ScriptManagerProxy ID="smp" runat="server">
    <Services>
        <asp:ServiceReference Path="DetailsBehavior.svc" />
    </Services>
</asp:ScriptManagerProxy>

<etcw:PageSettings runat="server"
    ClientClass="Epic.Training.Example.Web.Pages.Inventory.DetailsBehavior"
    ClientScriptPath="DetailsBehavior.js">
    <ManagedControls>
        <etcw:ControlEntry ControlID="txtItem" />
        <etcw:ControlEntry ControlID="txtName" />
        <etcw:ControlEntry ControlID="txtWeight" />
        <etcw:ControlEntry ControlID="txtPrice" />
        <etcw:ControlEntry ControlID="txtDescription" />
        <etcw:ControlEntry ControlID="btnEdit" />
        <etcw:ControlEntry ControlID="btnAccept" />
        <etcw:ControlEntry ControlID="btnCancel" />
    </ManagedControls>
</etcw:PageSettings>
```

- To ensure read-only controls appear correctly, verify that *Details.css* contains the following selector:

```
input[readonly], textarea[readonly]
{
    background-color:#F0F0F0;
}
```

- Run your site and verify that you can edit a cheese on the details page and save the changes by clicking **Accept**.

### Part 3: Create a Log-out Link

Logging out is an example where using post-back is ok. The user expects that this operation will take more time, so a complete page refresh is acceptable.

- In the master page, add a link button to the menu to log out.

```
<asp:LinkButton ID="lbLogout" runat="server"
    OnClick="lbLogout_Click">Log out</asp:LinkButton>
```

2. Implement the *onclick* event in the master page as follows:

```
protected void lbLogout_Click(object sender, EventArgs e)
{
    // Removes authentication
    FormsAuthentication.SignOut();

    // Empty the session variable (see next part of exercise)
    Session.Clear();

    // Return home
    Response.Redirect("~/Default.aspx");
}
```

3. Run your page. Verify that the logout button works as expected.

## Part 4: The Session Variable

The user record stores a variety of information that we want to cache for use later, rather than looking it up every time. The *Session* variable is useful for this purpose, because it is specific to one client session.

1. Open *Login.aspx* and change the *ID* attribute of the login control to *ID="login"*.
2. Open to *Login.aspx* in the design view by clicking the **Design** toolbar button at the bottom left of the code editing area.
3. If you see an error in the design view, right click the login control and select "Refresh"
4. Select the login control, then open the properties window: **View > Properties Window**
5. In the properties window, switch to events (the lightning bolt).
6. Double click the *LoggedIn* event. It will create C# event handler for the event.



Be sure not to double click *LoggingIn*, which is not the event you are looking for.

7. Within the *LoggedIn* event handler, load the user and store the country, email and name into *Session*:

```
protected void login_LoggedIn(object sender, EventArgs e)
{
    BigMouseUser u = null;
    UserList ul = XmlLoader<UserList>.LoadItems(UserList.FileName);
    foreach (BigMouseUser bmu in ul.Users)
```

```

{
    if (bmu.Name == login.UserName)
    {
        u = bmu;
        break;
    }
}
if (u != null)
{
    Session["country"] = u.Country;
    Session["email"] = u.Email;
    Session["name"] = u.Name;
    Session["role"] = u.Role;
}
}

```

8. Automatically populate the corresponding fields from the email section in the *Page\_Load* event of *Customer.aspx*:



Make sure that *runat="server"* for *selCountry*, *txtEmail* and *txtName*, otherwise the following code will not compile.

```

protected void Page_Load(object sender, EventArgs e)
{
    if (Session["country"] != null)
    {
        selCountry.Value = Session["country"].ToString();
    }

    if (Session["email"] != null)
    {
        txtEmail.Value = Session["email"].ToString();
    }

    if (Session["name"] != null)
    {
        txtName.Value = Session["name"].ToString();
    }
}

```

9. *Log-in* and *log-out* options should not appear simultaneously in the menu.

- You can set this up in the *Page\_Load* method of the master page.

- Check to see if the user is authenticated using:  
`HttpContext.Current.User.Identity.IsAuthenticated`
- If so, hide log in and show log out. If not, reverse this.
- Make sure that both log-in and log-out are server controls with an ID assigned, so you can access them on the server.

## Part 5: Create the CustomRoleProvider Class

---

You have user authentication, but there isn't a concept of different user roles (yet).

1. Add a new class to `Training.Example.Security.Web` called `CustomRoleProvider`
2. Have `CustomRoleProvider` inherit from the `RoleProvider` class.
3. Right click on `RoleProvider` and select **Implement Abstract Class**.
4. Implement the methods `GetRolesForUser` and `IsUserInRole`.
  - You can determine the user's role by loading the user list and finding the matching user
  - Use `GetRolesForUser` to write the body of the method `IsUserInRole`.

```
public override string[] GetRolesForUser(string username)
{
    UserList ul = XmlLoader<UserList>.LoadItems(UserList.FileName);
    foreach (BigMouseUser u in ul.Users)
    {
        if (u.Name == username)
        {
            return new string[] { u.Role.ToString() };
        }
    }
    return new string[] { };
}
```

## Part 6: Activate & Use the RoleProvider

---

1. Activate roles by modifying `Web.config`. Place the bolded code below inside the `system.web` element:

```
<membership defaultProvider="CustomMembershipProvider">
    ...
</membership>

<roleManager enabled="true" defaultProvider="CustomRoleProvider" >
    <providers>
        <clear/>
```

```
<add name="CustomRoleProvider"
      type="Epic.Training.Example.Security.Web.CustomRoleProvider"/>
</providers>
</roleManager>
```

2. In the *Page\_Load* method of *Details.aspx.cs*, check to see if the user is authenticated and roles are enabled. If so, determine if the user's role is that of an administrator. If not, hide the **Edit**, **Accept** and **Cancel** buttons by adding the CSS class *hidden*.

```
if (User.Identity.IsAuthenticated && Roles.Enabled)
{
    if (!Roles.IsUserInRole(User.Identity.Name, "Administrator"))
    {
        // Code to hide buttons
        btnAccept.CssClass += " hidden";
        btnCancel.CssClass += " hidden";
        btnEdit.CssClass += " hidden";
    }
}
```

You could also set the *Visible* attribute of the buttons to false. However, this would be different in that it actually prevents the control from being rendered to HTML.

3. It is not enough to prevent access to the web server on the client, as a skilled hacker could call the web service directly. Add the following code to the beginning of *CatalogBehavior.AddOrReplaceCheese*.

```
string username = HttpContext.Current.User.Identity.Name;
if (!Roles.IsUserInRole(username, "Administrator"))
{
    throw new Exception("The user is not an administrator");
}
```

Other session-specific information is also available in *HttpContext.Current*, such as *Session*.

4. Save and run your site.
5. Try logging in as a customer, then navigate to the details page. Are the buttons displayed?
  - Username: customer
  - Password: customer

6. Use the developer tools in IE or Chrome to enable remove the "hidden" CSS class from the buttons. You can do this by navigating the DOM to where the buttons are located, or by searching the DOM for the id of one of the buttons.
  7. Try changing the cheese details and saving the cheese. Does the web service prevent this from happening?
- 
- 

**Remember to always validate data coming from the client!**

Validate on the server even if you already validated it on the client. Consider client-side validation as contributing to the end-user experience, while server-side validation enhances the application's security.

**If you have time**

---

1. Navigate to the inventory page, then logout. Next, try the back button. What happens?
  - Chrome:  
\_\_\_\_\_
  - IE:  
\_\_\_\_\_
2. IE is pulling the page from cache and then reverting to Windows authentication to grant your user non-anonymous access under your Windows account.
  - To prevent this from happening, change the Inventory location in your web .config to allow the roles of Administrator and Customer, but deny access to anonymous users:

```
<location path="Inventory">
    <system.web>
        <authorization>
            <allow roles="Administrator"/>
            <allow roles="Customer"/>
            <deny users="*"/>
        </authorization>
    </system.web>
</location>
```
3. Navigate to the details page of one of the cheeses, log out, and then try the back button again. What happens this time? How is it different than before, and why is it different?
  - Chrome:  
\_\_\_\_\_

- IE:
- 
- 

4. Modify the inventory page so that administrator-only options are not displayed to customers.
  - Try removing table/row options from *gridInventory* on the server.
5. Finish creating the inbox page from the previous if-you-have-time exercise.
6. You should prevent non-administrators from accessing *Inbox.aspx*. Do so with a location entry in *Web.config*.

```
<location path="Inbox.aspx">
  <system.web>
    <authorization>
      <allow roles="Administrator"/>
      <deny users="*"/>
    </authorization>
  </system.web>
</location>
```

**The location entry above is being applied to an individual file**

Keep in mind that it is generally a better idea to set security to folders rather than directly to files, because it is easier to maintain.

7. The inbox link should only be visible to administrators. Add code to the master page to hide the link if the user is not an administrator (remember, the user's role is cached in *Session*).
8. Prevent the *GetEmails* web service method from being called by non-administrators.
9. Test the previous step by:
  - Logging in as an administrator
  - Navigate to the inbox
  - Log out
  - Log in as a customer
  - Click back until you reach the inbox page.

## Wrap Up

---

1. When would you hide controls with the hidden class vs. hiding them with the server property *Visible=false*?

- \_\_\_\_\_
  - \_\_\_\_\_
2. Why is it better to apply security to folders rather than individual files?
- \_\_\_\_\_
  - \_\_\_\_\_
  - \_\_\_\_\_
3. Which is more secure, loading all of the data with the initial page request, or loading it as a separate asynchronous web-service call?
- \_\_\_\_\_
  - \_\_\_\_\_
  - \_\_\_\_\_

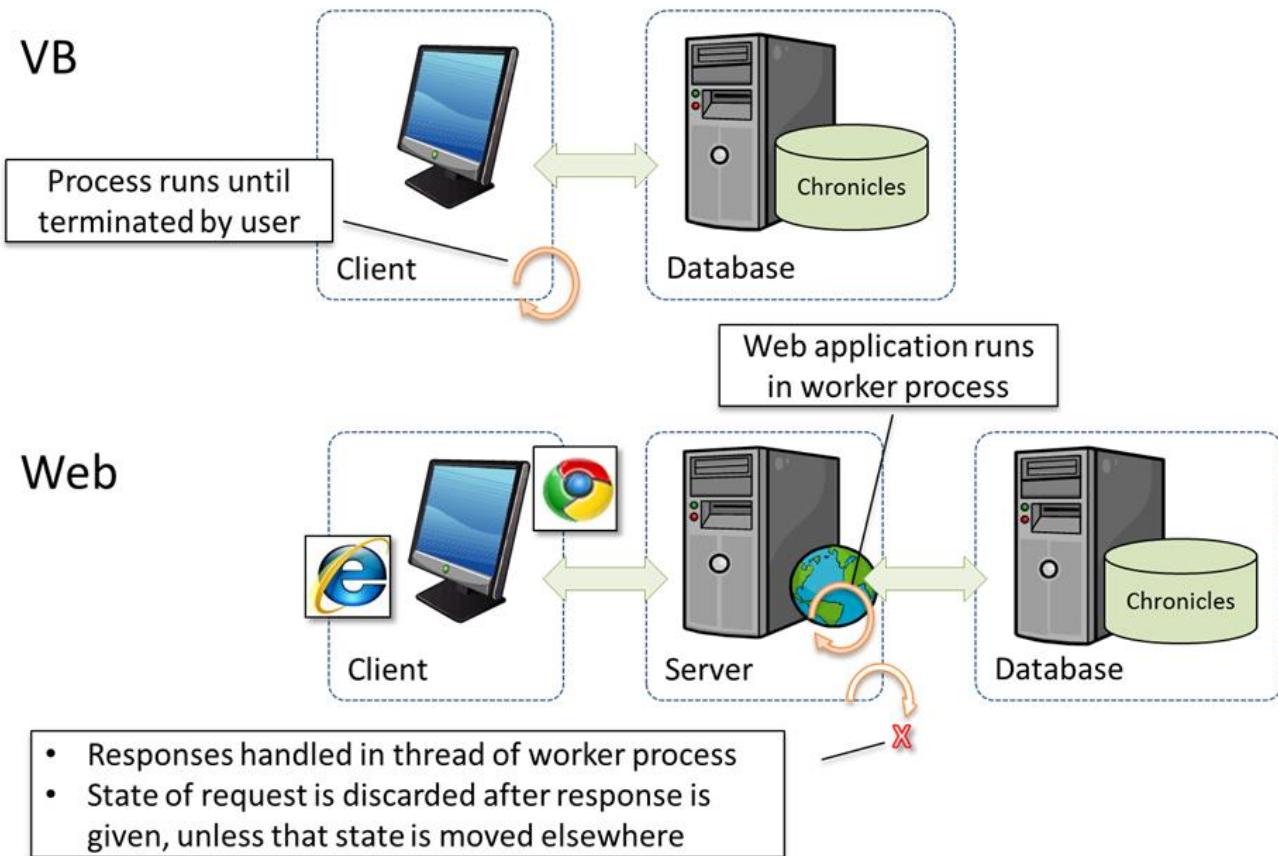
# Managing State

Managing state in a web application can be considerably more difficult than in a rich client application written in VB. This is both because state on the web server is handled differently, and because there are more kinds of state to choose from.

## State in VB vs. Web

In VB, data saved in memory lasts until the application closes. In other words, as long as Hyperspace is running, data that you have cached will remain available.

On the web, the state used to respond to a request is only available until the response is sent back. If there is any information that you want to keep, you need to first decide where you want to keep it.



### Process lifespan in VB and on the web

One form of state that needs to be maintained is the data in controls on the page. Answer the following questions to remember how this information is stored.

- What must you add to an element/control to make it available on the server?

- 

---

- 2. When are these controls first available on the server?

  - 

---

  - 3. When is the next time the controls become available on the server?

    - 

---


  - 4. Where is the state of these controls stored between post-back requests?

    - 

---


  - 5. Can you access the controls during web service calls?

    - 

---

It is precisely the split between client and server state that makes web programming so challenging. By default, ASP.NET keeps control and view state embedded within a page in hidden variables. This ensures that the state is always available during post back, but it is not ideal for the load pattern preferred by Epic, because it is not available during a web-service call.

The good news is that there are a variety of locations to store state that **are** available during web-service calls.

ASP.NET Session	<p>This is a string-keyed dictionary of objects accessed using <code>HttpContext.Current.Session</code>.</p> <p>ASP.NET session is a shared resource (in terms of capacity) in which each session gets their own separate view. This is an appropriate place to store information that is private to just one session.</p>
Data Files	Data files can be used to store information that is shared between sessions that must persist when the web application restarts.
Databases	A database, such as Chronicles, is often the best place to keep information that must be shared between sessions, accessed in a

	consistent way, and must persist between web application resets.
Static Variables	<p>Static variables can be used to store information directly in the worker process. This is an available option, but there are web-server configurations (i.e., web gardens and web farms) that make statics tricky to use.</p> <p>Additionally, when working within a framework like HyperspaceWeb, statics are very difficult to use correctly.</p>
ASP.NET Cache	<p>This is a string-keyed dictionary accessed using <code>HttpContext.Current.Cache</code>.</p> <p>Like static variables, ASP.NET cache is a shared resource in the worker process that all sessions running in that process have the same access to. This is an appropriate place to temporarily store information to reduce the number of database or file accesses, provided that the data does not change very often.</p> <p>The Cache can be invalidated based on certain conditions, such as a timeout, or when a file is updated.</p>



### State on the server

Where can data be stored to make it available on the server during a web-service call?

- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

## Web Gardens and Farms

The previous diagram illustrated a standard web server scenario, that is, one worker process is available to all users of the web application. There are two other common configurations that can make maintaining state even trickier:

Web Garden	<p>In this configuration there is one web server, but the web application will have multiple worker processes available. In this case, IIS will assign a worker processes to each request as it arrives.</p> <p>The reason state is a bit more complicated in this scenario is that each worker process has its own memory, so a storage location that was previously shared among all sessions on the server will now only be shared by all sessions running in the same worker process. Locations affected include ASP.NET cache and static variables.</p> <p>Additionally, some storage locations may still work but will need to be configured differently. For example, the ASP.NET session state must be moved out-of-process so requests in that session can be handled by any worker process on the server.</p>
Web Farm	<p>A web farm is a set of separate web servers that are all running the same web application. When a request arrives, a separate server, called a load balancer, chooses a web server for the request.</p> <p>Just like with web gardens, web farms complicate state maintenance because state that must be available to all sessions cannot exist on just one of the web servers.</p> <p>In this case, ASP.NET session state can still be preserved if it is moved to a separate SQL database.</p>

## Exercise: Maintaining State

In this exercise you will experiment with various options to maintaining state on the client and server.

### Part 1: Using session-specific state on the client

If you click the clock, it toggles between 12h and 24h time. However, if you switch from one page to another, the clock will reset to whatever was assigned on the server. It would make more sense to remember the user preference from page to page. This feature is easy to achieve using a cookie.

Cookies are nothing more than small amounts of information that are saved on the client machine that can be retrieved later. They are commonly used to save user preferences and to track other user-specific information.

Cookies are set by assigning a string to the `document.cookie` property:

```
document.cookie = "<name>=<value>; [expires=<date>;] [path=<path>;]  
[domain=<domain>;] [secure]";
```

### The cookie property is not like other JavaScript properties

If you make multiple assignments to `document.cookie` using the `=` operator, it actually will append additional cookies, not replace existing ones.

If you have multiple cookies to set, each one should be set in a separate assignment statement:

```
document.cookie = "cookie 1 string";  
  
document.cookie = "cookie 2 string";  
  
...
```

Property	Description
<code>name</code>	The name of the cookie
<code>value</code>	A string representation of the cookie's value
<code>date</code>	This optional value sets the date that the cookie will expire on. The date should be in the format returned by the <code>toGMTString()</code> method of the <code>Date</code> class. If the <code>expires</code> value is not given, the cookie will be destroyed the moment the browser is closed.
<code>path</code>	This optional value specifies a path within the site to which the cookie applies. Only documents in this path will be able to retrieve the cookie. Usually this is left blank, meaning that only the path that set the cookie can retrieve it.
<code>domain</code>	This optional value specifies a domain within which the cookie applies. Only websites in this domain will be able to retrieve the

Property	Description
	cookie. Usually this is left blank, meaning that only the domain that set the cookie can retrieve it.
secure	This optional flag indicates that the browser should use SSL when sending the cookie to the server. This flag is rarely used.

- When the display mode of the clock is set, save a corresponding cookie to the browser:

```
// Save a cookie to remember the clock state
var cookie = "isTwentyFourHour=" + this.__isTwentyFourHour.toString();
cookie += "; path=/"; // The cookie is available on all pages
document.cookie = cookie;
```

The next step is to get the cookie in the constructor and use it if it is available, instead of the value passed in by the constructor for *isTwentyFourHour*.

- The available cookies can be accessed from *document.cookie*.
  - The value is a string of the form "cookie1=value1; cookie2=value2; ...".
  - The cookies are delimited by semicolon-space ("; ").
- Before setting a value to *this.\_\_isTwentyFourHour* in the *Clock* constructor, determine if a cookie with the desired state has been set. If so, use that value to set *this.\_\_isTwentyFourHour*:

```
// Get the cookie named "isTwentyFourHour", if it exists

var cookies, ln, cookie;
if (document.cookie) { // make sure that there are cookies
    cookies = document.cookie.split("; ");
    for (ln = 0; ln < cookies.length; ln++) {
        cookie = cookies[ln].split("=");
        if (cookie[0] === "isTwentyFourHour") {
            isTwentyFourHour = Boolean.parse(cookie[1]);
        }
    }
}
```

- Run the web application and verify that the clock state remains the same when switching between pages.



The above information describes how to get and set cookies in JavaScript. If you are interested and have some free time, read the following to see how cookies are handled in ASP.NET:

- <http://msdn.microsoft.com/en-us/library/ms178194.aspx>

There are more modern storage options on the client beyond cookies. If you have the time, take a moment to read about them:

- [localStorage](#)
- [sessionStorage](#)

## Part 2: Using session-specific state on the server

You already have some information saved in Session from the previous exercise. In this part, you will use this information on even more pages.

1. Add the country-specific cheese lists as existing items to the *App\_Data* folder of your *Pages* project from the following path:
  - I:\Internal\Advanced\Web Tech Camp\Big-Mouse\App\_Data\
2. Modify *CheeseFileName* in *Global.asax.cs* so that it takes into account the user's country:

```
public static string CheeseFileName {
    get
    {
        // Get the country from the Session cache
        // based on the user's login information
        string country = "";
        if (HttpContext.Current.Session["country"] != null)
        {
            country = HttpContext.Current.Session["country"].ToString();
        }
        return s_CheeseFileNameRoot + country + ".xml";
    }
}
```

3. Try logging in as adminFR / admin and adminUK / admin, and then view the catalog page.

## Part 3: Using application-wide state on the server

ASP.NET *Session* is useful for storing information specific to one user's session, but what if there is data that can be shared among all users? In this part, you will explore using the *ASP.NET Cache* for this purpose.

*Cache* is like *Session* in that it is a key/value dictionary of objects stored on the server, but it is different in that all users of the system have the same view of *Cache*, whereas they each have their own view of *Session*.

Currently, when the *CustomMembershipProvider* or *CustomRoleProvider* accesses a *BigMouseUser*, the entire *Users.xml* file is loaded and scanned. An alternative to this is to store the user in *Cache*. This way, repeated accesses (such as those checking authorization) will not require that the entire XML file is loaded repeatedly.

1. Add the .NET reference *System.Web* to your *Epic.Training.Example.Security.Web* project.
2. Instead of directly loading the user list, determine if *username* is in the cache. If so, use that instance of *BigMouseUser*. Otherwise, read from *UserList* and place that user in the cache.

```
public override bool ValidateUser(string username, string password)
{
    BigMouseUser aUser = null;

    if (HttpContext.Current.Cache[username] is BigMouseUser)
    {
        aUser = (BigMouseUser)HttpContext.Current.Cache[username];
    }
    else
    {
        List<BigMouseUser> users =
            XmlLoader<UserList>.LoadItems(UserList.FileName).Users;
        foreach (BigMouseUser u in users)
        {
            if (u.Name == username)
            {
                aUser = u;
                HttpContext.Current.Cache[username] = aUser;
                break;
            }
        }
    }

    if (aUser != null)
    {
        return aUser.Password == password;
    }
    return false;
}
```

3. Place a breakpoint on the line where the user is retrieved from *Cache*.

4. Run your site and log in, log out and log back in as the same user. Did the user come from cache the second time?
5. Next, without closing the web application, open *Users.xml*, change the password of the user, log out, and then try to log in with the new password.
6. Is the new password active? If not, when will the new password become active?

■ \_\_\_\_\_  
\_\_\_\_\_

The user cache should become invalid if *Users.xml* is altered. To do this, add a *CacheDependency* on the data file.

7. Stop the web application.
8. Comment out the line that inserts the user into *Cache*.
9. Replace it with the following code:

```
HttpContext.Current.Cache.Insert(  
    username, // The key  
    aUser, // The value  
    // File dependency  
    new CacheDependency(UserList.FileName)  
) ;
```

10. Start your application and log in using the new password.
11. Log out and change the password back to the original one.
12. Log back in with the old password. It should work right away this time.

## Wrap Up

---

Fill out the following table for each type of state:

Type	Location (client / server / database)	Scope (application / user)
Session	server	user
Cache		
Hidden elements		

Type	Location (client / server / database)	Scope (application / user)
Data files		
Chronicles		
Query strings		
Static variables		
Cookies		

© 2016 Epic Systems Corporation. All rights reserved. PROPRIETARY INFORMATION - This item and its contents may not be accessed, used, modified, reproduced, released, performed, displayed, loaded, stored, distributed, shared, or disclosed except as expressly provided in the Epic agreement pursuant to which you are permitted (if you are permitted) to do so. This item contains trade secrets and commercial information that are privileged, confidential, and exempt from disclosure under the Freedom of Information Act and prohibited from disclosure under the Trade Secrets Act. This item and its contents are "Commercial Items," as that term is defined at 48 C.F.R. § 2.101. After Visit Summary, Analyst, ASAP, Beaker, BedTime, Break-the-Glass, Breeze, Cadence, Canto, Care Elsewhere, Care Everywhere, Charge Router, Chronicles, Clarity, Cogito ergo sum, Cohort, Colleague, Comfort, Community Connect, Country Connect, Cupid, Epic, EpicCare, EpicCare Link, Epicenter, Epic Earth, EpicLink, EpicOnHand, EpicWeb, Good Better Best, Grand Central, Haiku, Healthy People, Healthy Planet, Hyperspace, Identity, IntraConnect, Kaleidoscope, Light Mode, Lucy, MyChart, MyEpic, OpTime, OutReach, Patients Like Mine, Phoenix, Powered by Epic, Prelude, Radar, RedAlert, Region Connect, Resolute, Revenue Guardian, Rover, SmartForms, Sonnet, Stork, Tapestry, Trove, Trusted Partners, Welcome, Willow, Wisdom, With the Patient at Heart and World Connect are registered trademarks, trademarks or service marks of Epic Systems Corporation in the United States and/or in other countries. Other product or company names referenced herein may be trademarks of their respective owners. U.S. and international patents issued and pending.