Part 1

Moving on to HTML5

HTML5 is the newest incarnation of the HTML family of languages. HTML, which stands for HyperText Markup Language, is one of the main reasons the Web is as powerful and useful as it is. HTML is a reasonably simple system of plain-text codes that provide the structure of all Web pages on the Internet.

In this part, you take a quick look at how HTML5 fits in the history of the Web, and put together a few tools you'll need to get started.

Be sure to check out my Web site for working examples of every code fragment in the book: `www.aharrisbooks.net/h5qr`.

In this part . . .

- **Looking at the History of HTML**
- **Understanding What HTML5 Is**
- **Running Tests for Browser Features**
- **Deciding on a Suitable Browser**
- **Utilizing Chrome Frame to Add Support to IE**

A Quick History of HTML

HTML is a key part of the Internet. It has a short but extremely vibrant history. In order to understand what HTML5 is about, it's useful to look at where it came from. The Internet (and the Web in particular) has been changing at a dizzying pace. HTML has been trying to keep up.

When HTML was first devised, it comprised a handful of tags, and HTML did little more than determine how a page was laid out. As the Web matured, many features were added. Today's Internet is still about documents, but it's also about applications. Today's Web sites are dynamic interactive applications.

The kinds of devices used on the Internet are changing, too. In the early days, only desktop computers used the Web. Now cellphones and mobile devices are among the most important players on the Web. They require a different way of thinking than the standard desk-based behemoths of a few years ago.

It's time for a fresh new set of standards that will help support the way people are using the Internet today. HTML5 is that set of standards.

A bit of ancient history

In the distant mists of time (1989) Tim Berners-Lee created a new system of connecting electronic documents. He devised a simple language that allowed document authors to link various documents together with limited formatting options. This language was called HTML.

At that point, the Internet existed, but it was mainly accessed by basic command-line programs, and was not easy to use. HTML (and some other underlying technologies) was designed from the beginning to be easy to work with, and to create documents that were easy for users to manage. The design of HTML was deliberately kept simple, so as many people as possible could participate in the process of building documents in this new format.

Of course, the Web took off in a very major way, and soon Web pages became ubiquitous. It became clear that the simple features in basic HTML were not enough to satisfy the interests of the many people who were now building Web pages.

And the first browser war begins . . .

As various organizations started building *Web browsers* (the tools that read HTML and display it to the user), they began competing by adding new HTML features. By 1993, the Mosaic browser included the ability to add images (which were not part of the original specification). Many browsers were being created by small teams all around the world, and each had its own set of new features.

By 1994, one platform emerged as the dominant browser. Netscape Navigator was a profoundly successful browser. At the same time, there were working groups

forming to address the lack of standards in the Web browser world. The most important of these groups was called the World Wide Web Consortium (W3C) headed by Tim Berners-Lee (the same guy who started all this mess). However, Netscape had such a dominant position that Netscape representatives often skipped the standards meetings and created whatever features they wanted.

Microsoft did not come into the browser world until 1995. Internet Explorer (IE) was designed to compete directly with Netscape's browser. For a time (sometimes called the first browser wars), Netscape and Microsoft were in an arms race, each trying to produce exclusive features that would steer developers toward their own vision of the Web.

While there was a standards body in place, the reality was both Netscape and Microsoft added whatever features they wanted and basically ignored the W3C. There was some small progress made on Web standards. HTML 2 was adopted as a standard in 1994/1995 (although none of the manufacturers stuck with it completely). HTML 3.2 was released in 1997, followed by HTML 4 in Spring of 1998.

By about the same time HTML 4 was gaining traction, it became clear that Microsoft was dominating the browser space. By 2002, Internet Explorer was used by approximately 95 percent of Internet users. With that kind of clout, the future of HTML was almost entirely in Microsoft's hands, and efforts of standards bodies were largely irrelevant. By any measure, Microsoft won the first browser war. Internet Explorer 6 (which used mainly HTML 4) was the only browser that really mattered, and there was very little innovation for several years.

A new challenger arises from the ashes

However, there were some new browsers that challenged Microsoft's dominance. The Firefox browser (first released in 2004) in particular was especially important, as it introduced a number of innovative features and followed most of the standards of the W3C working group. Firefox (and to a lesser extent other browsers like Apple's Safari, Opera, and eventually Google Chrome) shook up the Web. These other browsers tended to be more committed to following standards than IE was, and they prompted new versions of IE following a long era of stagnation. Even Microsoft began to at least pay lip service to the notion of standards, promising more standards compliance in each of the new versions of IE introduced. Some consider this the opening of the second browser war, with various developers competing for share of the browser market.

However, there is a difference this time around. The Web is no longer a novelty, but now a key part of business and society. A Web-based document is now held to the same visual standards as printed documents, and HTML 4 is simply not capable of easily meeting this standard. In fact, the entire notion of the Web as a series of documents is being challenged. Web *pages* are being replaced by Web *applications*. Much of what people now do on the Internet isn't about reading documents any more. Today, developers are using the Web itself as a programming interface.

HTML 4 was getting old

Changes in the Web required a change in the thinking about document standards. HTML 4 was clearly not up to the task of supporting modern Web development. The various proprietary tags added through the years added some visual flexibility, but not nearly enough. There was not a satisfying solution for page layout or font management. There was a set of features for entering form data, but these tools were limited and ugly. Most browsers featured a form of the JavaScript programming language, but the implementations varied wildly, and making a real application using Web technologies was a chancy proposition.

The W3C introduced XHTML in 2002 to address some of these concerns. XHTML was proposed as a version of HTML adhering to the stricter standards of the XML markup language. XHTML is much less forgiving than HTML, so if a page meets the stringent requirements of the standard, it is (presumably) well-behaved and predictable. Unfortunately, the idealism of the XHTML movement was never realized. Creating valid XHTML documents proved difficult enough that very few developers tried. Browsers rendered inaccurate XHTML code decently (if not perfectly). In fact, most browsers didn't really render XHTML at all, but quietly converted it to a form of HTML. There was little incentive for developers to adhere to XHTML standards (unless they were taking my class).

In order to get the functionality that was missing from HTML, many developers turned to plug-in technology like Java Applets and embedded Flash. Java never caught on as a client-side environment (although it remains extremely important in other applications) but Flash was very popular for a time. Unfortunately, Flash introduces problems of its own. The content of a Flash applet can only be modified by a Flash editor, and it cannot be read (at least easily) by search engines. Many of the new features of HTML5 (particularly the font support and the canvas tag) can be seen as a direct response to Flash.

The W3C moved to create a new form of XHTML called XHTML 2.0, but in the mean time, a second group called WHATWG (Web Hypertext Application Technology Working Group) began working on their own competing standard, which came to be known as HTML5. The main reason for these competing standards was a sense that XHTML was too rigid, and was still focused on HTML as a document language. Part of the motivation for HTML5 was to create a framework for building Web applications that would really be used by developers. Eventually, W3C dropped support for XHTML 2 and is now supporting the WHATG proposal, so HTML5 appears to be the next standard.

Getting to Know the Real HTML5

The WHATWG group seems to have learned a few lessons from history. The design of HTML5 indicates these priorities:

- **The core language should be simple.** HTML5 is quite a bit cleaner than XHTML. The document type in particular is a breath of fresh air compared to the nonsense you have to write to start an XHTML page. Every tag is about describing some feature of the page. Most of the tags are plain English with few abbreviations.
- **Markup is based on semantics.** One of the original ideas in HTML was markup based on *meaning* rather than *details*. For example, a headline is simply marked as `<h1>` rather than specifying a particular font size or typeface. HTML5 returns to this tradition, adding a number of new tags to describe common parts of a page.
- **CSS is used for style details.** Like XHTML, HTML5 relies heavily on another language, called CSS (Cascading Style Sheets), to handle the details of how a particular element looks. In essence, HTML describes *what* a page element is, and CSS describes *how* that element looks. HTML5 does not contain tags like `<font>` or `<center>` because these characteristics are handled in a more flexible way by CSS.
- **Pages are often applications.** *Forms* (the elements that allow users to enter data in a Web site) have been a part of HTML since the beginning, but they have not seen much improvement over the years. HTML5 adds a number of very exciting new form elements that make HTML a much better tool for interacting with users.
- **JavaScript is central.** Most Web browsers have had a form of the JavaScript (JS) programming language built in for years. However, it was difficult to take JavaScript very seriously because it had a number of limitations. Some limitations were because of legitimate security concerns, and others were simply poor or incompatible implementation. With the advent of new powerful JavaScript engines and a new paradigm called AJAX (Asynchronous JavaScript and XML), JavaScript has re-emerged as a powerful and important programming environment. Many of the most interesting features of HTML5 (like the canvas tag) are mainly improvements in the JavaScript language. (The canvas tag is an HTML tag, but it doesn't do anything interesting without JavaScript.)

HTML5 Is More than HTML!

It's a little unfortunate that this technology has been called HTML5, because the HTML language is actually only one part of a much bigger picture. In truth, the thing we call HTML5 is the integration of several different technologies (HTML, CSS, and JavaScript, and server-based technologies), which each have their own role as follows:

HTML

Of course, there have been changes to the HTML language itself. A few tags have been added to the HTML 4 standard, and a number have been taken away. However, HTML5 remains backwards-compatible with HTML 4, so there's no absolute requirement to write your code in the HTML5 standard. Adapting from HTML 4 to HTML5 is probably the easiest part of moving to the complete HTML mindset.

Here are the main HTML features:

- **Semantic markup:** HTML5 now includes new tags that describe parts of a document. Now there are dedicated tags for navigation elements, articles, sections, headers, and footers.
- **New form elements:** HTML5 forms have some major updates. There are several new versions of the `<input>` element, allowing users to pick colors, numbers, e-mail addresses, and dates with easy-to-use elements.
- **Media elements:** At long last, HTML5 has native support for audio and video with tags similar to the `<img>` tag.
- **`canvas` tag:** The `canvas` tag allows the programmer to build graphics interactively. This capability will allow for very intriguing capabilities like custom gaming and interface elements.

CSS

Probably the biggest adjustment for those used to HTML 4 isn't really the HTML itself, but the changing relationship between HTML and CSS. In HTML5 (like in XHTML), the markup language only describes what various elements mean. CSS is used to describe how things look. If you're really going to switch to HTML5, you can no longer use tags like `<font>` and `<center>`, which are about describing details. CSS could be considered an optional add-on to HTML 4, but it's central to the HTML5 way of thinking. If you haven't yet learned CSS, it's definitely time. CSS is a different way of thinking, but it's incredibly powerful and flexible. Along with the HTML5 standard comes a new standard for CSS, called CSS3. It's nearly impossible to talk about HTML5 without also including CSS3 because they're so closely related. Here are the main new features:

- **Embedded font support:** With this long-awaited tool, you can include a font with a Web page, and it will render even if the user doesn't have the font installed on her operating system.
- **New selectors:** Selectors are used to describe a chunk of code to be modified. CSS3 now supports new selectors that let you choose every other element, as well as specific sub-elements (different types of input tags, for example).
- **Columns:** HTML has never had decent support for columns, and all kinds of hacks have been used to overcome this shortcoming. Finally, CSS includes the ability to break an element into any number of columns easily.

- **Visual enhancements:** CSS has a number of interesting new capabilities: transparency, shadows, rounded corners, animations, gradients, and transformations. These provide a profound new level of control over the appearance of a page.

JavaScript

If HTML describes what parts of the document are, and CSS describe how these parts look, JavaScript is used to define how elements act. JavaScript is a full-blown programming language, and it deserves its own book (which, of course it has; look to my book *JavaScript and AJAX For Dummies* [Wiley] for one example). It is not possible to describe JavaScript completely in this reference guide, but JavaScript is a very critical part of the HTML5 point of view. A few of HTML5's most interesting features (the `canvas` tag, geolocation, and local data storage, for example) are accessible only through JavaScript. I describe these features in this book. ***See*** Bonus Part 1 for an overview of JavaScript if you need a review or an introduction.

- **Vector graphics support:** Vector-based graphics provide an interesting alternative to traditional graphics because they can be created on the fly through code. HTML5 actually has two ways to do this: through SVG (Scalable Vector Graphics) and the `canvas` tag.
- **New selectors:** Most JavaScript programming begins by grabbing an element by ID. HTML5 now allows you to select elements by tag name, or by the same mechanisms you use to select elements in CSS.
- **Local storage mechanisms:** Previous versions of HTML allowed very limited storage of information on the client. HTML5 now allows the developer to store data on the client. There is even a built-in database manager that accepts SQL commands.
- **Geolocation:** This interesting feature uses a variety of mechanisms to determine where the user is located.

Server technologies

Modern Web development is about communication. All of the technologies that make up HTML5 reside in the Web browser, which is an important part of the Web. However, an equally important part of Web development is a raft of technologies that live on the Web server. Many of the most interesting things happening today use technologies like PHP or ASP to run programs that create Web pages. Many interesting applications also use database programs like Oracle or MySQL to manage large amounts of data. The advent of AJAX has made integration between those technologies and the browser much easier. Interesting as these tools are, I do not focus on them in this reference book. If you're interested in them, please see my book *HTML, XHTML, CSS All-in-One For Dummies* (Wiley) for a thorough treatment of these and other topics.

Looking At Browser Features

As you can see in the history of HTML, calling something a standard doesn't make it so. Officially, HTML5 hasn't been accepted yet, and there isn't a single popular browser that implements all of its features. If that's the case, you might wonder if it's worth it to study this technology yet. I think so, for these reasons:

- **Most of the ideas are accepted.** While HTML5 itself has not yet been ratified as a formal standard, most of the critical ideas are available today. Today's Web browsers will work fine with HTML5 even if they don't know how to do all the cool things with it.
- **There is little doubt that HTML5 is the new standard.** W3C has essentially conceded that XHTML 2.0 is not a viable solution, leaving HTML5 as the clear winner in the standards war. If there is to be any standard at all, HTML5 (and the related features in CSS and JS) is it.
- **Standards-compliance is now a desirable feature.** In the first browser war, manufacturers were competing to add new features without any regard to standards. Today, browsers are judged by their adherence to accepted Web standards. Even Microsoft has gotten into the mix, claiming that IE 9 supports a majority of the HTML5 features.
- **HTML5 promotes good coding habits.** The separation of content from layout is a critical part of modern Web development. If you're coming from XHTML, you're already comfortable with this situation. If you're more familiar with HTML 4, it's a new idea, but one that has been inevitable.

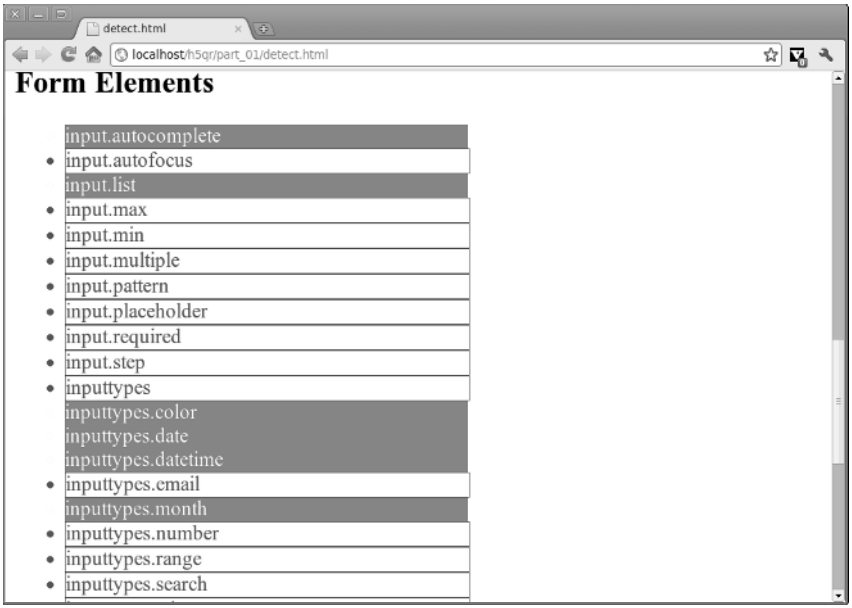
Officially, HTML5 is not expected to be completely accepted as a standard until 2022. This seems like an eternity in Web time. However, parts of the standard (such as the `canvas` tag) are universally available right now and are worth exploring immediately. Others (like most of the form elements and the semantic markup tags) provide suitable backups automatically if the browser doesn't support the advanced features. Others (like drag-and-drop) are simply not ready for use yet. A few (like the local data support mechanism) are hotly debated, and it is not clear which form of the technology will become part of the standard. As I discuss each of these topics throughout the book, I try to give you a sense of whether it is ready to be used yet, and which browsers support particular features.

Assessing your browser's capabilities

HTML5 has a lot of different technologies going on, and different browsers have adopted different parts of the standards. It can be very confusing to determine which features can be used. There are a couple of good solutions to this problem. A number of sites have charts that indicate which features are supported in which browser. I like the ones at `http://caniuse.com` and `http://en.wikipedia.org/wiki/Comparison_of_layout_engines_%28HTML5%29`. These tools can help you see what is currently supported by the major browsers. It's especially handy for checking browsers you don't have on your own machine.

However, browser support for HTML5 features literally changes every day. New versions of major browsers are appearing all the time, and it's very hard to keep track of what's currently happening. For that reason, I've provided you with a program you can use to check your current browser to see which HTML5 features it supports. Figure 1-1 shows the detect.html program in action.

Figure 1-1



The detect.html page can be found at my Web site, `www.aharrisbooks.net/h5qr/detect.html`. Use it with any browser to get real-time analysis of which HTML5 features are available in your browser.

The program uses a script called Modernizr, which automates checking for various browser features. You can get Modernizr for free from `www.modernizr.com`.

Checking for features in your code

You can also use the Modernizr script in your own code. Essentially, Modernizr creates a Boolean (true/false) value for each of the HTML features. You can check a variable to see if the current browser supports a particular feature. If it does, you can implement the feature. If not, you will generally implement some sort of fallback. Here's how it's done:

1. Download the Modernizr script. The Modernizr script can be downloaded free from `www.modernizr.com`. Install the script in the same directory as your Web page. (If you move your page to a server, you'll also need to make a copy of the script available.)

2. Include a reference to the script. Use the `<script>` tag to make a reference to the script in your header (before any other JavaScript code):

```
<script type = "text/javascript"
        src = "modernizr-1.6.min.js"></script>
```

3. Add a special class to the HTML tag. The Modernizr script needs to have a special tag available so it knows what to do. Add the "no-js" class to the HTML tag:

```
<html lang = "en"
      class = "no-js">
```

4. Write a new JavaScript function. Add a new JavaScript function to do the actual testing. Specific examples are shown in the code listing later in this section.

5. Use the appropriate Boolean property to check for a particular feature. Each of the HTML5 features supported by Modernizr has a corresponding variable. (You can look up the variables on the Modernizr site, or look at my detect.html script, which uses them all.)

6. Use the feature or an alternative. Normally, you'll use Modernizr to check for a feature. If that feature exists, you'll use it. If not, you'll implement some other alternative.

As an example, the following page uses the Modernizr script to test whether the current browser supports the HTML5 `video` tag. If so, it also checks for support of the two main video codecs.

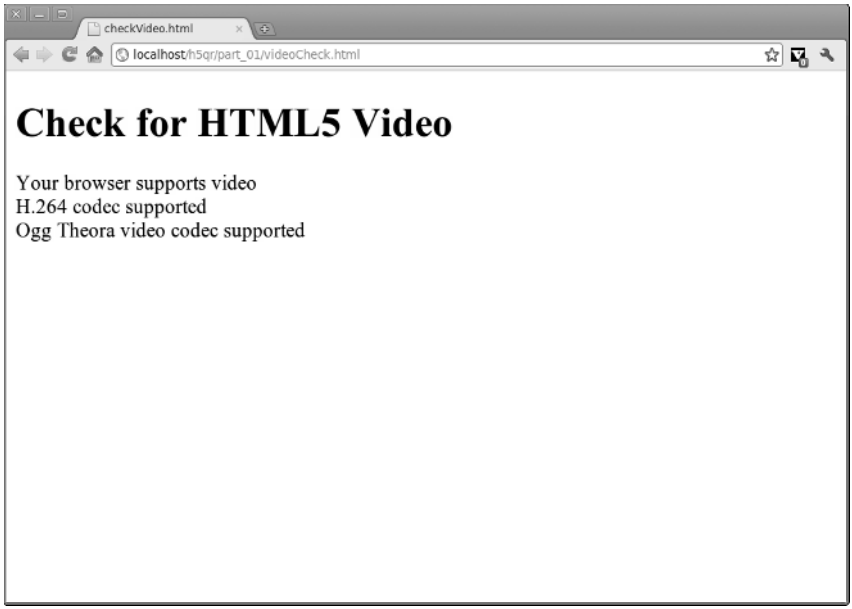
```
<!DOCTYPE HTML>
<html lang = "en"
      class = "no-js">
<head>
  <title>checkVideo.html</title>
  <meta charset = "UTF-8" />
    <script type = "text/javascript"
            src = "modernizr-1.6.min.js"></script>
    <script type = "text/javascript">
      function init(){
        var output = document.getElementById("output");
        if (Modernizr.video){
          output.innerHTML =
          "Your browser supports video <br /> ";
          if (Modernizr.video.h264){
            output.innerHTML += "H.264 codec supported <br
/>";
```

```
            } // end if
            if (Modernizr.video.ogg){
              output.innerHTML +=
              "Ogg Theora video codec supported <br />";
            } // end if
          } else {
            output.innerHTML = "Your browser does not support
the HTML5 video tag";       } // end if
        } // end init
  </script>
</head>

<body onload = "init()">
  <h1>Check for HTML5 Video</h1>
  <div id = "output">
    checking video...
  </div>
</body>
</html>
```

Figure 1-2 shows the video-checking script in action.

Figure 1-2



This example simply checks for the support for the video elements. A more sophisticated example would actually embed the appropriate tags or code in the page to display a video according to the browser's capabilities.

For more information on the video tag, please check Part 3.

Picking a Suitable Browser

If you're going to be writing HTML5 code, you'll probably want to view your pages in a browser that interprets HTML5 correctly. That's not as easy as it sounds. HTML5 isn't really one specification, but a number of different standards. The various browsers have differing versions of support. It's best to have a wide variety of browsers to see which one works best for you. There are several browsers currently available, which all have varying levels of HTML5 support.

While there are a large number of browsers available, most are based on a smaller set of tools called *rendering engines.* It's the rendering engine that really supports features or not. Here is a list of the primary engines, the browsers that use them, and how well they support HTML5:

- **Gecko (Firefox):** The Gecko engine is the main engine of Firefox, Mozilla, and a number of other related browsers. It has support for many, but not all features. Gecko 2.0 is expected to include most features of HTML5, but that version of the engine is not yet released (and will probably be the foundation of Firefox 4). Although Firefox is a well-known and respected browser in the Web development community, it does not (yet) have extremely good support for HTML5.
- **Trident (Internet Explorer):** The various forms of Internet Explorer all use the Trident engine. So far, this engine has the weakest support of HTML5 features among all the major browsers. IE9 promises to have much more complete support for HTML5, but even this version is projected to be missing some key features, including advanced form element support and geolocation.
- **WebKit:** The WebKit engine was originally created by Apple based on code from the open source KHTML project. Apple then released the code as open source, where it became the foundation of a number of browsers. The Safari browser on Macs, iPhones, and iPads all uses the WebKit engine. WebKit is also the foundation of the Google Chrome browser, and the browser on the Android mobile platform. WebKit has become the standard rendering engine for mobile platforms. If you want to see how your pages will look on mobile platforms, you should check with a WebKit-based browser like Chrome or Safari. WebKit has the widest support for HTML5 elements, although it still doesn't support everything. Most of the

code in this book was tested in Google Chrome 6, which supports the current WebKit rendering engine.

- **Presto:** The Presto engine is the engine underlying the Opera family of browsers. Opera has long been considered a technically superior browser, but it has never gotten the market share it should. A number of gaming and portable browsers are based on Presto, including the Wii Internet Channel, the Nintendo DS Browser, and Opera Mobile, available on numerous cellphones and portable devices.

Browser specifications are likely to change. It's likely that new features have been added by the time you read this book. You should always test your page in as many browsers as you can, so you won't be surprised. You might also check `http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML5)`. This Wikipedia site tends to have the latest information on what features of HTML are supported by which browser.

Using Chrome Frame to Add Support to IE

It might be depressing to note that the browser with the largest market share has the least support for HTML5 standards. However, there is an answer. Google Chrome Frame is a special tool that embeds the Chrome rendering engine inside IE. To use it, put the following code in your page:

```
<!DOCTYPE html>
<html lang="en">
<head>
     <title>ChromeFrame.html</title>
     <meta charset="UTF-8">
  <script type="text/javascript"
          src="http://ajax.googleapis.com/ajax/libs/chrome-
frame/1/CFInstall.min.js"></script>
</head>

<body onload = "CFInstall.check()">
</body>

</html>
```

The rest of your code can be written assuming the user has Chrome (which has excellent support for HTML5). This is the best way to use HTML5 in IE until Microsoft decides to add meaningful support to HTML5.