CHAPTER 8

Enhancing Flash Deployment

While all the fun occurs while authoring Flash content and preparing video, integrating Flash with HTML can't be avoided. This chapter covers issues you should know about when deploying to the Web.

Deploying Flash with HTML

Let's face it, Flash Video is primarily embedded in web pages. Given that relationship, there are several issues to wrangle when publishing video on a web site: web browser compatibility, ensuring the correct Flash Player is installed, making the content search engine friendly, displaying alternative content for those who don't have Flash or JavaScript enabled, and overcoming the "click to activate experience" caused by recent changes to Microsoft Internet Explorer. As a developer you can:

- Use standards-compliant markup.
- Use proprietary markup to solve browser compatibility.
- Use JavaScript to dynamically embed the video.
- Use combinations of the aforementioned methods.

None of these directions are perfect and there are several implementation approaches for each. In this chapter we'll cover the issues that affect publishing Flash on web pages and present tutorials that cover some of the popular methods.

How Flash Video Is Embedded

Flash Video (FLV) is "housed" inside a container Flash movie (SWF). The housing SWF streams the FLV and provides methods to control playback (either on its own or through a skin SWF). The housing SWF is embedded inside a web page (HTML). When a viewer visits a page with Flash Video, the browser loads the Flash Player plug-in and viewing begins.

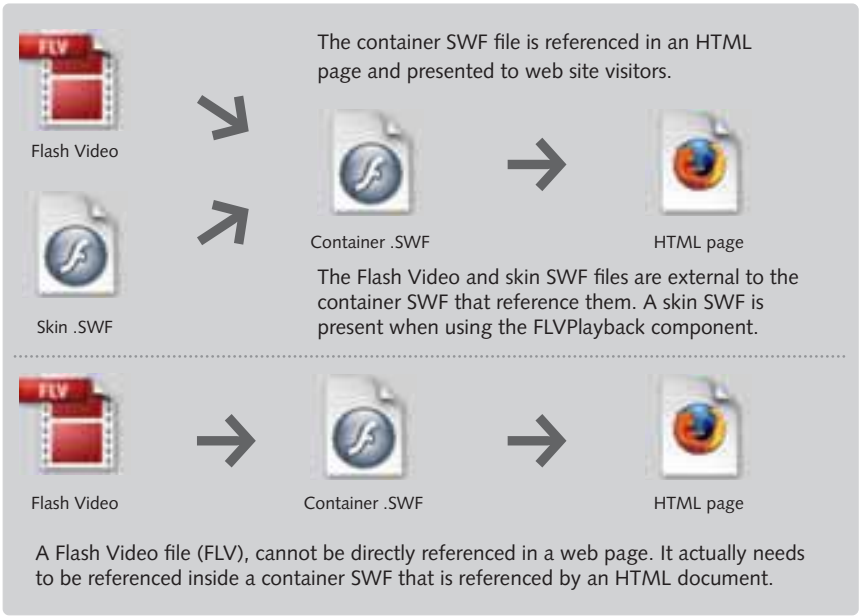

Figure 8.1: *Relationship between an FLV, SWF, and HTML document.*

Browser Compatibility and Web Standards

The Internet is not viewed by only one browser alone. Yes, Microsoft Internet Explorer commands a large portion of the browser market share, but there are millions of viewers who also use the Firefox browser, Opera, or Safari on Mac OS X. The problem is not in how each browser supports Flash, but really how they support non-HTML (or plug-in) content in general. Flash like QuickTime, Real Media, or Windows media is placed on a web page using the <embed> tag, the <object> tag, or both tags.

What Are Web Standards?

At the peak of the first Internet bubble, most web pages were created by mixing structure and presentation, did not use semantic markup, and used a variety of proprietary, hack-derived, and inaccessible technologies. The result was sites that worked for some and appeared broken to others. Coincidentally, many of these hacked-together pages are not optimized for search engines. The term and movement "Web Standards" grew from the desire for long-term universal access and interoperability on the World Wide Web.

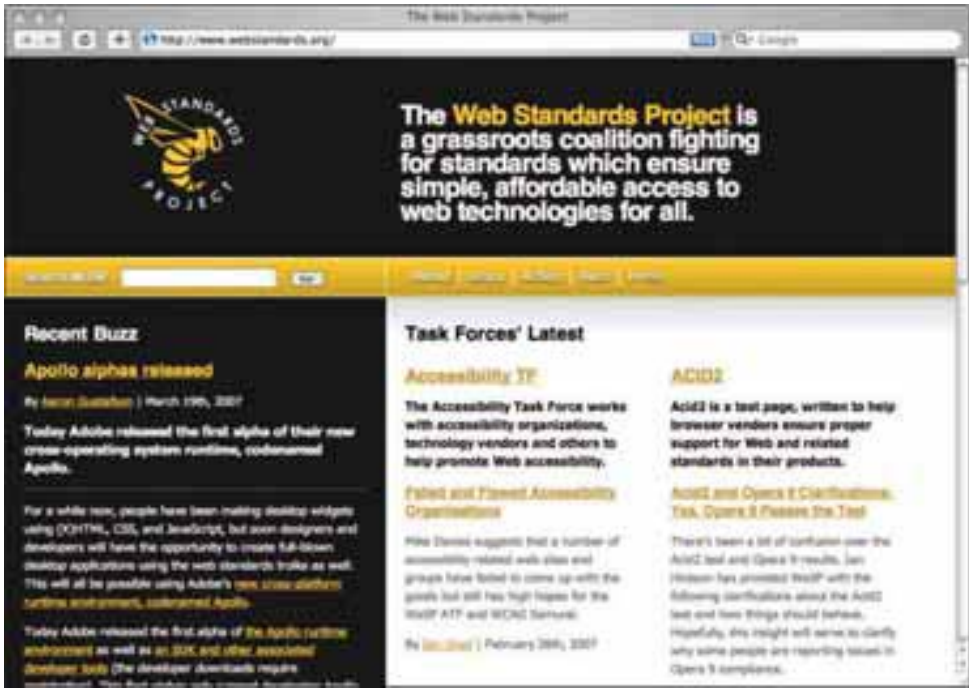

Figure 8.2: *Visit http://www.webstandards.org to learn more best practices.*

Web standards promotes the use of semantic markup, open standards (XHTML, CSS, JavaScript, microformats, and XML), and accessibility among developers and designers. The movement also works closely with browser manufacturers and web tooling companies such as Adobe to increase standards support.

Semantic Markup

Semantic markup is the practice of marking web pages using appropriate HTML tags in marking up content. For example, paragraphs are enclosed inside a paragraph tag, **<p>**, a first-level heading is placed inside a heading 1 tag, **<h1>**, and a numbered list is marked up using the ordered list and list item tags, **<ol>** and **<li>** respectively. When markup reflects the organization and meaning of content, it future-proofs the content, makes it easier to find via a search engine, and greatly facilitates changes and updates to both the content and the design.

The practice of applying semantic markup is a best-practice alternative to building web pages completely with tables. While tables were an effective layout mechanism in the early days of the Web, they should be avoided today. Tables should only be used for displaying tabular data. Pages should use ** ** tags along with other block-level HTML tags and cascading style sheets (CSS) to create layouts.

Digital Web also has an excellent primer on writing HTML using semantic markup by Joshua Porter and Richard MacManus. It's at: http://www.digital-web.com/articles/writing_semantic_markup/.

Open Standards

Open standards are web technologies such as HTML, CSS, and JavaScript. They are standards because they are managed by international nonprofit organizations with representation from educational institutions, corporations, and the open source community. These technologies have specification and review processes that are open to anyone. When a technology reaches a final draft, browser manufacturers and tool developers are encouraged to support the specifications. By using open standards and avoiding proprietary markup, universal access is achievable.

A List Apart has many articles on designing web sites using standards. It can be found at: http://www.alistapart.com.

While not an open standard, Flash, like all other rich-media technologies, has its place when it is implemented responsibly. Responsible implementation means using unobtrusive techniques for inserting content and providing alternative content and assistance for those who cannot view the material being presented.

Web Page Validation

Having pages validate is sort of like passing a grammar quiz. While there may be a few correct answers to a question, there's no doubt when an answer is wrong. That said, it's not surprising that many developers do not or conveniently forget to validate their pages using the W3C's (World Wide Web Consortium) page validator. To ensure your page passes validation, here are a few things your markup needs to do:

- Properly declare a document type and character encoding. The document type declares what version of HTML you are using. The character encoding is important when using mathematical or foreign language characters.
- All tags should be well-formed. This means items have opening and closing tags such as a ** **A line of text** **. For single tags such as the line break tag, a closing slash should be used: **
** instead of simply **
**.
- Avoid deprecated or nonstandard tags and attributes. Use ****important **** rather than the deprecated ****important**** tag. If you do use non-standard attributes and tags, be sure to namespace them.
- Do not improperly nest tags. For example, an **<h2>** tag should not be inside a ** ** tag.

To learn more about page validation, read Ethan Marcotte's article, Where Our Standards Went Wrong *at: http://alistapart.com/comments/whereourstandardswentwrong/. To validate a page, go to http://validator.w3.org/.*

Accessibility

Accessibility has taken on two different but related meanings. Pages are *accessible to those with disabilities* when they provide hooks for assistive technologies. Pages are *universally accessible* when they are viewable by a wide range of user agents, for example, computers, mobile phones, and consumer electronic devices such as a Sony PS3 or Nintendo Wii.

To achieve accessibility, here are general things you can do:

- Set properties such **alt** and **title** on **<image>** and **<link>** tags. These make the page easier to read by screen-readers for the visually impaired.
- Include closed captions for video content. This makes it accessible to the hearing impaired.
- Properly set tab order and specify access keys on form elements and links. These two things make it easier to control a web page using a keyboard.
- Implement the page using standards-based markup. This will ensure that a wide variety of user agents can display the content.

These are just some of the things you can do to make your web pages more accessible. To learn more, visit: http://www.webstandards.org/action/atf/ and http://www.w3.org/TR/WAI-WEBCONTENT/.

Making web pages accessible is not just for assisting those with disabilities, but also for making content readable by machines. Setting alt and title properties as well as metadata in SWF files help with indexing, search engine optimization, and natural language search.

Object and Embed Tags

Flash is embedded with two tags: **<embed>** and **<object>**. The former is a non-standard tag that originally found its way into HTML when Netscape introduced

browser plug-ins. The tag unfortunately never was adopted by the W3C, the body that governs the HTML specification.

The **embed** tag, however, has better cross-browser and cross-platform support since it was fairly well defined at the beginning and all the browser manufacturers implement it the same way. Besides having an unknown future, it invalidates HTML because it's not part of the HTML spec, and it has horrible support for showing alternative content. Its **<noembed>** tag counterpart only works when the client technology doesn't support the tag, which is the case with some mobile web browsers. When the client supports the tag (as is the case with all desktop web browsers), it shows an outline with a broken plug-in and ignores the content set inside the **noembed** tag.

The W3C instead developed the **object** tag because it was less prone to patent issues (see *EOLAS Patent and Active Content* on the following page). The problem with the object tag has been how browser manufacturers have implemented it. Everyone but Microsoft implemented it using the **type** property, which uses MIME file-type descriptions to indicate the object's file type and helper application or plug-in technology. (MIME types have been used since the beginning of the Web to describe file formats to web servers.) Instead of going with this standards-based approach, Microsoft created the proprietary property, **classid**, to identify what Active-X control to use when displaying the plug-in content.

Hopefully the day when Internet Explorer deprecates **classid** while supporting the **type** property will come soon. While the support for the object tag is not as good as the embed tag, it does support alternative content. Assuming the object tag is used for inserting Flash, the content placed between the opening and closing object tags is not rendered when browsers support Flash. When there is no support available for Flash, the content appears. This content can contain a description of the Flash movie and a picture serving as a preview. This content is, however, visible to web crawlers and search engines.

Flash Player Version Detection

The Flash Player is updated with major releases every 12–18 months. When a new release comes out, it takes less than a year for that version to be on the majority of computers connected to the Internet. While Adobe improves the update process with each new release, there is still the need to detect the player version when serving content that relies upon the latest Flash Player release. For example, after the release of Flash Player 8, player detection was crucial when serving Flash Video that used the On2 VP6 codec as it is only available in Flash Player 8 and above.

Player detection is implemented by including JavaScript code that can detect the version of the Flash Player installed on the viewer's computer and comparing it to a variable indicating the player version required to view the content. When the

installed version is equal to or greater than the required version, everything works. When the installed version is less than the required version, viewers see a message communicating that they need to upgrade their player. For users using Microsoft Internet Explorer on Windows, there is Express Install, an Active-X script that does a seamless install for the Flash Player.

The EOLAS Patent and Active Content

This patent lawsuit caused Microsoft to alter the way active content (Active-X controls and plug-ins) are experienced. To comply with the lawsuit, Microsoft had to add a click-to-activate feature inside Internet Explorer. This speedbump, or pane of glass, interrupts and complicates the Web experience for all viewers. To circumvent the click-to-activate feature, web developers can insert the object and embed tags dynamically using JavaScript. While this takes a bit more programming effort, it is far better to do than to force users to have to click a few more times to view content.

Figure 8.3: *What happens when active content is not inserted dynamically.*

Tutorial: Using SWFObject

The following tutorial covers inserting Flash content using the JavaScript library: SWFObject. Back in Chapter 5, we published the custom video player using the Flash Detection Kit, which is a part of Flash CS3. This tutorial will walk you through using markup that is unobtrusive, preserves validation, and provides alternative content. For it we'll use the custom player created in Chapter 5.

SWFObject is a JavaScript library created by Geoff Stearns. To use it, you download the library from http://blog.deconcept.com/swfobject/, include it with your

web site, reference it in the page containing Flash content, and write a few lines of JavaScript and HTML.

ADDITIONAL LIBRARIES

Besides SWFObject, there are other libraries one can use for inserting Flash dynamically. All these libraries help skirt the EOLAS issue, can be applied using unobtrusive scripting, and can be used to replace alternative content. Choosing one is akin to choosing a wine: it partly depends upon your preferences and the entree (or project) you're about to have.

Here are a few other libraries you can consider:

UFO.js, or Unobtrusive Flash Object, was written by Bobby van der Sluis. It's similiar to SWFObject in practice and adoption.

http://www.bobbyvandersluis.com/ufo/

flash.jquery.js was written by web developer and designer Luke Lutman. It's a plug-in for the popular Ajax framework, jQuery. If you plan to work with Ajax, XML, or dynamic HTML, I recommend it. It's frequently updated, so check it out for revisions.

http://jquery.lukelutman.com/plugins/flash/

Adobe's **Flash Detection Kit** is good for those who don't want to write a lot of JavaScript and HTML code and are not concerned with validation and standards compliance. It's built into Flash Professional CS3.

The **SWFFix** library, which is a collaboration between Geoff Stearns and Bobby van der Sluis to offer a best-in-class approach to inserting Flash content. It's still in development, but might be available by the time you read this book.

http://www.swffix.org

Figure 8.4: *Download SWFObject from Geoff Stearns' blog, blog.deconcept.com.*

1. Navigate to the **Tutorials > Chapter 8** folder. Copy the **SWFObject Embed** folder to your computer.
2. Using the HTML editor of your choice, open **index.html**.

3. Insert a new line after line 7 and enter the following script tag:

```
<script src="js/swfobject.js" type="text/javascript"></script>
```

This tag references the SWFObject JavaScript library. By including it in the page, the page can access all the functionality defined within the library.

4. After the opening **<body>** tag, insert:

```
<div id="flashcontent">

</div>
```

This ** ** tag will contain the Flash movie as well as the alternative content. The id (identifier) attribute is a hook for the SWFObject script to replace the content inside it with the Flash movie we will soon specify. For now, we'll use an identifier of **flashcontent**.

5. Let's now place the alternative content inside this ** ** tag. Place the cursor inside the ** ** tag, and enter the following lines of code:

```
<p><img src="assets/alternative_content.jpg" width="480" height="268"
alt="video still from one interview"></p>
      <h2>The video is a short clip from several filmmaker interviews.</h2>
      <p>In order to view it, you need to enable JavaScript and install or
upgrade <a href="http://www.adobe.com/go/getflashplayer" title="Get Adobe
Flash Player">to a newer version of the Adobe Flash Player</a>.</p>
      <p>
         <a href="http://www.adobe.com/go/getflashplayer">
            <img src="http://www.adobe.com/images/shared/download_buttons/
get_flash_player.gif" title="Get Adobe Flash Player" />
         </a>
      </p>
```

The first paragraph tag contains a graphic showing a still from the video and includes a message stating that the video cannot be played. It instructs the user to download the latest version of Flash Player and to enable JavaScript. The text that follows essentially says the same and the code ends with Adobe's Get Flash Player button. With JavaScript off or when an obsolete browser is installed, the page will appear like the following screenshot.

Figure 8.5: *How the page appears when the browser cannot display the Flash content.*

6. Now let's insert the JavaScript to insert the movie. Place the cursor after the closing ** ** tag (probably at the end of line 20 if all is going to plan) and insert:

```
<script type="text/javascript">
   // <![CDATA[

   // ]]>
</script>
```

This script block will contain the JavaScript code for inserting the Flash content. The two forward slashes are single-line JavaScript comments. They prevent the JavaScript engine inside the web browser from interpreting the code on that line. The brackets and CDATA statement instructs any HTML page validator to ignore the content inside the statement and helps with validating the page against a particular document type. So this double-comment technique is a good snippet to use whenever you're writing JavaScript code inside the page.

Ideally most code should be written in an external file and referenced for clear separation of structure and behavior. Since this is a small tutorial, however, it's perfectly okay to mix them up a little.

7. The code to insert the Flash video is quite simple, if not a little terse. Inside the CDATA block, enter:

```
var so = new SWFObject("assets/customplayer.swf", "flvplayer", "480",
"406", "9", "#FFFFFF");
so.addParam("allowFullScreen", "true");
so.write("flashcontent");
```

The first line creates a new SWFObject named **so**. When it creates the object, it specifies the location for the Flash content it will use as well as its identifier, width, height, required Flash Player version, and background color. The second line adds an additional parameter for allowing full-screen mode to work. The last line calls the write method, which does the hard work of taking all the attributes we just passed to it and dynamically writing this content to the web page when it loads inside a capable web browser.

Figure 8.6: *The page displays properly when JavaScript is enabled inside a capable browser.*

On the DVD-ROM are examples of inserting Flash Video using the UFO.js and the flash.jquery.js JavaScript libraries. Look in > Additional Content > Inserting Flash.

Ensuring Your Web Site Can Serve Flash Video

In the case your hosting provider or internal IT-supported web server hasn't registered the Flash Video file format with its servers, you won't be able to serve the files. Flash Video, like JPEG, GIF, or SWF files, are complex file formats (anything

beyond simplified text) and web servers need to be instructed on how to serve them. That's where MIME types (Multipurpose Internet Mail Extensions) come in. A mail client, web server, or web browser uses MIME types to correctly interpret complex file formats. The MIME type for Flash Video is video/x-flv.

If you're running Microsoft Windows 2003 and IIS Server 6.0 and cannot see Flash Video correctly, check out: http://www.adobe.com/go/tn_19439.

Wrapping Up

This chapter covered the issues you'll encounter when integrating Flash Video in a web page. By following open standards, offering alternative content, and using unobtrusive insertion techniques, you ensure that your content is future proof, better optimized for search engines, and more accessible.