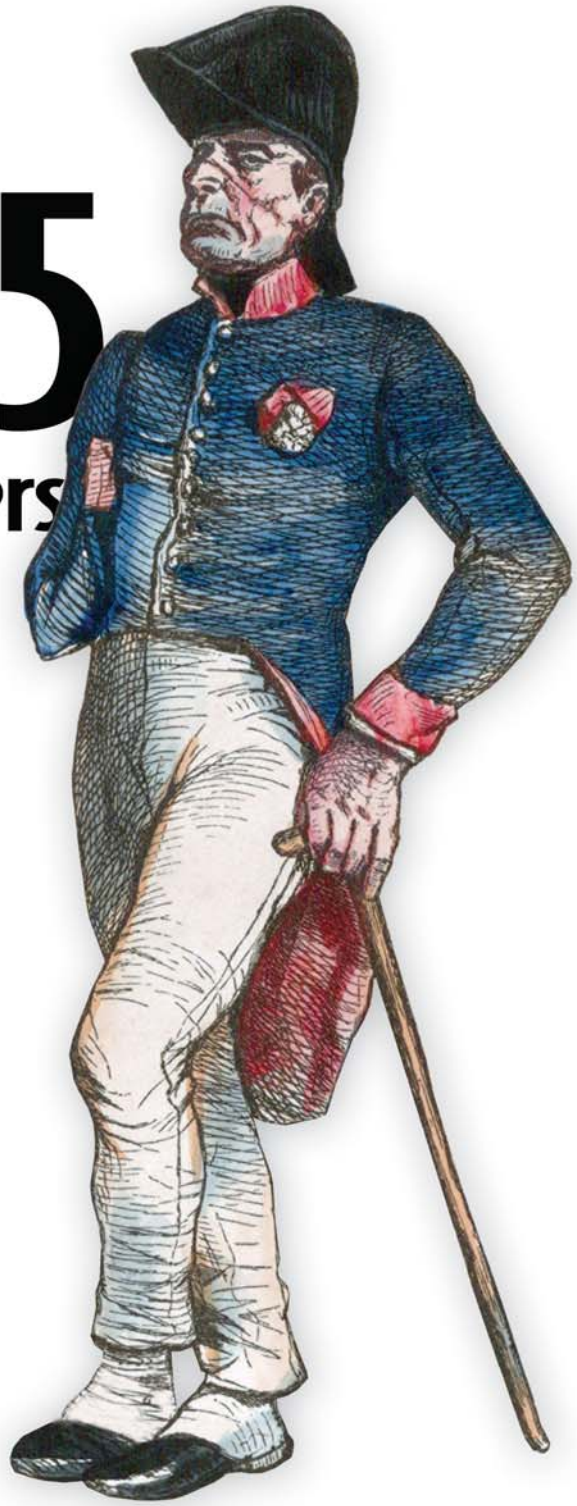
SAMPLE CHAPTER

Single page web apps, JavaScript, and semantic markup

HTML5 for .NET Developers



Jim Jackson II
Ian Gilman

FOREWORD BY
Scott Hanselman

MANNING

HTML5 for .NET Developers

by Jim Jackson II
Ian Gilman

Chapter 1

brief contents

1 HTML5 and .NET

This chapter covers

- Understanding the scope of HTML5
- Touring the new features in HTML5
- Assessing where HTML5 fits in software projects
- Learning what an HTML application is
- Getting started with HTML applications in Visual Studio

You're really going to love HTML5. It's like having a box of brand new toys in front of you when you have nothing else to do but play. Forget pushing the envelope; using HTML5 on the client and .NET on the server gives you the ability to create entirely new envelopes for executing applications inside browsers that just a few years ago would have been difficult to build even as desktop applications. The ability to use the skills you already have to build robust and fault-tolerant .NET solutions for any browser anywhere gives you an advantage in the market that we hope to prove throughout this book.

For instance, with HTML5, you can

- Tap the new Geolocation API to locate your users anywhere on the planet

- Build photo editing or animation products with the Canvas API
- Build high-performance user interfaces for using the History and Drag-and-Drop APIs
- Accomplish a tremendous amount of work with just a few lines of JavaScript

What, exactly, is HTML5? In a nutshell, it's one part semantic organization that can add additional meaning to content on the web and one part JavaScript programming interfaces that allow you to do things in a simple web page that weren't possible just a short time ago. The opportunities are limited only by your imagination, and the tools and environments you're currently using to develop software will probably be the same ones that help you build this new class of application. You can see some examples in figure 1.1.

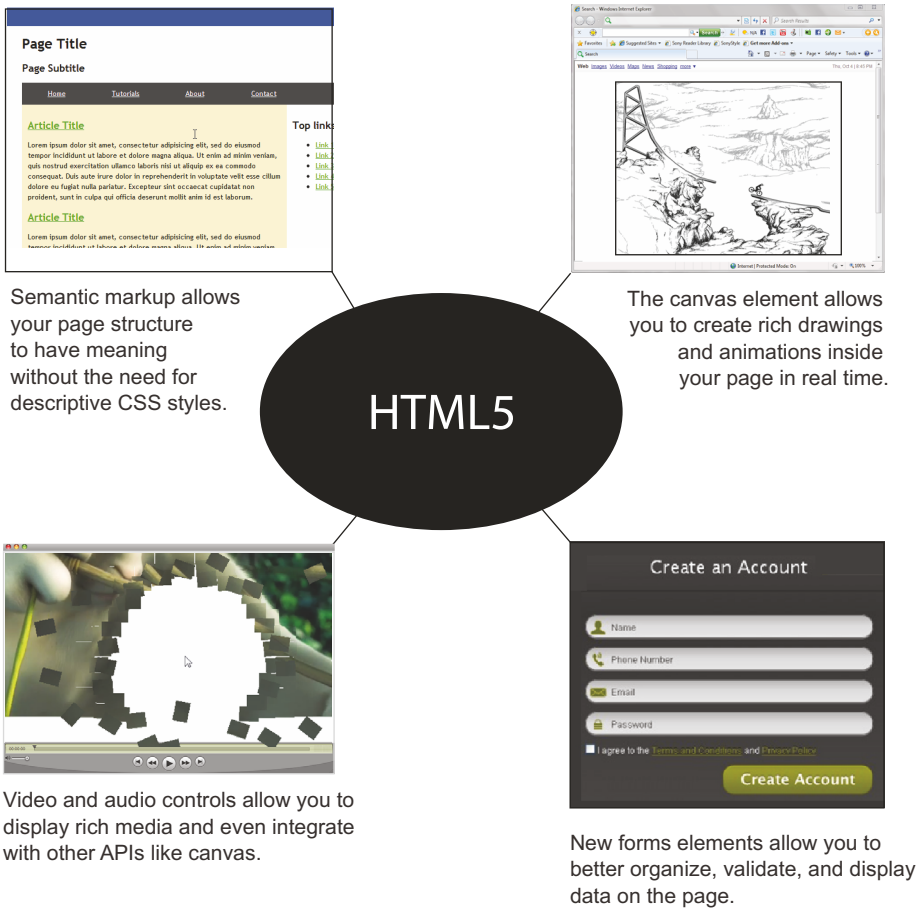


Figure 1.1 From games like Canvas Rider to semantic page layout to audio/video to form presentation, HTML5 has something for everyone in the web design and application space. Rich HTML applications are the new normal for web development.

Fellow developers, now is the time to sit up and take note. The semantic web, which HTML5 taps and which we'll talk more about in the next chapter, is here. Even better, you already have many of the skills you need to build robust applications for this market. The same tools and technologies you use now, like Visual Studio, ASP.NET, and web services, can be effectively integrated into HTML5 applications. You'll need to build on your existing knowledge and expand it into some new areas, but the rewards—such as seamless integration with tablets and phones, ease of deployments and upgrades, and rich client feature sets—are worthwhile.

In this chapter, we'll look at the new toys that HTML5 brings to .NET developers, such as the following:

- New HTML5 elements and microdata, which bring meaning to the markup beyond just the contents of the tags on the page
- New web app form factors that let you add features to your page with little or no additional code
- New JavaScript APIs that not only lead to better performance but also give you the ability to build rich interactive graphics and speed performance in your web apps

We'll also look at JavaScript and why it needs to be a first-class language in your skill set if you intend to take advantage of HTML5, and we'll look at the server-side processes and options for HTML5 available from the .NET framework.

Finally, we'll look at HTML5 applications from end to end, and we'll implement a Hello World example that will give you the minimum JavaScript you need to work through the example applications in this book and will give you a taste of the HTML5 smorgasbord to come.

Without further ado, let's begin with a tour of the new toys that HTML5 adds to your toy box.

1.1 New toys for developers thanks to HTML5

HTML5 is a big topic, and figure 1.2 should give you a better understanding of the various moving parts in a web application that uses HTML5. If it feels like you're looking at the underside of a race car with only a vague idea of how things work, don't worry. We'll provide all the details as we progress through the book. What's important here is the big picture and the basic interactions among the parts.

In this section, we'll give you a high-level but grounded tour of some of the most exciting new features of HTML5, many of which you'll learn how to use in this book. If we won't be covering a particular feature in this book, we'll point you to other good resources on the topic so you can take side trips whenever you need or like. Specifically, we're going to cover the following topics in this section:

- New HTML5 tags and microdata, which help you build search-optimized, semantic pages
- How HTML5 lets you develop across devices and browsers, without having to write multiple programs

- Improvements to JavaScript and the plethora of libraries, extensions, and frameworks that make your development work so much faster and easier
- Identifying and implementing the HTML5 APIs that everyone is talking about by creating user-friendly, graphics-rich, interactive web applications
- Reviewing where Cascading Style Sheets 3 (CSS3) and ASP.NET MVC fit into the picture

For our first stop, we'll turn to HTML5 tags and microdata.

1.1.1 New HTML5 tags and microdata

Imagine that you're a member of a band called Four Parts Water. You're creating a very basic web page just to test out your newly acquired HTML5 knowledge.

You know about HTML tags, which are the little pieces of text inside brackets that you write to render elements on a web page. Each tag starts with an opening < symbol and ends with a closing > symbol. Content is placed next, and then the tag is closed with the `</tag>` marker. Opening tags may also include attributes to give them further meaning:

```
<div>
   <p>My name is Neil.</p>
   <p>My band is called Four Parts Water.</p>
   <p>I am British.</p>
</div>
```

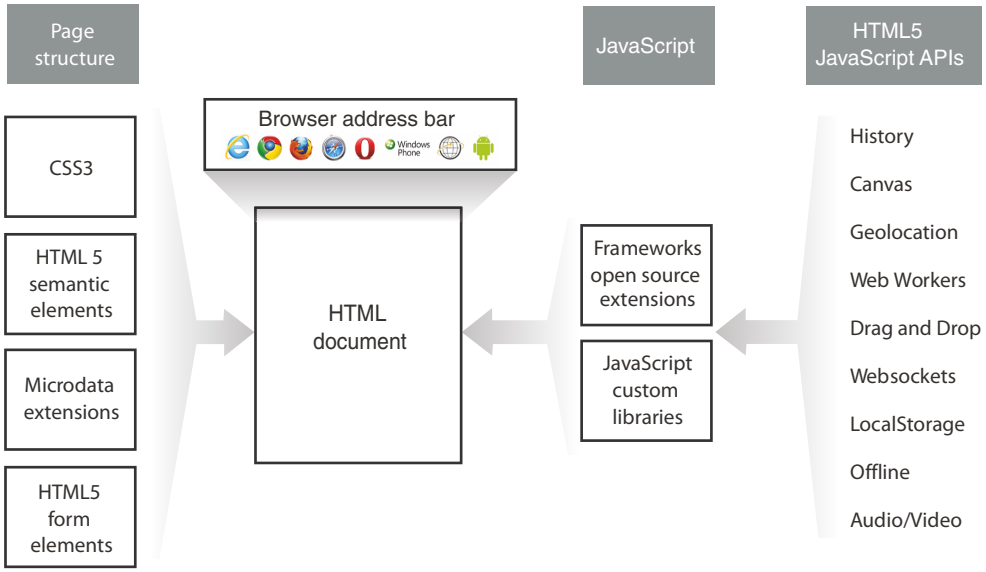


Figure 1.2 The basic organization of a web application built using HTML5. The application is consumed by a web browser that reads an HTML text file and interprets the content, loading other resources like JavaScript files, images, or stylesheets as necessary. The markup is rendered on the page using stylesheets that are linked or placed directly into the markup, and JavaScript code executes at the proper time to change the interface, communicate with the server, or interact with the HTML5 APIs available from the current browser. These APIs can interact directly with the client system, but JavaScript, as a rule, can't.

That's good, but now you want to try adding some microdata. *Microdata* is additional information you can add to your page using special attribute keywords. It can be set, read, and changed via JavaScript, and the values your microdata contains can be nearly anything you like. You can extend tags using microdata to add semantic or other meaningful information that search engines and JavaScript libraries can use to make even more sense of the data on the page. A holistic interpretation of your page data and content will help optimize it for search as well as for accessibility applications like page readers. Microdata extensions can also reduce the amount of code and increase the expressiveness of the markup in nearly any page.

Armed with this knowledge, you write up the code in the following listing (from html5rocks.com), which displays the same basic page with your name and the name of your band, but with extra information meant for web crawlers and search engines.

Listing 1.1 Microdata tags describing content

```
<div itemscope>
   <p>My name is <span itemprop='name'>
      Neil</span>.</p>
   <p>My band is called <span itemprop='band'>
      Four Parts Water</span>.</p>
   <p>I am <span itemprop='nationality'>
      British</span>.</p>
</div>
```

Itemscope declaration defines boundaries of itemprops for object.

Itemprop here is name, standard microdata vocabulary term that's useful for search engines.

Closing tag for element declared with itemscope closes object referenced by microdata.

Band itemprop isn't in standard vocabulary but is allowed nonetheless.

As you can see, the various microdata tags help the engines and crawlers to interpret which pieces of the text are important and what each one means.

1.1.2 HTML5 applications for devices

HTML5 has not only given us .NET developers new ways to make our code make sense on the web; it has also brought us the ability to develop for exciting new devices that used to exist only in the imaginations of sci-fi writers: think iPad, Kindle, and smart phones. Mobile phones have fully featured browsers with display technologies better than most computers available five years ago, and even laptops now have powerful graphics processors. Gaming PCs have graphics support that allows them to seamlessly render complex 3D graphics and animations. HTML5 lets .NET developers enter this new world, where the challenge is to take advantage of the diversity of browser platforms while maintaining functional continuity.

> **NOTE** Currently the web community uses the terms *HTML application* and *HTML5 application* interchangeably. This is because the new functionality that's available as the HTML5 specification comes to market is what is stimulating the new ideas and methods of developing rich internet applications. Here, we'll refer only to "HTML applications," but our examples will be

focused on the parts of HTML5 and JavaScript that make the applications deeper and more useful to users.

How do you develop a single application to work across all the screens listed in figure 1.3? It's certainly possible, but it takes a good understanding of the compromises and features available across the entire range of target browsers. We'll provide you with that knowledge in chapters to come as we teach you how to use HTML5's features in multiple browsers.

1.1.3 Better, faster JavaScript

Another feature that makes HTML applications compelling is the incredible improvement in JavaScript engine performance over the last few years, across all browsers. Gone are the days when JavaScript was only suitable for handling click events or posting forms. Just take a look at figure 1.4 to see how dramatically execution time has improved through various versions.

Add HTML5's native support for JSON data transmission and the array of performance-enhancing coding techniques available, and it gets difficult to say that compiled binary libraries are always faster. While perhaps this is true in many instances, there are plenty of normal operating situations where a JavaScript routine can be just as fast as the same routine compiled in the .NET runtime. This means that plugins like Silverlight and Flash have much less of an advantage in the application market. In some instances, they have no advantage at all.

1.1.4 Libraries, extensions, and frameworks

JavaScript development also benefits from a wide range of open source projects and free tools. While not new toys themselves, these pieces of the application puzzle allow you as the developer to make better, more efficient use of the HTML5-specific toys.

Figure 1.3 The form factor, size, and resolution of browsers available to you is growing all the time.

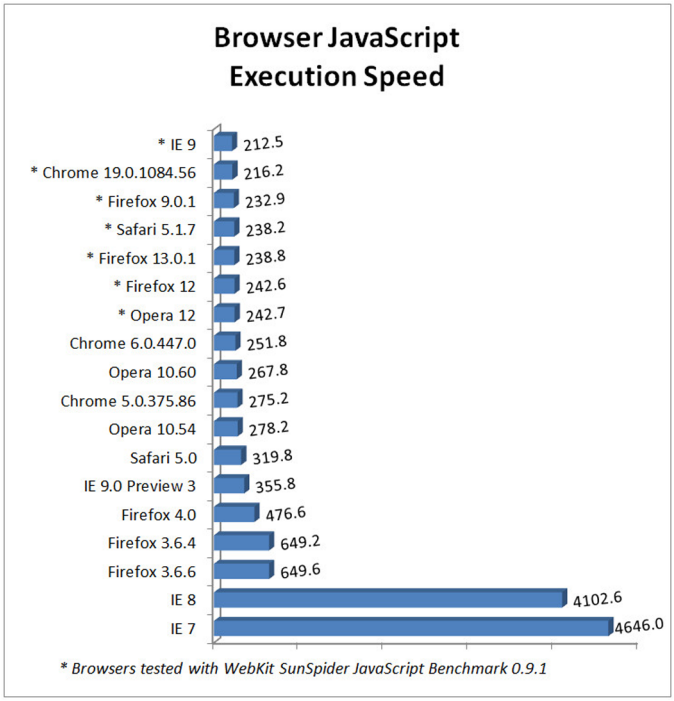


Figure 1.4 JavaScript engine performance improvements in the past few years (courtesy of webkit.org) have led to impressive speeds all around. In this graph, the time required in milliseconds to perform a large number of very specific JavaScript benchmark tasks is measured.

Windows 8

The Windows 8 announcement and subsequent release is big news to all .NET developers. It brings a new set of features, better security, and an app store, and it takes the beautiful Windows Store[1] styling from Windows Phone. While this book isn't specifically about building native Windows 8 applications with HTML5, CSS, and JavaScript, the good news is that what you learn here *will* be applicable on this new platform.

The Windows-specific version of JavaScript is called WinJS, and it's JavaScript at heart with the added ability to call native functions and libraries on the host system. The markup and styling from your HTML5 applications should be relatively easy to port into the new Windows 8 environment, making your skills all the more valuable.

In addition, Internet Explorer 10, shipped with Windows 8, is the most compliant, compatible browser ever from Microsoft, and it's incredibly fast. This gives you the option of building your application as an HTML5 web app to use on multiple devices and browsers or as a native Windows 8 application, suitable for deployment to the app store.

In short, this book, while not targeted toward any specific platform, will allow you to use everything you learn to get a major head start on native and browser-based Windows 8 development.

[1] See the "Roadmap for Windows Store apps using JavaScript" page in the Windows Dev Center at http://msdn.microsoft.com/en-us/library/windows/apps/hh465037.aspx.

For instance, there are dozens of unit-testing frameworks for JavaScript including QUnit, a free framework for JavaScript and jQuery (github.com/jquery/qunit). You can build complete applications using pattern-based approaches with libraries like Backbone.js (documentcloud.github.com/backbone) or Knockout.js (knockoutjs.com). These frameworks give you a client-side MVC (Model-View-Controller) or MVVM (Model-View-ViewModel) paradigm to build large HTML applications while keeping them maintainable. There are thousands more; just think of any feature you might want for a rich website and search for it. You're almost guaranteed to find something to get you started.

It's hard to say exactly where to start when considering third-party commercial and open source JavaScript libraries. There are components for performing specific tasks, libraries that act as development frameworks, libraries for unit testing, graphics helpers, communications tools, documentation enhancers, and plenty of others. Just take a look at GitHub (http://www.github.com/) and see for yourself. A search for "JavaScript" turns up over 9,000 projects. Now jump over to the jQuery site (www.jquery.com) and take a look at the plugins page. There are almost 500 pages of plugin projects.

Wondering where to start when it comes to libraries? Consider jQuery.

jQuery is the obvious place to start when looking at JavaScript libraries to improve the quality of your applications and speed of your development. It's one of the most popular frameworks for developing HTML applications, used in nearly half of all active websites today.[2] The library, a creation of John Resig, is under constant development and is both fast and easy to use. It also sports a plugin model that allows others to add new features to it.

Microsoft clearly understands that jQuery is an ideal tool for building the next wave of applications, and it has invested a lot of energy into data binding, templating plugins, and pattern-based frameworks like Knockout.js. Using HTML5, a Microsoft developer can now build once and deploy practically anywhere. (Where have we heard that before?) But more important than Microsoft's contribution is the fact that it's an equal partner in the jQuery ecosystem.

Nearly every JavaScript library available today is open for your review and for subsequent inclusion in your website based on the license that accompanies it. In addition to using these libraries outright, you can use them to learn how to do specific tasks or for architectural guidance.

As you work through the examples in this book and become more versed in the JavaScript language, you'll learn to look at these libraries with a critical eye toward

[2] See W3Techs "Usage statistics and market share of JQuery for websites" article at http://w3techs.com/technologies/details/js-jquery/all/all.

instancing models, resource allocation, binding to existing elements, and how each library can fit into the overall goals of your application.

1.1.5 New HTML5 JavaScript APIs

There are also various JavaScript objects and APIs that can help your pages interact with the outside world and with the rest of the browser's operating system. There are quite a few such features, but we'll focus our discussion on some of the most stable and useful for building rich web applications.

CANVAS

Canvas is a raster-based drawing mechanism in HTML5. The Canvas JavaScript API has a lot of functionality, and we'll cover it in detail in chapter 4. If you want an early peek though, try using the following code to draw a simple rectangle on a canvas element:

```
var myCanvas = document.getElementById("rectCanvas");
var canvContext = myCanvas.getContext("2d");
canvContext.fillRect(50, 25, 150, 100);
```

The key is to get a reference to the canvas and then grab its context object. The context object is what you use to do all work inside the rendered element.

How can you use it? As a drawing surface, for graphs and charts and for animations ranging from very simple to extremely complex.

HISTORY

The History API in HTML5 is used to add or replace data in the current browser's session history. You can use it to overwrite the current page with something more generic or with a more helpful landing page. You can also use it to add a new item to session history so that on-page navigation events can be accessed using the browser's forward and backward buttons:

```
history.pushState();
history.replaceState();
```

We'll discuss the History API in chapter 5.

How can you use it? To enhance application navigation between views or pages and to remove unwanted steps from the browser history for the current site.

GEOLOCATION

Our favorite API is Geolocation. Using the `geolocation.getCurrentPosition()` function, you can return a latitude and longitude from a device's onboard GPS device. Note that the `geolocation` object is only available to the `navigator` object in JavaScript. `Navigator` isn't, as you might expect, a wrapper just for `geolocation`. It's a global object that contains a number of functional pieces. Check out chapter 6 on geolocation for more on this.

How can you use it? As a tool to let users locate themselves in the world and as the basis for providing meaningful data about points of interest around a user.

WEB WORKERS

A web worker allows your HTML application to use multiple threads. For heavy processing applications or long-running JavaScript tasks, the web worker object can be invaluable. The web worker is declared as a `Worker` object and is passed a JavaScript file:

```
var wrk = new Worker("BackgroundProcess.js");
```

Once instantiated, the background process script and the hosting worker object can listen for messages sent back and forth. The worker object could do this:

```
wrk.postMessage("Hello to the web worker");
```

And inside BackgroundProcess.js, you could do this to send a message back to the host:

```
self.postMessage("Hi from the background process");
```

This is a minimal example without any of the required plumbing code. What's important here is that the values passed back and forth are strings. This leaves open the possibility of sending JSON data objects as well as other more complex arrays of values. We'll cover Web Workers in chapter 7.

How can you use it? To speed application performance by performing processor-intensive calculations in the background, freeing up cycles for graphics rendering and user interaction.

DRAG AND DROP

Drag and drop is a new feature in HTML5 that allows you to programmatically pick up and drop elements on your page relative to the page, to each other, or to the user's desktop. This is done by wiring up events on elements for `drag`, `drop`, `dragover`, and `dragenter`. While a drag operation is occurring, other features of the API can be activated to provide feedback to the user about what is happening. We'll look at drag and drop in chapter 7.

How can you use it? As a means of bringing natural user interactions to web applications reliably and quickly.

WEBSOCKETS

Websockets are a means of breaking away from the request/response paradigm of web page interaction to a bi-directional communication channel. This means that communications can be happening in both directions simultaneously during a session. This is best described with examples, but we need to cover more JavaScript basics first. Look for coverage of Websockets in chapter 8.

How can you use it? For building real-time communication web applications like chat, white boards, or collaborative drawing.

LOCAL STORAGE

The Local Storage HTML5 API provides a solution for storing local data through the use of a key/value style storage specification that's available for reading and writing within a single domain. You can read, insert, update, and delete data very easily and

store much more information than would normally be possible in a web application. We'll cover this API in chapter 9.

How can you use it? As the basis for building applications that store user data locally while sending only the data necessary for server functions.

Local Storage doesn't provide any specification for synchronizing with a server database, nor does it provide transactional support. If you need transactional support, you would be better off looking to the IndexedDB HTML5 specification. This API uses a document-database (or NoSQL) style approach, but the specification is incomplete and unstable at this time, so we won't cover it in this book.

OFFLINE ACCESS

The ability of a site to remain available offline is new in HTML5. It's done by specifying a manifest file that describes which files must be downloaded for use offline, which files should only be accessed while online, and which files, when requested, should get a substitute file instead. The manifest file is specified in the top-level `<html>` element on a page:

```
<html manifest="/cache.manifest">
```

How can you use it? As a means of creating rich games or business applications that function even when an internet connection isn't available.

AUDIO/VIDEO

The Audio and Video tags allow you to play music and video without Flash or Silverlight plugins. Browser vendors have built in their own default players, but you can easily extend or replace them as we'll show in chapter 3. Because support formats vary between browsers, you can create your content in multiple formats and allow the browsers to automatically choose which version to use. This allows for forward and backward compatibility, keeping you current with the ever-changing multimedia format landscape.

A simple audio tag might look something like this:

```
<audio src="/content/music.mp3"></audio>
```

1.1.6 Cascading Style Sheets 3

Cascading Style Sheets (CSS) version 3 technically isn't a part of the HTML5 specification, but the graphics capabilities of media queries and transformations make it a crucial part of any browser-based rich application. Putting your presentation rules into styles allows you to build more manageable and pluggable user interfaces for your clients. Well-engineered cascading styles can also significantly reduce your development time.

We'll cover the core CSS3 concepts necessary for implementing HTML5 applications and understand where CSS3 fits into application design in chapter 2. We'll touch on it again throughout the rest of the book as a means of adding smooth animations and rich styling. While we aren't providing a definitive CSS3 reference in this book, you'll certainly come to realize the benefits of learning CSS more deeply.

The book *Smashing CSS: Professional Techniques for Modern Layout* by Eric Meyer (Smashing Magazine, 2010) is a great addition to any technical library.

1.1.7 MVC and Razor

While not directly part of HTML5, MVC (Model-View-Controller) is a software development pattern that allows for the clear separation of concerns between business logic components and user interface display. The Visual Studio templates for Microsoft's latest version of ASP.NET MVC are being constantly updated as free, out-of-band releases directly to the development community. ASP.NET MVC presents a couple of ways to operate in the context of an HTML application.

The first and easiest way is to ensure that all your views are HTML5 compliant. This can be done online at sites like validator.w3.org that allow you to enter a URL and return a listing of valid and invalid markup. This includes the semantic organization of your markup and the use of unobtrusive JavaScript (discussed shortly). You can also build a single HTML page to contain an entire piece of your application and include it in your MVC site. We'll do this in chapter 4, when we cover HTML5 Canvas.

The next method is to use Razor, the view-processing engine that was introduced as part of ASP.NET MVC. Razor facilitates readable inline code within your views, allowing you to write properly formatted HTML with bits of server code interspersed to perform work based on data models that you can build. Using Razor, your markup becomes more terse, easier to read, and faster to code. Using Razor and ASP.NET MVC, you can incorporate all the features of .NET development that you're accustomed to and transition seamlessly into the world of HTML5 application development. Razor is used in our MVC views throughout this book and it's covered in more detail in appendix B on ASP.NET MVC.

1.2 HTML5 applications end-to-end

Now that you have a basic understanding of the toys you'll get to learn about and play with in this book, the next thing you need to know is how each piece interacts with the next and where they touch each other in a normal system.

> **NOTE** At the beginning of each chapter, we'll clearly define which browsers and versions are supported. You should be able to download, install, and test with Google Chrome, Internet Explorer, Firefox, Opera, and Apple Safari. In addition, you can use any mobile browser at your disposal to test site rendering and function.

Figure 1.5 shows a very simplified view of where each part can fit into the overall scheme of an HTML5 application. This is the same diagram you saw in figure 1.2 but with the addition of Microsoft's server-side components. This is by no means the only way these parts can fit together, but it will get you started.

On the server side of an HTML application, MVC controllers will present a view (HTML text sent to the browser), take data from a form POST operation, or send or receive data using Ajax calls. In later chapters, we'll cover all of these communications and how to integrate them in an HTML application.

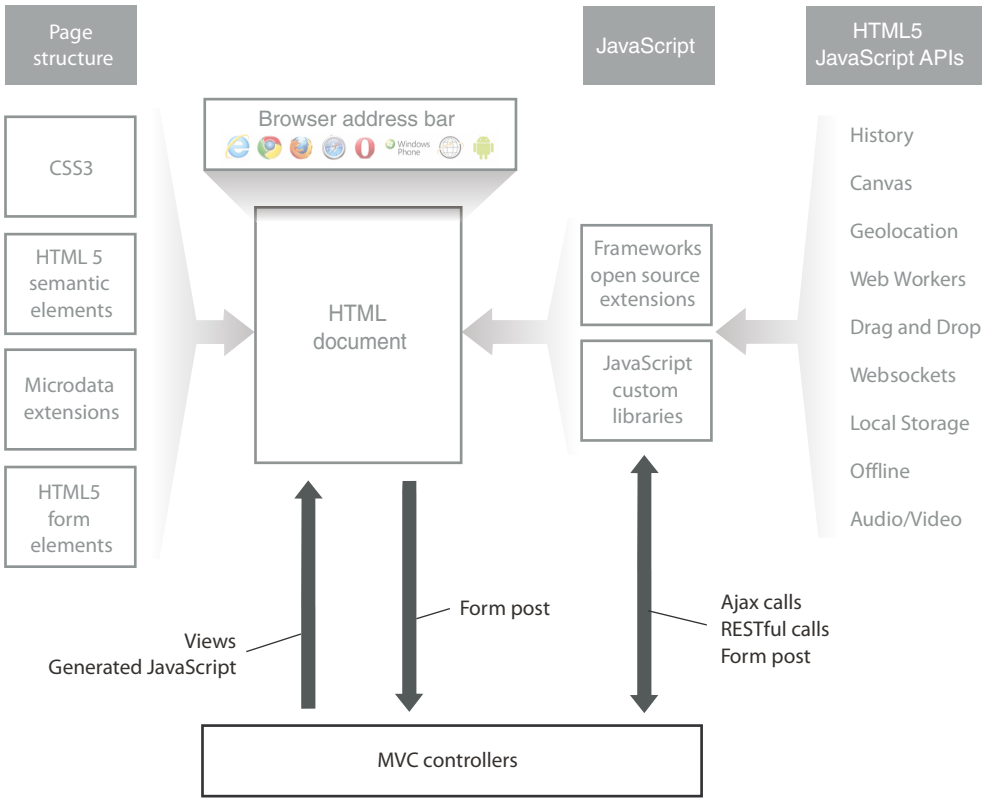


Figure 1.5 Basic client and server interactions between HTML5 features and JavaScript APIs within an application

You might find all these pieces a little overwhelming, so we'll dig a little deeper into each area to help firm up your understanding. We'll start with the page structure.

1.2.1 Page structure and page presentation

Figure 1.6 identifies the *page structure* and where it fits into the scheme of an HTML application.

The structure of a single application page consists of the semantic elements, such as `<header>`, `<footer>`, `<nav>`, `<article>`, and `<section>`, as well as any traditional HTML tags, like `<div>`, `<span>`, and `<a>`. Semantic tags, which will be covered in more detail in the next chapter, provide organizational cues and a means of denoting where various parts of the content will exist. Structural elements receive styling using CSS and can have JavaScript behaviors attached at runtime. Elements in the page structure can be delivered from the server at runtime, built from templates on the client, or downloaded on demand.

Note that the styles that a page uses can also determine its structure. A common instance of this is when an element is *floated*. Floated elements (denoted by the CSS

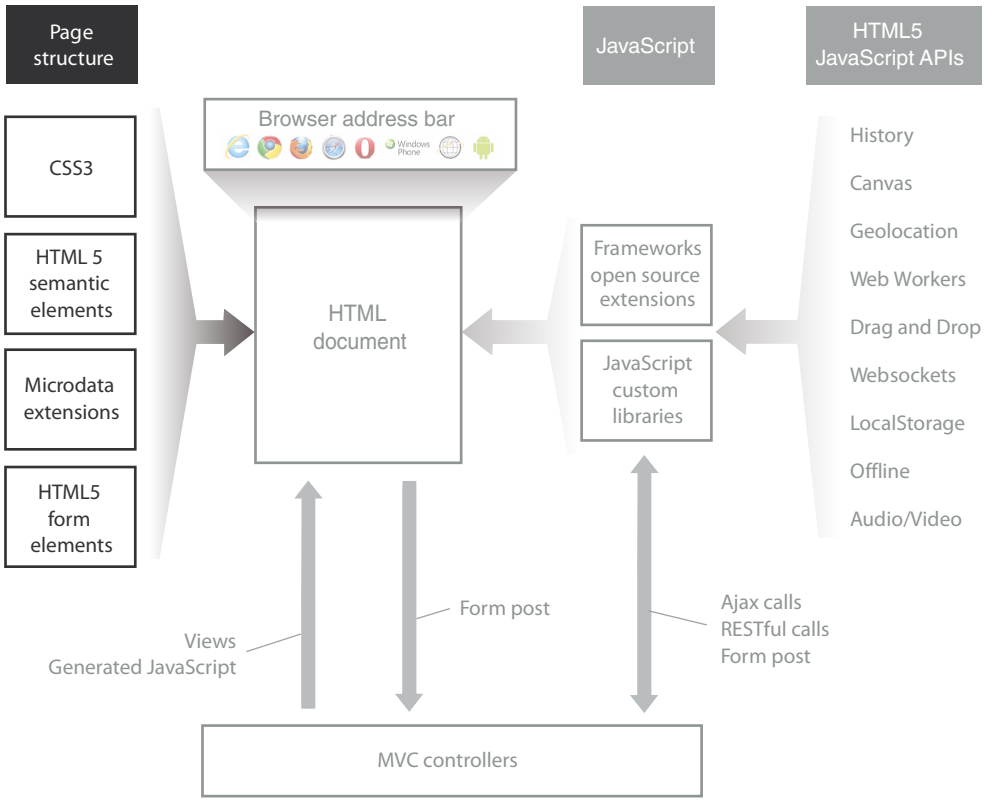


Figure 1.6 Page structure is the physical organization of an HTML page. Which tags exist inside other tags can determine how elements can be moved or accessed using JavaScript.

style `float:left` or `float:right`) don't participate in page flow but will dock themselves to the appropriate side of the window. We'll discuss positioning elements on the page when we look at the Canvas API in chapter 4 and the Geolocation API in chapter 6.

Page presentation is the visual styling that a page structure receives, based on the location of elements in the structure and the stylesheets included on the page. Styles in a stylesheet are the starting point for operations that can occur at runtime. While the page is displayed, changes to the browser layout can trigger media query changes, and interactions by the user can trigger JavaScript functions. We'll cover what media queries are and how they work in chapter 2. For now, the important concept is that by using CSS and JavaScript, you can dramatically change the presentation of the page based on changing conditions in the browser.

1.2.2 Page content

The content of your application can be anything from a map to an editable grid. It can be data from a content management system, pictures uploaded by a user, or news articles. Whatever the content, it's the most important part of your application, and it should be

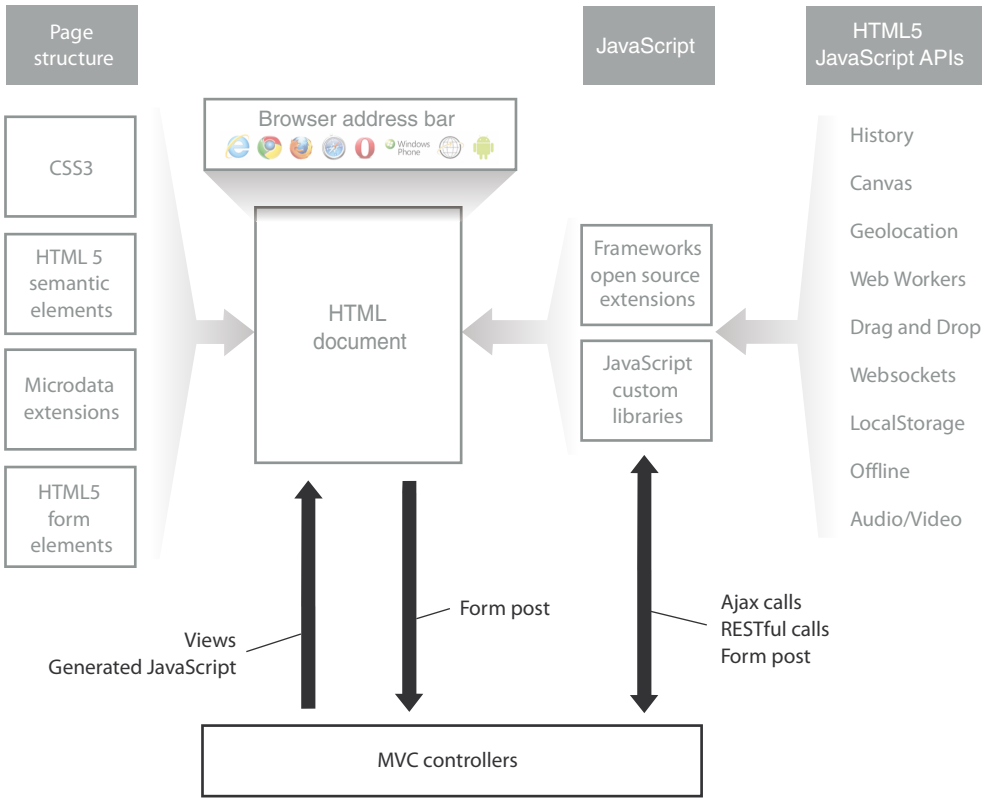


Figure 1.7 Static content is written directly inside the HTML elements in a page. Dynamic content can be delivered to the browser in an MVC application by means of views or via JavaScript and Ajax.

placed in the structure in a way that makes it very obvious what it is and why it's important. Figure 1.7 shows the role that content plays in the HTML application scheme.

Page content can be static, dynamic, or a mix of both depending upon the needs of the application. It can be added by the user while the application executes or be pulled on demand when the application detects updates from some other process.

1.2.3 Application navigation

In HTML5 applications, there are two parts related to navigation: manipulation of the browser URL and posting of values to a server to move to another page. Figure 1.8 highlights the POST operations at the bottom of the diagram and the use of the new HTML5 History API to manage the URL.

Navigation can occur when a user clicks a link to another page or submits a form, or it can be initiated via JavaScript by some other event. In traditional web pages, these operations were abrupt and sometimes jarring, but in a rich HTML application, a user's actions can be considered and handled gracefully. Natural or instinctive interactions are

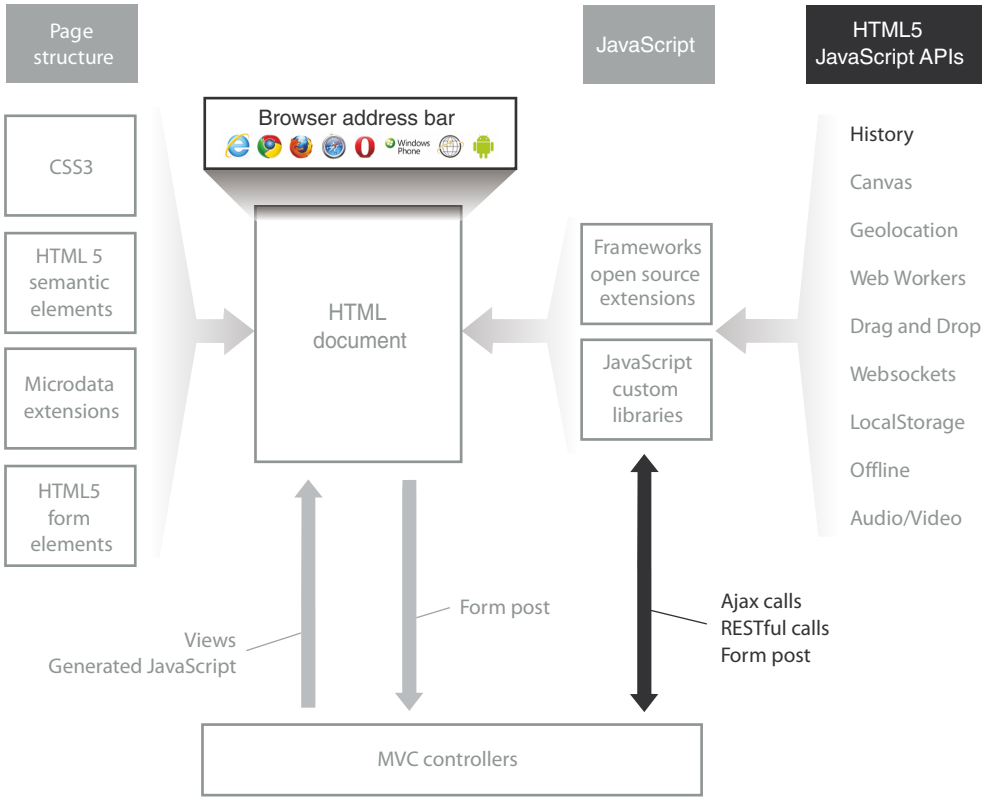


Figure 1.8 Application navigation can happen when a user POSTs a message to the server, when a JavaScript event occurs, or when browser URL changes are intercepted with the HTML5 History API.

an area gaining a lot of traction in the mobile market today because what may seem like small parts of the usability story can have a large effect on user satisfaction. Keeping operations subtle and instinctive is an art form where the ability to draw the eye, the mouse, or the hand to a specific place to perform an operation is critical.

1.2.4 Business logic

The business logic in an HTML application will nearly always be JavaScript on the client; the corresponding server-side implementations can be .NET or any other server technology. As shown in figure 1.9, the custom libraries and frameworks you include in your application will be responsible for changing the user interface, communicating with the server, and integrating HTML5 APIs.

On the communication side, we'll use ASP.NET MVC in this book but you aren't limited to this technology. Any server solution capable of receiving HTTP calls and returning data will work. The decisions you'll have to make will revolve around how, when, and where to validate your business data and which external libraries to use.

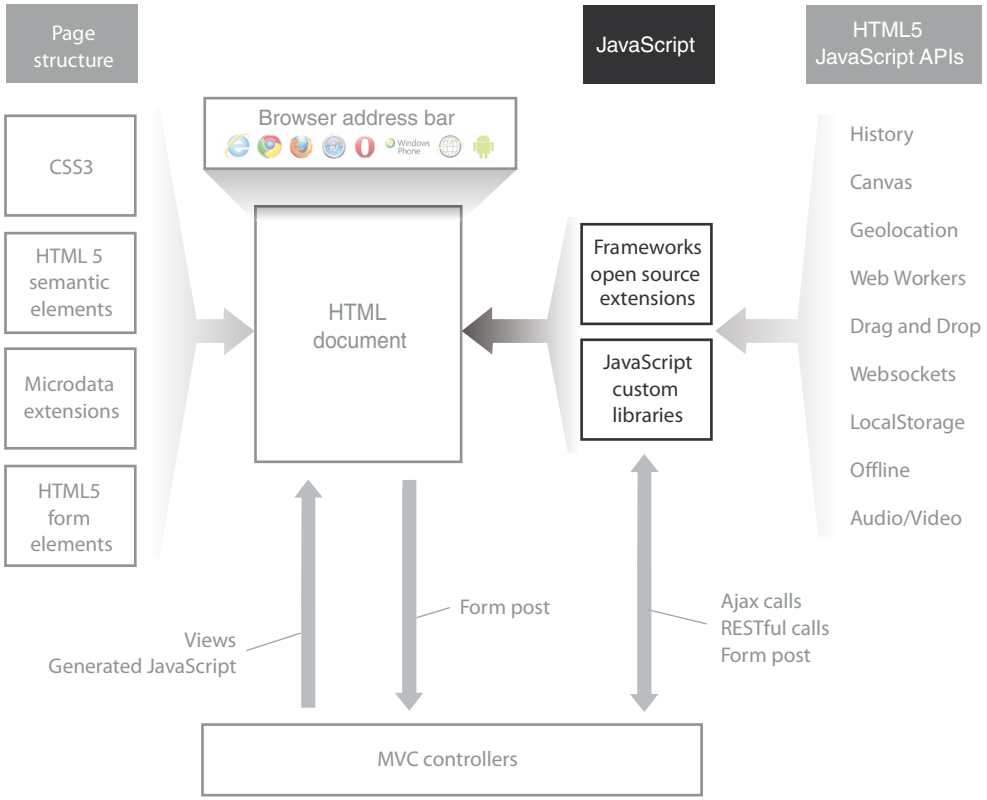


Figure 1.9 The business logic in an HTML application resides almost exclusively in JavaScript on the client and on the server in .NET libraries.

1.2.5 Server communications

Communication with the server is accomplished via the initial load of a page, by the posting of a form, or via Ajax calls to web services. A good communication model will keep the traffic frequency low and the content volume to the barest minimum. Figure 1.10 highlights a limitation in ASP.NET MVC whereby all communications will be transmitted through controllers and can be initiated by either a JavaScript Ajax call or through a form POST.

When security is necessary, SSL is available using MVC to keep your data transmissions private, and when security isn't required, it sometimes makes sense to make communications with the server somewhat transparent. Doing this will enable your system to operate in a software-as-a-service (SAAS) model and allow other applications to consume or manipulate your application's data.

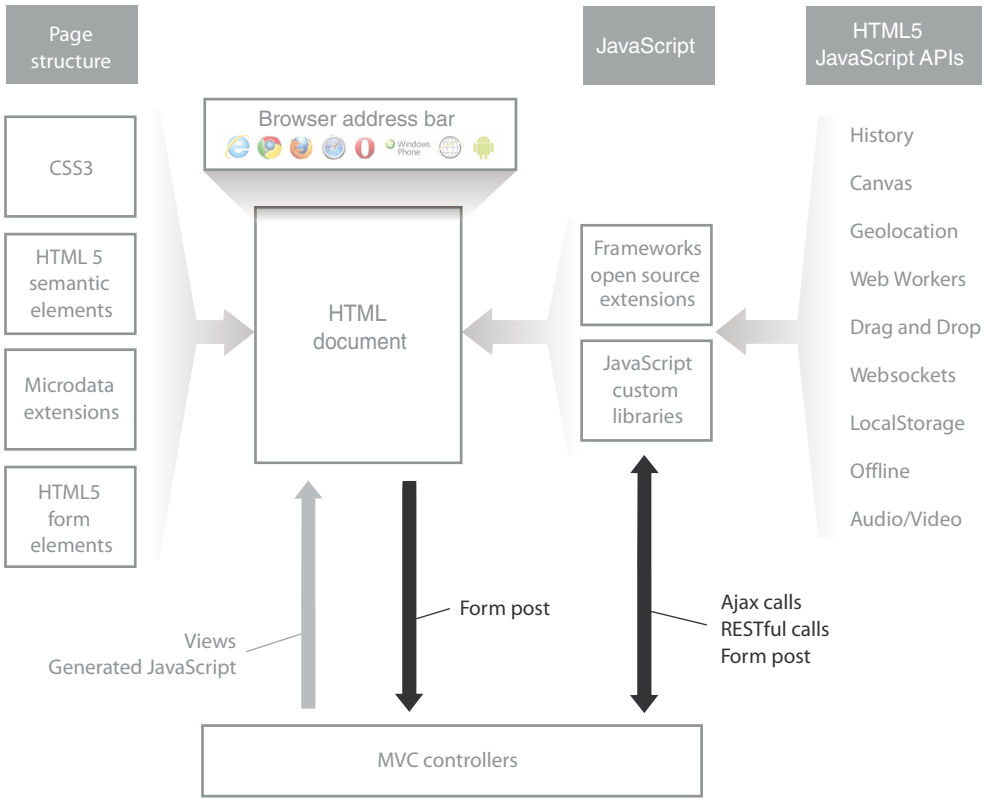


Figure 1.10 Communication with servers is vital to most business applications and can be performed by contacting controller actions using Ajax or by POSTing forms to the server.

1.2.6 The data layer

The data layer is interesting in an HTML application because it can involve both client and server data storage. On the server, you'll store all the normal business data, security, and transactional information. The client is more complex, because you'll often need to store durable state information as well as client data for use offline. Figure 1.11 shows the intersection of JavaScript with the Local Storage API.

Local Storage, as mentioned earlier, isn't the only means of maintaining data on the client, but it currently has the best mix of supported browsers and simplicity in usage. Local Storage has far more support and stability than IndexedDB and is far easier to use than browser cookies, though these other solutions have their place.

Whether or not a page set up to be accessed offline could also be considered part of the data layer is an interesting topic that we'll give some consideration to in chapter 10, but, for now, understand that certain directives placed in your page will allow it to be seamlessly accessed when the browser isn't connected to the internet.

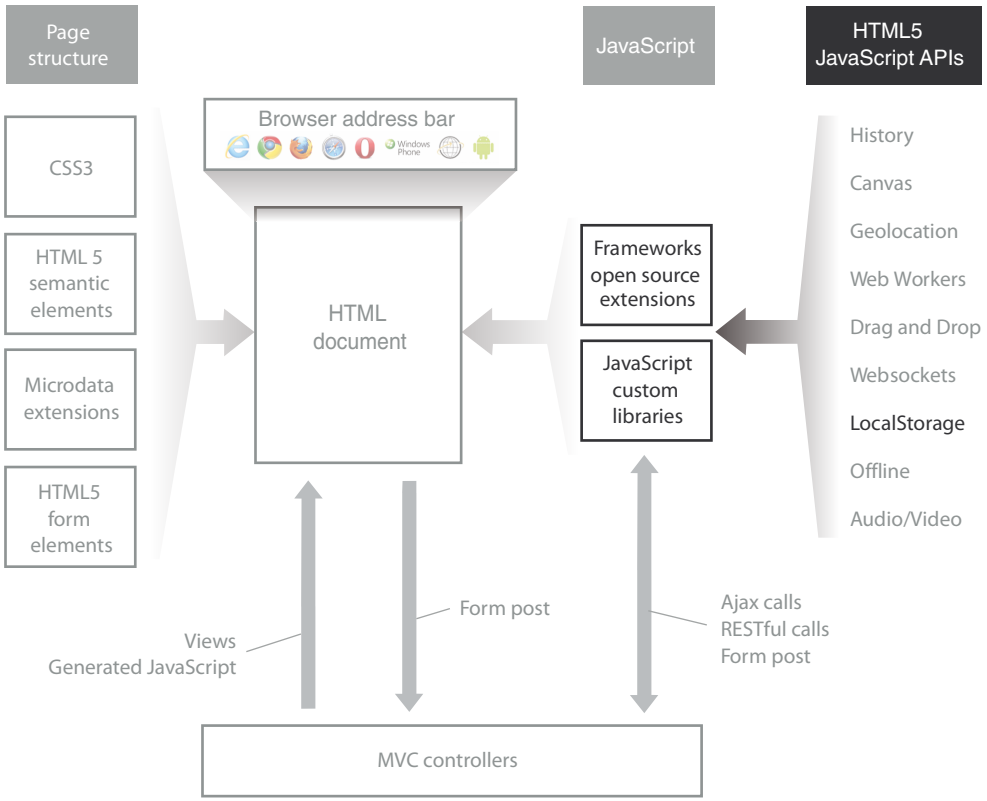


Figure 1.11 Storage of local data inside an HTML application is best accomplished using the Local Storage API and JavaScript methods to maintain it.

Now that we've looked at many of the facets of developing applications using HTML5, JavaScript, and .NET, it's time to put it all together by building a Hello World application.

1.3 Hello World in HTML5

Your Hello World application will, in just a few lines of code, display a web page, create a JavaScript object, and get JSON data from an MVC controller to present to the user. Figure 1.12 shows that this application will work in all major browsers. It will also work on iPhone, iPad, Windows Phone, and Android devices with no modifications!

When a user enters a name and a date, the server will add a "verified" tag to the name and validate the date passed in. It will assign a server date to the returned object that will then be displayed in the interface. No additional navigation will be required by the user to perform any of these steps.

This application won't only let you get your hands dirty right away; it will also introduce you to the following features that will be useful immediately and throughout the rest of the book:

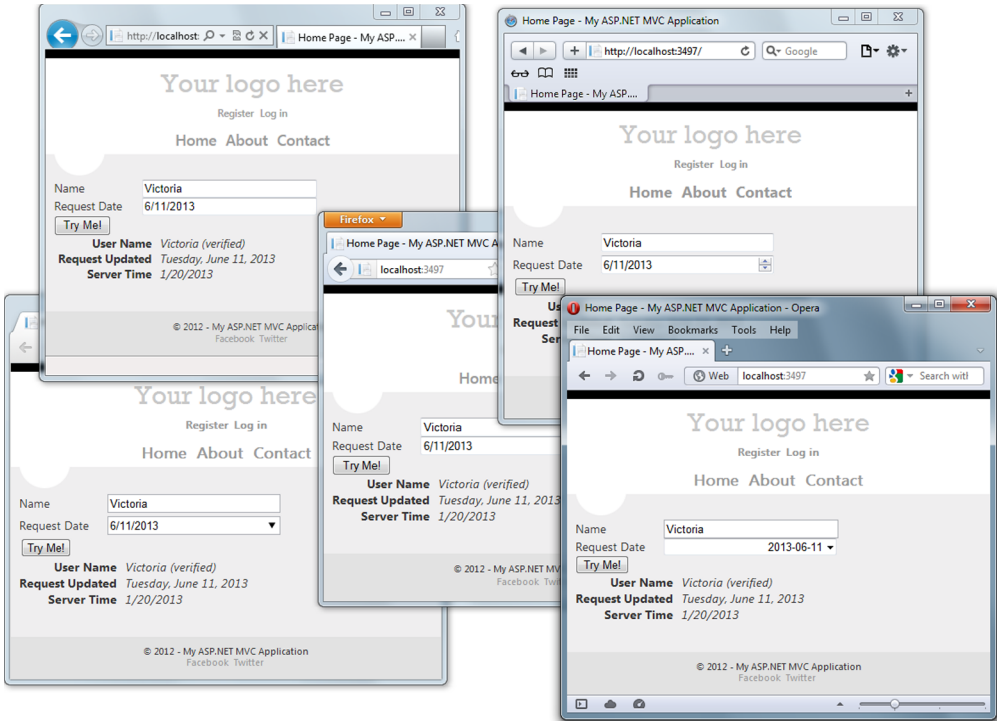


Figure 1.12 Hello World executing with no modifications in all the popular browsers

- Updating or removing Visual Studio NuGet project packages as appropriate
- ASP.NET MVC model binding
- Posting data to a server using Ajax and jQuery
- Using JavaScript to add content to and remove it from the page
- Building and using JavaScript objects
- JavaScript closure and scoping
- The jQuery `ready` handler

In the following sections, you'll create a new template, customize the application, build your JavaScript library, and then build what you need for the server side. Time to get started!

1.3.1 Creating the template

To get started, open Visual Studio normally and start a new project called HelloWorld:

1. Select the Web tab and find the project template called ASP.NET MVC Web Application. The version of the MVC project template will depend on the latest version you have installed in your copy of Visual Studio, but for your purposes

you need at least MVC 3. This can be downloaded for free from the Microsoft site or installed via the Web Platform Installer (http://www.microsoft.com/web/downloads/platform.aspx).

2. Set the project name to HelloWorld, as in figure 1.13, and click OK.
3. Select Internet Application using HTML5 Semantic Markup and the Razor View Engine.
4. Leave the Create Unit Test Project item unchecked.

The template will create a baseline MVC website that you can fire up immediately. You should have folders containing a list of controllers, folders for views, models and scripts, and the web.config file, which is probably familiar to you. In the Scripts folder you'll notice that there are quite a few files that appear to contain version numbers in the filenames. This is a common practice in JavaScript libraries. If you have some familiarity with jQuery, you'll probably also notice, as shown in figure 1.14, that the files are out of date.

The inclusion of NuGet, an open source project started by Phil Haack, in Visual Studio can make updating these files quick and painless. In the Visual Studio menu bar, select Tools > Library Package Manager > Manage NuGet Packages for Solution. In the Manage NuGet Packages window, you can select the Updates tab on the left and see a screen similar to figure 1.15.

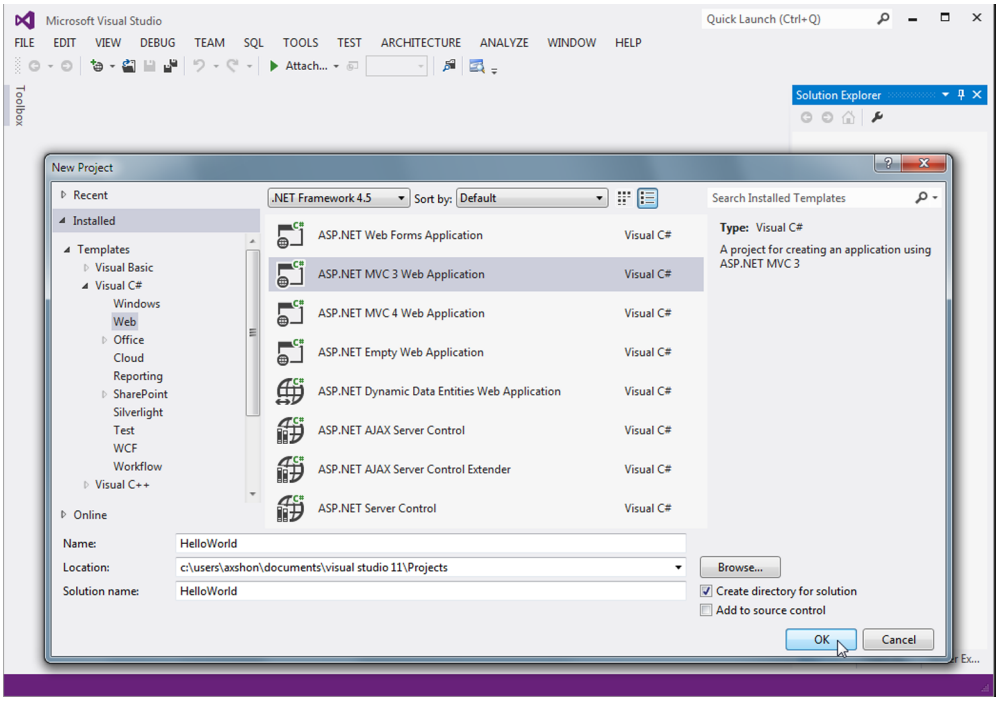


Figure 1.13 Create a new ASP.NET MVC project in Visual Studio using at least MVC 3.

All you need to do is click the Update button next to each package (some are linked, so updates can cascade) until everything is finished. You should now have all the latest JavaScript libraries and project references to continue with the HelloWorld application. These files aren't necessarily linked in the appropriate places, but you'll update those links as you encounter them. In later chapters, we'll only include and update the packages you'll actually be using, but for this exercise we'll keep things a little more simple.

> **TIP** This section covers a number of topics that are specific to the JavaScript language. If you aren't already familiar with the constructs and features of JavaScript, we recommend you read through appendix A on JavaScript at the end of this book.

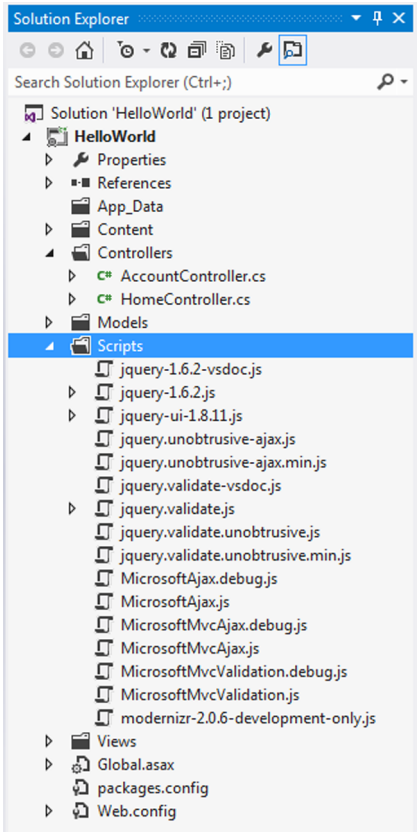


Figure 1.14 The starting MVC project usually contains files that are somewhat out of date. Using NuGet, you can refresh these files very quickly to the latest versions.

1.3.2 Customizing the application

The first step to customizing your application is to modify the user interface. Open the Views > Home > Index.cshtml file and add the markup from the next listing.

Listing 1.2 The Index.cshtml markup

```
@{ ViewBag.Title = "Home Page"; }
<title>@ViewBag.Message</title>
<article id="inputSection">
   <section class="submission">
      <label for="userName">Name</label>
      <input type="text" id="userName" />
   </section>
   <section class="submission">
      <label for="reqDate">Request Date</label>
      <input type="date" id="reqDate" />
   </section>
   <section>
      <button id="makeRequest" type="button">
      Try Me!</button>
```

<article> will be container for area where you'll work.

Various <section> elements will divide up working area.

<button> will be bound to click event handler in JavaScript using its id value.

```
    </section>
    <section id="outputSection">
    </section>
</article>
<script src="/Scripts/HelloWorld.js" type="text/javascript">
</script>
```

Final <section> will be filled with data returned by server after successful callback.

The markup will automatically be placed inside the master page of your application by MVC. If you're using version 3 of MVC, the master page file will still contain references to the old files. This is the file you need to open next:

1. Navigate in the Solution Explorer to Views > Shared > _Layout.cshtml. Notice at the top of the page that you have references to various script files. Compare those references to what is in the Scripts folder of the application and update them accordingly.

```
<script src="@Url.Content("~/Scripts/jquery-1.7.2.min.js")" ...
<script src="@Url.Content("~/Scripts/modernizr-2.5.3.js")" ...
```

2. While you're here, find the `<h1>` tag and change its contents to `"Hello World"`. Run your application now, and you should see something similar to figure 1.16.
3. This is fine but it could use some improvement. Find the Content > Site.css stylesheet and open it up. Scroll to the bottom and add the following styles:

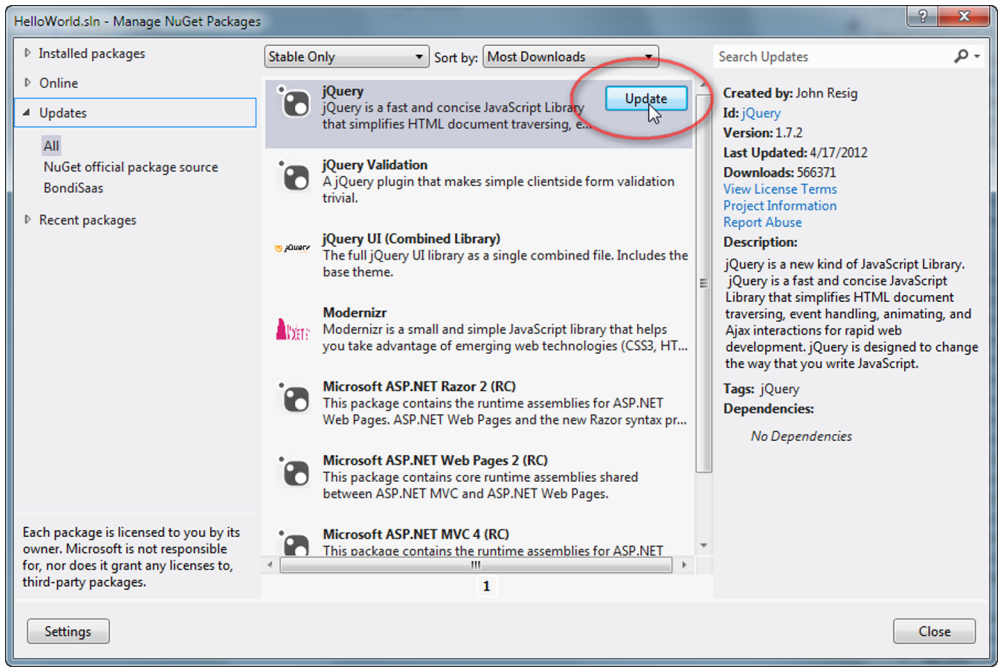


Figure 1.15 Use NuGet to update all the packages that are included by default in the standard MVC solution.

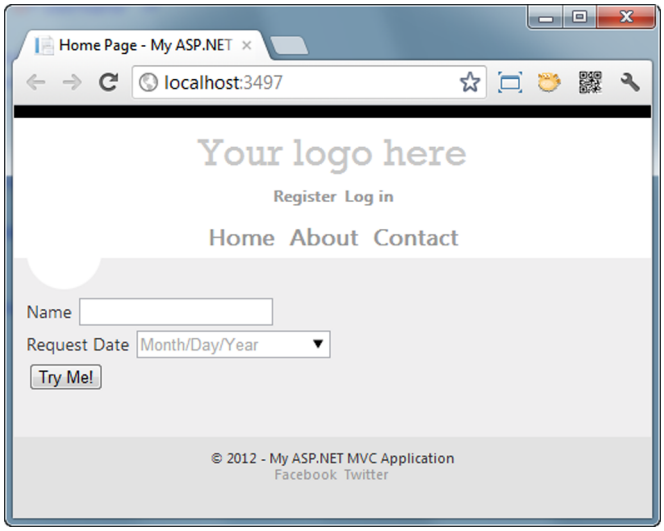


Figure 1.16 The content of the home page has the controls you need for the application, but it still needs additional styling.

```
.submission label {
    display: inline-block;
    width: 100px;
}
.submission input {
    width: 200px;
}
.result label {
    display: inline-block;
    margin-right: 10px;
    width: 115px;
    text-align: right;
    font-weight: bold;
}
.result span {
    font-style: italic;
}
```

These styles will keep things lined up and pretty later on, when you're moving data back and forth between the client and server and dynamically adding and removing HTML elements. The styles all take the same basic selector (the heading for each style that determines which elements will be selected). There are plenty of ways to write styles to get the work done, but we're being very specific with these styles so as not to inadvertently edit other styles in other parts of the page.

1.3.3 Building the JavaScript library

You may have noticed in listing 1.2 that we referenced a script file named HelloWorld.js. It's time to create that file. To do this, we'll walk through the following steps:

1. Create the JavaScript file and wire up the jQuery `ready` function.
2. Create the `myApp` object.

3 Create the Ajax request to call the MVC controller.
4 Create the JavaScript function to handle the results from the MVC controller.
5 Create the function that displays results on the page.

Let's get started.

Expand the Scripts folder and add a new JScript file. At the top of this file, add the following bit of JavaScript:

```
$(document).ready(function () {
   myApp.helloWorldWireup();
});
```

This is known as the `ready` handler. It's an event thrown automatically by jQuery when any page that contains the jQuery library reference completes all of its loading and page layout tasks. This isn't necessarily an easy thing to know, so the jQuery team went to great lengths to check multiple sources of information to infer this state of readiness. All you need to do as a developer is wire up the event and you're good to go!

The call to `myApp.helloWorldWireup` doesn't do anything yet, so you'll need to look at that next. The following listing has the declaration for the object, along with a bit of logic to get you started. You'll fill in the stubbed out functions shortly.

Listing 1.3 The myApp object and its functions

```
var myApp = {
   helloWorldWireup: function () {
      $("#makeRequest").click(function (event) {
         var nm = $("#userName").val();
         var dt = $("#reqDate").val();
         var myData = {
               UserName: nm,
               RequestedDate: dt
         };
         // Ajax request will go here
      });
   },
   processResult: function (returnedData) {
   },
   displayResult: function (label, value) {
   }
}
```

Declare myApp object using var keyword. Object is immediately attached to window object.

First function declaration is one you called from ready.

Using jQuery selector, find makeRequest button and bind click event handler to it.

Inside click event handler find userName and reqDate input boxes and extract values using jQuery.

Create a new temporary object called myData that contains the properties you'll send to the server.

Once you call server processResult function will be called. Stub will be filled in shortly.

displayResult function allows you to segregate code that changes user interface in your object.

Your JavaScript object (`myApp`) will be created as soon as the JavaScript file is loaded and before the `ready` event fires, so you can be sure it exists when you call it. This is the normal flow in an HTML application, regardless of how many JavaScript libraries you're loading.

JavaScript object creation

There are various ways to create an object using JavaScript code. You'll be using a number of them throughout this book, but the simplest is as follows:

```
var myObject = {
   prop: "prop 1",
   prop2: "property 2",
   func1: function() {
      alert('hi there!');
   }
};
```

This tells the JavaScript engine to add a reference in memory to the window object and call it `myObject`. This object will have two properties (`prop` and `prop2`) that are prepopulated with values and a function (`func1`) that, when called, will pop up a message box in the browser. The significant rules for this kind of object are

- Use `var` to declare the object
- Separate the property or function name from the value with a colon
- Separate properties and functions with a comma

Another way to create the same object is as follows:

```
var myObject = {};
myObject.prop = 'prop 1';
myObject["prop2"] = 'property 2';
myObject.func1 = function() {
   alert('hi there!');
};
```

The functionality of this second object is exactly the same as the previous one. The only difference is in the way it's instantiated. The advantage of this method is that each addition of a property or function is independent of the others. This means you can add new properties and functions to your JavaScript objects whenever you like using either the dot notation (`object.property`) or the string notation (`object['property']`).

Two other ways of creating objects are to parse JSON text into objects and to copy one object and/or its properties to another object.

The next step is to fill in the Ajax request. jQuery has a built-in function to do this. It can take a number of different optional properties when executed, but for the purposes of this example, you need only the kind of request to execute (the `type`), the `url` to call, the content type, the `data` to pass, and the function that will be executed when the call succeeds (`success`). Each of these is a property that's assigned in the same manner as the `myApp` object from listing 1.3. Place the code from the next listing in the commented section of the `makeRequest` `click` event handler.

Listing 1.4 The Ajax request that will call the MVC controller

```
$.ajax({
   type: "GET",
   url: "/Home/GetMessage",
   data: myData,
   contentType: "application/json",
   success: myApp.processResult
});
```

Home/GetMessage URL will be filled in shortly and takes data object created earlier in click handler.

Ensure that request headers pass "json" as correct data type.

When Ajax call returns successfully myApp.processResult will automatically be called with data returned from the server.

When the server is called with the data payload you created earlier, your MVC controller code will update the name, verify that the date passed in is indeed a valid date, and add a new server date property.

With that data in hand from the successful execution of the Ajax call, you can fill in the `processResult` function. The following listing shows that code.

Listing 1.5 processResult is called automatically when the Ajax call returns successfully

```
processResult: function (returnedData) {
   $("#outputSection section").remove();
   myApp.displayResult(
      "User Name",
      returnedData.UserName);
   myApp.displayResult(
      "Request Updated",
      returnedData.RequestedDate);
   myApp.displayResult(
      "Server Time",
      returnedData.ServerDate);
},
```

Use jQuery to find element with ID of outputSection and clear all contents.

Call displayResult function for each property in data object returned from Ajax call.

Your client-side code is nearly finished. All you need to do is fill in the `displayResult` function. This function takes a label and a value, concatenates them into a series of HTML elements, and then places them inside the recently cleared out `outputSection` element. The next listing shows how it works.

Listing 1.6 Adding elements to the page using jQuery and string-based HTML

```
displayResult: function (label, value) {
   var start = "<section class='result'><label>";
   var mid = "</label><span>";
   var end = "</span></section>";
   $("#outputSection")
      .append(start + label + mid + value + end);
}
```

Creating elements using string concatenation is simple and objects created will be styled and laid out automatically.

Using jQuery's append function find outputSection object and insert string contents as HTML elements.

You can run your application now, and all your code will execute. The only problem will be that calls to Home/GetMessage will fail because you haven't implemented that endpoint yet. Next stop, the server!

> **TIP** During normal operations, most modern web browsers won't report JavaScript errors unless this feature is turned on specifically. An easy way for a developer to see these errors is to open the console, which is a kind of debug engine that most JavaScript engines provide with the browser. The simplest version to use currently is the one found in Google Chrome. When you're in the browser, right-click anywhere on the screen and select Inspect Element. A new window will appear docked to the bottom of the browser or possibly as a completely separate window. Across the top, you'll find the Console button—click it. You should see all the exceptions thrown during the current session.

1.3.4 Building the server side

Now that your client side is complete and you have at least a basic understanding of how various pieces of JavaScript are initialized and executed, it's time to build your server implementation. The HTML application you're building in this chapter requires both a client and a server implementation.

Many web applications use the server only as file storage. In these applications, once the resources such as stylesheets, HTML files, and scripts are loaded, the server is never contacted again. Games are a normal example of this kind of application. Your application, however, needs to talk to a server to send information and receive updates. To do that, you need something more than a normal HTML page or MVC controller that returns a view. You need something that takes only data and returns only data. You need JSON.

Your steps here will include

1. Building the model object to contain data on the server.
2. Building the MVC controller method to handle the Ajax request.

JavaScript Object Notation (JSON)

JSON is used for transferring text-based data from one point to another over HTTP and for serialization of JavaScript objects. It can also be used for many other purposes, but its roots are in the web. It's fast, human readable, and broadly supported.

Syntax in JSON is extremely simple. Specific characters are used to wrap text into serialized fields with very little effort and overhead. Data types of field values are implied, not specified, and objects need not conform to a specific schema. Arrays in JSON can contain any kind of object.

Here are the basic rules:

- Curly braces, `{}`, wrap each object instance, and square brackets, `[]`, wrap each array instance.

- Each property in an object has a name and value separated by a colon.
- Each property in an object and each object in an array must be separated from the next by a comma.
- Property names that correspond to keywords must be wrapped in quotes.
- Property values that are strings are always wrapped in quotes.
- Object properties can be other objects or arrays.

Here is a simple JSON object:

```
{"fname":"George", "lname": "Washington"}
```

This code will result in the direct creation of a JavaScript object with two properties, each with a value.

Some projects will require sending large amounts of data to the client, and JSON is perfectly capable of doing this as well. The following JSON code contains an array of two objects, each containing a timeline that can be immediately parsed and used in JavaScript:

```
[{
   "Timeline":"1800s", "StartYear":1800, "EndYear":1899,
   "Events": [
      {"Date":1803, "Event":"Louisiana Purchase"},
      {"Date":1808, "Event":"Napoleon Occupies Spain"},
   ]
},
{
   "Timeline":"1900s", "StartYear":1900, "EndYear":1999,
   "Events": [
      {"Date":1917, "Event":"US Declares War"},
      {"Date":1991, "Event":"Desert Storm"}
   ]
}]
```

JavaScript is used to parse an object from a string using the JSON parser that's either built into most modern browsers or available with the free json2.js library found at http://www.JSON.org/js.html. If your browser doesn't support JSON, just include this script and all the JSON parsing logic will be automatically added. Using this method, you can create an object from a JSON string by calling

```
JSON.parse('string variable');
```

An object can also be turned into a string using the `stringify` method of the same library:

```
var x = JSON.stringify(myObject);
```

Before you can build an endpoint on your server to take a JSON object and turn it into a .NET object, you first need to define the properties for that object. In your solution, add a new class to the Models folder called UserData.cs. This object should contain three properties, so add the code shown here:

```
namespace HelloWorld.Models
{
   public class UserData
   {
      public string UserName { get; set; }
      public string RequestedDate { get; set; }
      public string ServerDate { get; set; }
   }
}
```

This object will hold the user's name, the string version of a request date, and the string version of the current date on the server.

> **NOTE** You could have made these `DateTime` properties, but that would distract from our goal of showing you how to receive, manipulate, and send data from the server. If you want a more detailed investigation of date handling in JavaScript, take a look at appendix A on JavaScript toward the end of this book.

The next step is to build a controller call that can respond to data posted from the client using Ajax. You could create an entirely new controller, but that's unnecessary because you already have the Home controller available. Open that controller by navigating in Solution Explorer to Controllers > HomeController. Note that the MVC convention is to refer to controllers by their name with the suffix of "Controller," so the `AccountController` will be referred to by the URL `/Account` in your browser.

> **NOTE** There is a bit more to controller naming than just the standard naming convention, but that conversation will involve setting up ASP.NET MVC routes. You can find more information about routes as they relate to MVC applications in chapter 5. You can also check out the great books by authors K. Scott Allen, Steven Sanderson, Phil Haack, and Adam Freeman. All contain a wealth of knowledge in this area.

Back in the `HomeController`, you need to add a new method. It will receive a JSON object from the client and automatically transform it into your `UserData` object. It will make some changes to that object and return it, transforming it back into JSON. This action is shown in the following listing.

Listing 1.7 `GetMessage` receives and sends data using client Ajax calls

```
public JsonResult GetMessage(UserData myData)
{
   myData.UserName += " (verified)";
   var dt = DateTime.Now.AddYears(-1);
   DateTime.TryParse(myData.RequestedDate, out dt);
   myData.RequestedDate = dt.ToLongDateString();
   myData.ServerDate = DateTime.Now.ToShortDateString();
   return Json(myData, JsonRequestBehavior.AllowGet);
}
```

Model binding in MVC will automatically convert inbound JSON to UserData object.

Attempt to parse data to date and leave it as arbitrary value if conversion fails.

Method returns JsonResult object so you can generate JSON using MVC serializer.

Note that in the controller call in listing 1.7 you're taking a `UserData` object as a parameter, but in the client, the object you're passing looks like this:

```
var myData = {
      UserName: nm,
      RequestedDate: dt
};
```

This works because ASP.NET MVC will attempt to transform the input parameters into the appropriate object type using its model-binding mechanism. If you wanted to, you could also have written the controller function signature as follows:

```
public JsonResult GetMessage(string UserName, string RequestedDate) {
```

This would have resulted in the same data being received on the server. The advantages of model binding are that your method signatures are smaller and easier to understand, and your objects are created with constructor methods that perform logic that will be automatically executed when the function is called. These features make MVC controllers and model binding the ideal way to implement Ajax endpoints for an HTML application.

You may be surprised to learn that your first HTML application is complete! Run the solution now and try it out in various browsers. You should see a nearly identical implementation in each. Input a name and a date, and watch the results return from the server after being "verified," as shown in figure 1.17.

As you test your freshly minted application, it's worth taking a look at how various desktop browsers implement the `<input type='date' />` tag (figure 1.18). Opera, for instance, gives you a built-in date picker, whereas Safari has a small up/down implementation that changes the date value one day at a time.

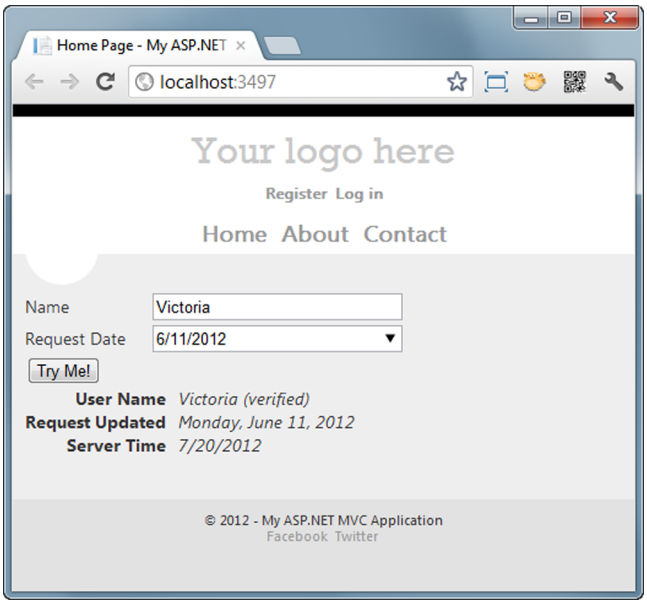


Figure 1.17 After completing the controller call, you should see data from the server displayed and updated automatically.

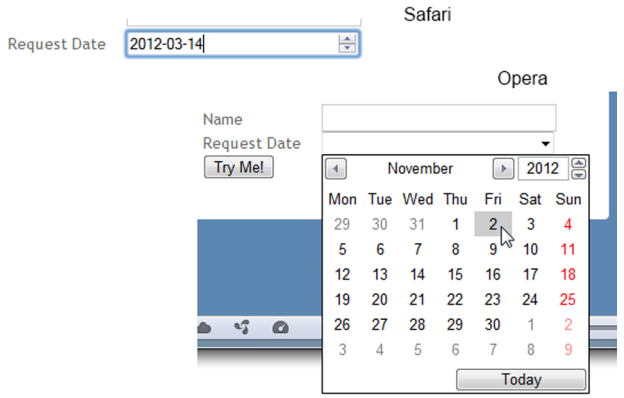


Figure 1.18 The date picker implementations in various browsers highlight the wide range of interpretations in the browser market.

This date field difference highlights both the necessity of testing your HTML applications across a range of expected browsers and the need for feature testing. Feature testing is usually done with a JavaScript library like Modernizr (http://www.modernizr.com/) that will return a Boolean value for a specific feature. If you decide that you absolutely require a feature, and it isn't present in the current browser, you can alert users that they must use a different browser.

Modernizr is the natural choice for this kind of feature detection because it's widely used, well-maintained, and is included by default in all recent versions of ASP.NET MVC. The code is as simple as this:

```
if (Modernizr.touch){
   // .. bind touch events here
}
else {
   alert('touch is not supported');
}
```

1.4 Summary

With the level of knowledge you now have about the moving parts and interactions of an HTML5 application built on an MVC foundation, you should be ready to dive into your own applications and start tinkering to see what you can make happen on your own! The architecture is straightforward and the possibilities are endless.

But this chapter is far from the end of the story. In order to build richer, more functional applications that can interact with all the new HTML5 APIs, style properties, and semantic markup we talked about in this chapter, you need to dig deeper, starting with the new semantic elements and CSS features available in HTML5 and CSS3. That's what we'll look at next.

If you're already familiar with the new elements and CSS features, you may want to skip chapter 2 and move straight to the later chapters, where we dive into each of the HTML5 JavaScript APIs to show you how you can use your current .NET skills to build the next generation of applications in the browser.

HTML5 for .NET Developers

Jackson • Gilman

A shift is underway for Microsoft developers—to build web applications you'll need to integrate HTML5 features like Canvas-based graphics and the new JavaScript-driven APIs with familiar technologies like ASP.NET MVC and WCF. This book is designed for you.

HTML5 for .NET Developers teaches you how to blend HTML5 with your current .NET tools and practices. You'll start with a quick overview of the new HTML5 features and the semantic markup model. Then, you'll systematically work through the JavaScript APIs as you learn to build single page web apps that look and work like desktop apps. Along the way, you'll get tips and learn techniques that will prepare you to build "metro-style" applications for Windows 8 and WP 8.

What's Inside

- HTML5 from a .NET perspective
- Local storage, threading, and WebSockets
- Using JSON-enabled web services
- WCF services for HTML5
- How to build single page web apps

This book assumes you're familiar with HTML, and concentrates on the intersection between new HTML5 features and Microsoft-specific technologies.

Jim Jackson is a software consultant and project lead specializing in HTML5-driven media. **Ian Gilman** is a professional developer passionate about open technologies and lively user interfaces.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/HTML5for.NETDevelopers

"Speaks directly to the interests and concerns of the .NET developer."
—From the Foreword by Scott Hanselman, Microsoft

"Looks under the hood of HTML5 to teach more than just pretty pages."
—Joseph M. Morgan, Amerigroup

"A comprehensive jumpstart for the .NET developer looking to make a leap into HTML5."
—Peter O'Hanlon
Lifestyle Computing Ltd

"A great HTML5 and API learning resource!"
—Stan Bice
Applied Information Sciences

ISBN 13: 978-1-617290-43-5
ISBN 10: 1-617290-43-2
54499
9 781617 290435

MANNING $44.99 / Can $47.99 [INCLUDING eBOOK]