

# Duke Web Production Standards and Best Practices

---

## Overview

The Duke Web Standards are designed to help web developers at Duke create and maintain websites that are broadly accessible, easy to use, and efficient to maintain. The guidelines are divided into the following sections:

- **Standards for Web Documents**  
Provides details on Duke's suggested Web Standards (XHTML, CSS, JavaScript) and suggested Page Coding Standards at Duke
- **Web Accessibility**  
Provides summary-level and detailed descriptions of Duke's suggested accessibility standards for websites. Much of this information is adapted from the W3C Forum's [Web Accessibility Initiative](#).
- **Web Security**  
Provides several top-level considerations for safe coding practices, as well as a recommend listing of Security Resources.
- **Practical Examples: Coding for Plugins**  
Provides several extensive examples of real-world code to use when embedding plugin-based media files (particularly Encoding Flash media, Embedding Quicktime Media, and Embedding Real Media)
- **Useful Tips: Ways to Keep Your Site Clean and Easily Maintained**  
Provides general principles on designing and implementing usable and accessible sites. Each tip also provides references for more in-depth research.
- **Reference: Compatible Browsers and Platforms**  
Provides a reference listing of browsers and platforms on which Duke websites should render successfully.
- **Reference: Duke's Web-Services Offices**  
Provides a reference listing of offices that provide web services for specific communities within Duke.

In general, these standards are written primarily for intermediate to experienced web developers. If you are developing a web project at Duke and you would like assistance with these standards, or you feel that your project may conflict with these standards because of special considerations, please contact your department's or division's web-services office. See Reference: Duke's Web-Services Offices.

Much of the material for these guidelines is based on the Web Content Accessibility Guidelines, version 1.0 ([full guidelines](#), [checklist](#)), developed by the [W3C \(World Wide Web Consortium\)](#), the chief standards body for the Web.

If you would like to join a community of Duke developers to discuss these standards, or to post inquiries about web standards-related questions, please send an e-mail to [majordomo@duke.edu](mailto:majordomo@duke.edu) with the following line in the body of the e-mail (without quotation marks): "**subscribe web-standards [your e-mail address here]**".

## Standards for Web Documents

The following topics cover various aspects of web pages and websites. Each topic provides general recommendations for proper usage, any identified "best practices" for Duke websites, as well as any useful applied examples of the topic. In general, however, all of the topics here follow the general guideline that **the easiest way to make your site accessible and usable is to write your code compliant to the web standards you employ**. Following this simple dictum should make your web site easier to use, easier to maintain, and longer-lasting.

### *General References:*

An excellent overview on making standards-compliant websites that are supported well by most browsers can be found at [Apple's website](#). This page also provides a number of good articles on various aspects of good website design and implementation.

Validating your website against the current standards is also an excellent way to ensure compliant websites. The World Wide Web Consortium has [several excellent validators](#) you can use. Some browsers, such as Mozilla and Firefox, have validator extensions available to them (such as [Checky](#)).

## Web Standards (XHTML, CSS, JavaScript)

### XHTML

All HTML pages produced by Duke web developers should conform to the XHTML 1.0 Transitional standard. There are many benefits to using this standard, but the foremost one is that all newer browser versions are being written to this standard, thus ensuring a longer shelf life for any pages written compliant to XHTML. A few highlights of this standard to keep in mind:

- All tags must be closed. Include a space before the trailing / and > of empty elements, e.g. `<br />`, `<hr />` and ``.
- Keep all tags and their attributes in lowercase.
- Avoid line breaks and multiple white space characters within attribute values. These are handled inconsistently by user agents.
- All attributes within tags must be enclosed by quotation marks.

### *XHTML References:*

The full specification for this standard is available at [the W3 Consortium](#). [A concise set of guidelines](#) for this standard is also available.

### Cascading Style Sheets

Wherever possible, Duke web developers should try to employ Cascading Style Sheets (CSS) extensively, as CSS coding currently allows the most promising method of separating content from presentation. Ideally, the entire stylesheet definition should be encapsulated with one or more "[link](#)" tags that reference external stylesheet files (typically files ending with the extension ".css"). HTML files should not use the `<font>` tag at all--or any other deprecated elements--and style attributes should be replaced with the use of class definitions.

### *Web Standards Highlights*

- Use XHTML Standards
- Use CSS to separate content from presentation
- Use external CSS and JavaScript files

An example of the ideal CSS section of an HTML document is one or more of the following lines within the `<head>` section of the HTML document:

```
<link href="mystyle.css" rel="stylesheet"
type="text/css" />
```

### *CSS Tips:*

As a rule, web developers should consider using relative sizing for font sizes ("em"s, percentages, etc.), so that users may effectively change font sizes on their browsers. In addition, developers should ensure that any "font-family" definitions use a very robust list of font families, to ensure broad compatibility across browsers and platforms.

### *CSS References:*

An excellent reference point for CSS issues and tutorials is the W3C's forum on "[Learning Cascading Style Sheets](#)."

### JavaScript

As with most client-side technologies, JavaScript has helped developers extend the functionality of their sites, and its presence has become ubiquitous in websites. As with any client-side technology, however, a web developer cannot be assured that a particular user's browser will use JavaScript at all or be able to parse the site's particular version of JavaScript correctly. Consequently, web developers should code their JavaScript functions as defensively as possible. At a minimum, any web page developed with JavaScript in its navigation functions (menus, rollovers, etc.) should be able to degrade effectively for users who have turned off JavaScript options in their browsers (this can often be accomplished simply with a "`<noscript>`" section in the page to complement the "`<script>`" section).

Where possible, all JavaScript code should reside in an external document (typically ending in the extension ".js"), and be linked to using a `<script>` tag in the `<head>` of the document. JavaScript functions can be initiated from within the document, but function code should be kept to external files only. An example of the ideal JavaScript section of an HTML document is one or more of the following lines within the `<head>` section of the HTML document:

```
<script src="myscript.js" language="JavaScript"
type="text/JavaScript"></script>
```

## Page Coding Standards at Duke

### DOCTYPE Declarations

DOCTYPE declarations are one of the most powerful tools to ensure predictable and consistent website rendering across browsers. A properly-specified DOCTYPE declaration indicates to the browser what standard you're using and what mode you would like the browser to use in rendering the page. To conform to the XHTML standard, the following HTML code should appear on the top of every document:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Special Note: Internet Explorer (5 and 6) have problems understanding the initial XML declaration line, and if this line is present the browser goes into "quirks" mode, which may wreak havoc with pages designed with CSS box-layouts. Although removing the XML declaration on a page may conflict with the page's character set if it uses non-standard character-encoding (see Meta Tags), if the XML declaration is causing rendering problems in Internet Explorer, you should remove the declaration.

### Forms

Forms are one of the major information-gathering tools on the Internet, and they can be used quite effectively on a site to enhance a user's visit to the site. They can also be used as a feedback device that provides an alternative to posting e-mail addresses on a website (and thus exposing that address to spam harvesting--see the "Web Security" section). Yet forms are also one of the hardest web tools to implement in an accessible and efficient way. Forms require web developers to organize their content in logical (and not always intuitive) ways, and they provide considerably less flexibility in separating content from presentation. The following tips may help when conceptualizing and designing forms on Duke sites:

- Whenever possible, use the "POST" method for forms submission. This tends to keep the URL addresses cleaner, and makes it more difficult to intercept or override form submissions;
- Avoid nesting `<form>` elements within each other. Not only do nested forms complicate already-unpredictable browser behavior when the user presses the "Enter" key to submit a form, but it can also be equally difficult to predict what information will be submitted.
- Looking at a web page as a tree of elements, it is advisable to keep all `<form>` leaves as close as they can be to the `<body>` branch on a web page. Keeping all the form elements grouped closely together in the page's code is also a helpful organizational technique.

### *General Coding Practices*

- Use DOCTYPE declarations
- Avoid nesting forms
- Use meta tags for clarity or robot exclusion

- Use the `<label>` element generously when designing forms to keep a field's label and its input area informationally-related. This is particularly important for accessibility concerns when a form uses a table to keep its labels and fields aligned with each other, but more generally this practice also encourages browsers to present field's labels in logical ways. The following is an example of using the `<label>` element to explicitly tie two table cells together (the example also uses the `"tabindex"` attribute to help browsers guide users through a form):

```
<tr><td><label for="TITLE1">Title:</label></td>
<td><input type="text" name="TITLE1" id="TITLE1"
tabindex="1"></td></tr>
```

### Meta Tags

Meta tags ensure that browsers and search engines understand the content and structure of the page more completely. The following tags are recommended for all sites (please note that the actual `"content"` entries are for illustration only):

```
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Language" content="en-us" />
<meta name="keywords"
content="Duke, University, news, duke
university, duke news, office of news and
communications" />
<meta http-equiv="description" content="Important news
and communications about Duke University" />
```

For internationalization purposes, please note that if the `"charset"` specified is not the default `"UTF-8"` or `"UTF-16"` encoding, then an XML declaration must be used in addition to the `<meta>` tag.

The following meta tags, utilizing the `"Robots"` meta tag, are suggested but not required for Duke web pages. The advantage to these tags is that they explicitly direct search engines to use (or not to use) the current page. Only one of the following tags should appear on a page (they direct a search engine either to index and follow all links on the page, to not index the page but follow all links on the page, to index the page but not to follow the links on the page, or not to index or follow the links on the page, respectively):

```
<meta name="robots" content="index, follow" />
<meta name="robots" content="noindex, follow" />
<meta name="robots" content="index, nofollow" />
<meta name="robots" content="noindex, nofollow" />
```

If you would prefer that your entire site use one of these options for particular search engines or particular users, you may want to consider implementing a `"robots.txt"` file on your site's web server.

### Meta Tags References:

For more on the use of the robots Meta tag, see [the robotstxt.org website](http://the.robotstxt.org). This site also provides useful information about the syntax and use of "**robots.txt**" files, which are more widely accepted by search engine crawlers.

## Recommended Coding Guidelines (Best Practices)

### File-Naming Conventions

To preserve a level of consistency and compatibility within and across web sites, we strongly recommend that all files for a site should be named with lowercase characters only, with no spaces in the name (underscores are acceptable replacements for spaces), and they should have descriptive filenames. All HTML files should end in ".html" only; all external CSS files should end in ".css", and all external JavaScript files should end in ".js". These standards will ensure the highest degree of accessibility across browser types.

In general, when referencing documents within your site, we strongly recommend that you use relative path references (either root- or document-relative) rather than absolute URLs. This helps ensure that your site will be portable and resistant to domain-name changes.

(NOTE: Some users may find that Dreamweaver on a PC will try to save HTML pages by default with the ".htm" extension. If you would like to change the default extension for HTML pages, please visit Macromedia's [support page on this topic](#).)

### Comment Tags

It is always good practice to comment your code extensively. Although HTML files are not always the most complex documents, strategically-placed comments can make editing the page much easier, particularly if you find yourself editing someone else's code. At a minimum, include comments indicating the beginning and/or ending of sections of the document (e.g.

"<!-- **Begin Page Header with Logo** -->"), and provide at least brief comments for any style or JavaScript definitions in a document. Comment tags should also be used to highlight specific areas of content or functionality that may later be transformed into components of templates used by a CMS. In general, it is not a good idea to delete comment tags unless they are obviously dated. It is also good practice to use comment tags to *temporarily* remove content from pages in lieu of deleting.

### Screen Resolution

Given the wide variance in today's web page display sizes and dimensions, it is impractical to suggest a standard page size for Duke web pages. Instead, developers are best-advised to keep their audience in mind as they design their site, and they should try to balance the needs of flexible display for accessibility with the design needs of the site's aesthetics. (Developers may even want to

### *Recommended Coding Guidelines*

- Use consistent file-naming conventions
- Use comment tags to organize a web document
- Try to make the size of your site pages as flexible as possible
- Use ISO codes for special characters

consider developing a checklist of their audience--browsers, bandwidth usage, resolutions, etc.--to determine this balance.)

Web developers may want to consider adopting the concept of scalable sites, sites that can be written with an ideal of "liquid" positioning so that users with small screens (or small resolutions) can still fit most of the content within their browser window.

Centering a site in all browsers is often an excellent strategy to ensure maximum visibility for a site--an easy way to accomplish this is to consider using a "**text-align: center**" definition in the "**body**" class definition in a CSS stylesheet. Centering a site is also an excellent way to support scalability of resolutions, especially for sites using fixed-width tables (although fixed-width tables can prove problematic when printed from browsers, often cutting off the right side of the page in the output).

### **Special Character Usage**

Special characters should use the ISO version at all times. The exceptions are single and double quotes appearing in content areas ( ' and " , respectively) which do not need to be represented by their ISO equivalents, as most browsers can parse these characters effectively.

#### *Special Character References:*

A chart of the ISO equivalents is available at [mountaindragon.com's website](http://mountaindragon.com's). For more information on ISO equivalents and the XHTML standard, see the [Entity Sets appendix](#) to the XHTML specifications.

## Web Accessibility

Duke University is committed to making the web more accessible to all. We understand the term "accessibility" to mean "the ability of a heterogeneous set of users to access the content of a Web site." This includes users with disabilities such as impaired vision, users with older browsers (who may not be able to upgrade because of old hardware or other issues), text-browser users, and users of other, specialized browsers. Duke web developers who are able to satisfy the following list of priorities will have ensured that they meet minimal accessibility standards.

### *Web Accessibility Highlights:*

- Consult the W3C Forum's [Web Accessibility Initiative](#)

#### *Priority One:*

- [Text equivalents for all essential non-text content](#)
- [Pages "transform gracefully" in varied user environments \(or provide an alternate page with equivalent content\)](#)
- [Test for cross-browser/cross-platform compatibility](#)
- [Pages "validate" for accessibility \(using a tool such as Bobby\)](#)
- [Pages optimized for bandwidth \(file size\), with total page size under 50K where possible](#)
- [If essential content relies on plug-ins \(e.g. Shockwave/Flash, Acrobat\), provide an alternate, accessible version](#)
- [Avoid making meaning dependent solely on color](#)

#### *Priority Two:*

- Ensure that foreground/background colors have sufficient contrast when viewed by users with color deficits or on a black and white screen
- HTML code "validates" based on W3C specifications (using a tool such as the [W3C's HTML validator](#))

#### *Accessibility References:*

An excellent resource to explore accessibility issues is the W3C Forum's [Web Accessibility Initiative](#); make sure to explore their [web content accessibility guidelines](#)), and in particular the guidelines' [Appendix on Testing and Validation](#).

An excellent validator service is [Bobby](#), an online page validator that focuses on accessibility issues and guidelines. Because of the comprehensive nature of the tests, Bobby test results should be taken as a useful reference tool, but not as an absolute guide.

The W3C has several good validator services as well, tailored to specific standards. Try out the [W3C HTML Validator](#) for (X)HTML compliance, and the [W3C CSS Validator](#) for CSS compliance.

The Web Design Group has also developed an easy-to-use [WDG HTML Validator](#) to check your (X)HTML code.

For even more information on website testing and validation, see the W3C Guidelines [Appendix A- Validation](#).

### Priority One Details

#### Text-equivalents for non-text elements (W3C Checkpoint 1.1)

All content contained in non-text elements (graphics, multimedia, dynamic menus) should be accessible to non-graphic browsers. Techniques include:

1. Include descriptive alternate text ("alt tags") for images and client-side image maps (image "hot spots"). Alternate text should be meaningful: "Girl using a spectrograph." is a good example of alternate text for an image; "photo" is not.
2. Multimedia files should have a text equivalent, particularly if they include essential content. (See "Practical Examples: Coding for Plugins")

Note - Equivalents for graphical/non-text content must be updated at least as often as the graphical/non-text content. ([W3C checkpoint 6.2](#))

#### Pages that Transform Gracefully (W3C checkpoint 6)

All pages should "transform gracefully"- i.e. be accessible when JavaScript, Cascading Style Sheets, applets, images, plugins, or other new technologies are not supported or turned off. If this is not possible, an alternate page with equivalent content should be provided, and should be updated as frequently as the primary page ([W3C, 6.2](#)). Techniques for testing:

1. Disable advanced functions (JS, CSS) in your browser's preferences
2. View the page using the Lynx text browser (accessible on most UNIX servers)
3. Validate the page with an online accessibility-checker such as Bobby

#### Test for Cross-browser, cross-platform compatibility:

Pages should function correctly in a variety of environments, including different platforms (Mac, PC, Unix/Linux), browsers (Netscape, Internet Explorer, Opera), screen resolutions, and color depths (256 colors, thousands of colors, millions of colors). Duke has drawn up a compatibility listing of browser and platform specifications for web developers to consult; see "Reference: Compatible Browsers and Platforms" for more details.

Following the "transforming gracefully" guidelines often resolves most cross-browser/cross-platform issues. Techniques to implement this priority:

1. Test early and test often. If you don't have access to machines on different platforms, try contacting one of Duke's web services organizations to use their testing labs (the Sulzberger Lab, OIT's Office of Web Services, etc.). Your own machine can also be set to different screen resolutions and color depths for testing
2. Avoid using proprietary techniques/tags (techniques that work only in a particular browser).

### *Web Accessibility Priority One Highlights:*

- Text equivalents
- Pages "transform gracefully"
- Test for compatibility
- Validate pages
- Optimize pages
- Plugins and alternatives
- Color and meaning

### **All pages should "validate" for accessibility: ([WSC, Appendix A](#))**

Various online tools are available to test the accessibility of your pages. See the reference section in Web Accessibility for information on recommended validator resources.

### **Pages optimized for bandwidth (file size):**

The total size of a Web page is calculated by adding the size (in kilobytes, or K) of the HTML file and all included files which must load with the page, e.g. images, imported scripts, applets, flash movies. For example, a Web page which consisted of a 8K HTML file, one 13K banner image, an 8K inline image, and a small (2K) flash button would have a total page size of 31K.

Total page size should be less than 50K if at all possible, with 20-30K or smaller being the goal, especially for pages other than the home page. The smaller your pages (in file size), the faster they load; this particularly affects users with dial-up connections, meaning you should test your pages over dial-up if at all possible.

Techniques:

1. For text content, use HTML text rather than graphics where possible. Using CSS (Cascading Style Sheets), you can achieve fairly precise control over font display.
2. Reuse common graphic elements among pages on your site. A user only has to download an image file or plug-in file once--after that it is placed in the browser's cache, and can be reloaded almost instantly. For example, a Web page consisting of a 6K HTML file and a 13K banner image would have a total size of 19K. However, if the banner image were the same as the banner on a previous page, the image would already be in the browser's cache and would load right away, making the effective size of this page only 6K.
3. Optimize all images using a good image editor. Duke has a number of software packages licensed at academic pricing for students and staff available at the [Duke computer store](#) or through the [OIT site license program](#).
4. Never resize an image in HTML or using Dreamweaver (by changing the width and height attributes of the `<img>` tag). This does not change the native size of the image file, but only the size at which it is displayed. This leads to wasted bandwidth (because the user is downloading a file that is much larger than necessary) and also creates delays while the user's browser re-renders the image from the native to the display size. Always resize your image files to the appropriate dimensions in an image editor.

### **Essential content should not rely on plug-ins unless an alternate, accessible version is provided: ([WSC 1, WSC 11](#))**

Using plugins such as Shockwave/Flash, QuickTime, Real Player, or Adobe Acrobat is an excellent way to add multimedia content to your site and enhance the experience of your users. However, since these plugins are not accessible to all users, they should be used only for nonessential site content. If you choose to convey essential content (e.g. navigation, important departmental information,

announcements) using a plugin, you must provide an alternate, broadly accessible version of this content which is updated as frequently as the plugin version.

Techniques:

1. PDF files should be paired with an HTML or plain text equivalent
2. Links to files requiring plugins should be annotated to include the type and size of the file, e.g. "manual (PDF, 45K)" or "dog barking (RealAudio, 100K)", with links to accessible alternatives also listed and annotated, where available.
3. Include accessible captions, transcripts, or detailed descriptions for audio or video files which contain essential site content. ([WSC 1.1,1.3-1.4](#))

For more information on how to embed plugin content, see the "Practical Examples: Coding for Plugins" section.

### **Avoid making meaning dependent solely on color : ([WSC 2](#))**

"If color alone is used to convey information, people who cannot differentiate between certain colors and users with devices that have non-color or non-visual displays will not receive the information" (Quote from the WSC Guidelines).

Techniques:

1. Information conveyed by color should also be conveyed through context or document structure ([WSC 2.1](#))

## Web Security

While most Duke web developers do not have to deal with web security on a daily basis, everyone should be aware of the security issues inherent in websites and web-based technologies. This section cannot be comprehensive in its discussion of security issues, as the specifics change frequently and are subject to constant revision. Consequently, web developers are strongly encouraged to use the resource links provided in the "Security Resources" listing, and consult these as often as necessary.

Nonetheless, the following topics cover some basic, and often overlooked, security issues related to developing and maintaining websites:

- **Protect database and filesystem access information.** If your site uses connections to password-protected utilities such as databases or encrypted filesystem areas, it is important to shield the access information from web browsers. While it may be convenient to write usernames and passwords directly into JavaScript or scripting-language functions (e.g. PHP, ASP, etc.), this is a serious security breach, because any document publicly accessible by the web server is a potential target for hackers who can harvest access information. A much better solution is to work with your web server's system administrator to save access information in offline or CGI filesystem areas, where access is much more restricted.
- **Use JavaScript code sparingly and carefully.** Most JavaScript code libraries on the web provide a number of useful and benign functions that help extend a website's functionality. However, always keep in mind that many of known security exploits use JavaScript code to perform security breaches on networks or on personal computers, particularly in a Windows environment. If you don't understand what a JavaScript code does, it's generally not a good idea to embed it in your site.
- **Be aware of e-mail harvesting.** Any e-mail address on a web page is potential fodder for e-mail "harvest" lists, in which a web spider crawls through sites and uses a variety of algorithms to detect e-mail address patterns. These lists are then sold to companies for mass e-mailings (i.e. spam).

A large number of techniques have been developed over the years to combat e-mail harvesting, but most have proved ineffective in the long run. The [most recent proposal at Duke](#) is to mask the e-mail address, following the results of a [2003 study](#) by the Center for Democracy and Technology. Another feasible approach is to use an online form to solicit feedback, comments, and correspondence. If you use a simple mail-sending form technology, such as the [mailform script](#) offered by the Sulzberger Lab, you can avoid posting any contact e-mail addresses on a page permanently. Another possibility is to have all contact information reside on dynamically generated pages (server-side technologies), which can discourage access by harvest robots that focus on static web pages. In general, however, the best approach you can take is simply to

### *Web Security Highlights:*

- Protect databases and filesystems
- Use JavaScript carefully
- Understand e-mail harvesting
- Understand forms security
- Use privacy policies
- Read up on advisories

understand that a contact e-mail address on a web page makes that address publicly available, with all of its benefits and limitations.

- **Be aware that unencrypted forms submit information in the clear.** Most forms on most web sites are stored and viewed from normal, unsecured web server URLs (http). As a result, you should assume that any information submitted on these forms is transmitted as clear text, without any security precautions. For the majority of a website's forms, this lack of security is fine. If your site needs to collect confidential or commercial or private information, however, this is a glaring security breach. The best solution is to save any secure forms to a secure web server URL (https).
- **Develop and post a privacy policy where applicable.** Privacy policies are particularly useful for websites that use personal information or collect any personally-identifiable information (including form information for e-mail contacts), but can be useful for almost all websites. At a minimum, a site should indicate how it uses the information it collects or displays, as well as under what authority it makes claims or statements. There is currently no Duke-wide standard for a privacy policy format, but many major Duke websites provide policies that adapt well to other sites.

### *Security Resources (additional reading)*

The following resources at Duke and on the web provide further detail on the subjects covered in this section, as well as up-to-date advisories on current security concerns:

- [Duke's Arts and Sciences Web Programming Policy Guide](#);
- [Duke's Arts and Sciences CGI Programming Guidelines](#);
- [Duke Security and Privacy Policy Guidelines](#);
- "[Security in CGI Programming](#)," by Duke Math Department programmer Dr. Yunliang Yu;
- [Open Web Application Security Project](#) (look particularly at their [Top Ten list](#) of web applications vulnerabilities);
- [SANS top 20](#) (an updated list of the "The Twenty Most Critical Internet Security Vulnerabilities");
- [CERT](#) guidelines and advisories on website development and security

## Practical Examples: Coding for Plugins

### Plugins in General

A number of portable browser-based applications (plugins) have been developed for various browsers and platforms that extend the functionality of a browser's rendering of a website. Some of the most popular plugins, such as Adobe's Acrobat Reader, have become so ubiquitous that they have become part of a browser's standard installation on most platforms, and do not require any special coding to link to documents created for the plugin. Others--such as Macromedia's Flash player, Apple's Quicktime player, or Real Media's player--require special coding considerations to work properly in most browsers. In terms of streaming media, Duke officially supports two different plugin technologies to support to allow developers to stream multimedia content from Duke servers: Quicktime and Real media. (Duke developers can find out more about the University's streaming media services via the [OWS media services pages](#).)

In general, it is wise to code defensively when using plugin-based documents in a website. Accordingly, any page on a site that links to or embeds a plugin document--a Quicktime movie, a Flash file, a Real media stream, a LUNA image, etc.--should also contain some method of providing access to the plugin player that will allow the browser to view the document. In addition, for plugins that require embedding the document within a page (such as with Flash, Quicktime or Real media), the page should include code to allow the browser to detect and update its plugin player automatically (at least for those browsers that support this feature). It is also good practice to include the logo of the plugin player with this link.

When encoding plugin media on a web page, the general format is to provide the encoding within an `<object>` tag, with an `<embed>` tag nested within it with the same parameters. This configuration allows as much cross-browser compatibility as possible, and it tends to provide the most consistent behavior for auto-updates through the `<object>` parameters "classid" and "codebase" and the `<embed>` attribute of "pluginspage".

Lastly, a word of caution about plugins and accessibility. While the makers of various plugin applications are improving the ways in which their media can provide multiple and alternative formats, plugins by their very nature are quite restrictive at delivering content, and a site's audience may be incapable of interpreting the media itself, or even accessing a plugin technology. Duke web developers need to be quite conscious of this fact at all times, and consider whether the plugin media is necessary, and if so, whether there are efficient ways to deliver the same content in alternative formats. One possibility for improving accessibility in multimedia documents is the use of captioning within the document.

### General Plugins References:

For detailed guidance on the rules surrounding the `<object>` tag, see [Netscape's Introduction to Plugins](#) and [W3C's Treatment of Objects](#). For specific plugin types not covered in these web standards (for example,

### *Coding for Plugins Highlights:*

- Consider alternative formats
- Provide download links
- Consider captioning
- Use both `<object>` and `<embed>` tags
- Consult our coding examples for Flash, Quicktime or Real media

LUNA or MrSID images), please visit the plugin programs' websites for embedding information.

For information on how to caption embedded multimedia documents, the following URLs provide good tutorials from which to begin experimenting: [WebAim.org on captioning](#) (provides description of captions and links to tutorials for captioning various types of multimedia files); [MagPie tutorial](#) (captioning with MAGpie 2, with links to tutorials for RealPlayer, Quicktime, and Windows Media); and [Hi-Caption and Flash](#) (using Hi-Caption software to caption Flash media)

### Encoding Flash media

While Flash enjoys wide usage in websites, it is generally advisable to avoid using Flash to drive any core functionality of a site (for Web Accessibility reasons). For any site that proposes to use Flash for navigation, logo usage, or even general content, the developer should strongly consider business cases for doing so, and make sure to tailor the Flash media to the site's audience.

Typical code for a Flash 6.0 movie would be as follows. Most of the parameters specified in these code excerpts can be changed to suit specific needs--the only exceptions are parameters like "**classid**", "**codebase**", and "**pluginspage**", which must appear exactly as shown. As with all plugins, remember to make sure that the **<object>** parameters and **<embed>** attributes contain the same values.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,40,0"
  width="749" height="68" id="sample_movie"></span>
<param name="movie" value="sample_movie.swf"><param
  name="quality" value="high"><param name="bgcolor"
  value="#FFFFFF"><param name="menu"
  value="0"><embed src="sample_movie.swf"
  quality="high" name="sample_movie"
  pluginspage="http://www.macromedia.com/go/getflashplayer"
  type="application/x-shockwave-flash"
  width="749" height="68" menu="0"></embed>
</object>
```

#### *Flash Plugin References:*

Macromedia provides a discussion of the [best way to embed Flash and Shockwave movies](#).

### Embedding Quicktime Media

There are two primary ways to provide links to Quicktime media: embedding the media on a web page, or providing a link to the media to be played in an external Quicktime player application. The following code snippets provide examples of both techniques. Most of the parameters specified in these code excerpts can be changed to suit specific needs--the only exceptions are parameters like

"**classid**", "**codebase**", and "**pluginspage**", which must appear exactly as shown. As with all plugins, remember to make sure that the **<object>** parameters and **<embed>** attributes contain the same values.

### *Quicktime Plugin References:*

Comprehensive coverage for all the parameters involved in embedding Quicktime media can be found at [Apple's website](#). Apple also provides a [good overview](#) of how the **<object>** and **<embed>** tags differ with regards to Quicktime implementation. For details on the "**qtsrc**" addition for non-"**mov**" media, see Apple's [Quicktime Web FAQ](#). Apple also provides a [step-by-step guide](#) to developing **QTL** Reference movies.

### **Option 1: Quicktime media embedded in browser window**

Use this code excerpt to provide Quicktime media embedded in the browser window itself. Please note that, in Internet Explorer, Quicktime movies will not play or download the Quicktime player if Active X is disabled or unavailable. In general, it is advisable to add 16 pixels to the height of the movie to allow room for the movie controls.

```
<object classid="clsid:02BF25D5-8C17-4B23-BC80-
D3488ABDDC6B" width="160" height="144"
codebase="http://www.apple.com/qtactivex/qtplugin.cab"
>
<param name="src" value="samplemovie.mov" />
<param name="autoplay" value="true" />
<param name="controller" value="true" />
<param name="loop" value="false" />
<param name="name" value="title of movie" />
<embed src="samplemovie.mov" width="160" height="144"
autoplay="true" type="video/quicktime"
controller="false"
pluginspage="http://www.apple.com/quicktime/download/"
loop="false" name="title of movie"></embed>
</object>
```

Please note that if you are trying to embed Quicktime streaming media from an **rtsp** server (or any non-"**.mov**" media that you want to play in an embedded Quicktime player), you will need to add an additional "**qtsrc**" parameter to the code snippet above (the "**src**" lines do not need to be changed and can point to valid or even a bogus "**.mov**" URL, since they become simply placeholders to indicate to the browser to launch the QuickTime player). Within the **<object>** area, you will want to add the following line for an RTSP stream:

```
<param name="qtsrc"
value="rtsp://quicktime.oit.duke.edu/samplemovie.mp4"
/>
```

Within the `<embed>` statement, you will want to add the following name/value pair after the "src" name/value pair:

```
qtsrc="rtsp://quicktime.oit.duke.edu/samplemovie.mp4"
```

### Option 2: Launching Quicktime Media in the Quicktime player

This option provides the ability to link to Quicktime media and have the media play in the standalone Quicktime player (as opposed to using the Quicktime browser plugin). To accomplish this, you will basically create a hyperlink to a "reference movie" that has the extension ".qtl" (QuickTime Media Links). An easy way to create this reference movie is by creating a new file with the extension ".qtl", and inserting the following code excerpt into this new file, and have the desired web page(s) link to this file (using a standard `<a href="sample.qtl">` link). Please note that the "src" parameter in the `<embed>` tag must be a full URL--the QTL file cannot parse relative URLs. The content of the ".qtl" file should be as follows (replacing the "src" attribute value with the appropriate URL):

```
<?xml version="1.0"?>
<?quicktime type="application/x-quicktime-media-
link"?>
<embed
src="http://www.duke.edu/movies/sample_movie.qtl"
width="160" height="144" autoplay="true"
type="video/quicktime" controller="false"
fullscreen="normal" quitwhendone="false" loop="false"
name="title of movie"></embed>
```

Please note that this technique assumes two things: the user's version of Quicktime is 5.x or higher, and the web server hosting the file with the link has a ".qtl" MIME-type association (the association is "`application/x-quicktimeplayer`"). If either of these conditions are not met, the user will see an XML code page when they click on the link, or their Quicktime player may report file-compatibility errors. Currently, the "`www.duke.edu`" web server at Duke supports the ".qtl" MIME-type.

### Embedding Real Media

There are two primary ways to provide links to Real media: embedding the media on a web page, or providing a link to the media to be played in an external Real player application. The following code snippets provide examples of both techniques. Most of the parameters specified in these code excerpts can be changed to suit specific needs--the only exceptions are parameters like "classid" and "type", which must appear exactly as shown. As with all plugins, remember to make sure that the `<object>` parameters and `<embed>` attributes contain the same values; this is particularly true for the "console" parameter/attribute, which can be used to tie an embedded Real console to an embedded Real media screen.

#### *Real Media Plugin References*

Comprehensive coverage for embedding Real media and specifying all the possible attributes can be found in a series of Real guides on [web page embedding](#), [embedding clips in web pages \(quick guide\)](#), [playing clips in a web page](#) and [delivering streaming Real media](#).

Details on using JavaScript to control embedded Real media consoles can be found in the [RealOne Player Scripting Guide](#). A simple guide to setting up Real media embedding in a browser window can be found at [streamace.com's website](#).

#### **Option 1: Real media embedded in browser window**

Use this code excerpt to provide Real streaming media embedded in the browser window itself. As with most forms of embedded media, the HTML code should include both `<object>` and `<embed>` elements, but this time with a twist: for embedded video you need separate pairs of `<object>` and `<embed>` tags for the video display and the video playback controls (for audio media, only the playback control are necessary). In the example code provided below, make sure to replace the "name\_of\_movie.rm" reference with an actual Real media reference (which can be an actual ".rm" file, or an intermediate specification file that points to web-served Real media, such as text ".ram", ".rpm", or ".smil" files).

```
<!-- Real media embedding code -->
<!-- This object/embed pair contains the display
information -->
<object id="RVOCX" classid="clsid:CFCDA03-8BE4-11CF-
B84B-0020AFBCCFA" width="320" height="240">
<param name="src" value="sample_movie.rm">
<param name="autostart" value="true">
<param name="controls" value="ImageWindow">
<param name="console" value="video">
<param name="maintainaspect" value="true">
```

```
<embed src="sample_movie.rm?embed" type="audio/x-pn-
realaudio-plugin" controls="ImageWindow" width="320"
height="240" nojava="true" console="video"
autostart="true" maintainaspect="true" />
</object>
<!-- This object/embed pair contains the playback
information -->
<object id="RVOCX" classid="clsid:CFCDA03-8BE4-11CF-
B84B-0020AFBCCFA" width="320" height="36">
<param name="controls" value="ControlPanel">
<param name="console" value="video">
<embed src="sample_movie.rm?embed" type="audio/x-pn-
realaudio-plugin" controls="ControlPanel" width="320"
height="36" nojava="true" console="video" />
</object>
```

In the example above, the only attributes that should always be included in the `<embed>` element are `"src"`, `"type"`, `"width"`, and `"height"` and `"console"`--the other attributes are provided as examples and can be used or discarded as needed. Note the `"?embed"` parameter appended to the Real media reference in the `"src"` attribute of the `<embed>` element: this parameter specifies to the Real player that the media will be embedded in the web page. Also note that there are two `<object>` and `<embed>` elements included above--the first (with the parameter `"controls='ImageWindow'"`) provides access to the actual RealMedia, while the second (with the parameter `"controls='ControlPanel'"`) provides the actual controls to manipulate the RealMedia stream or file (play, stop, clip information, etc.). For the latter `<object>/<embed>` pair to control the former pair, the `"console"` attribute must be the same value in both pairs.

With the `<object>` element, make sure to include the `"classid"` exactly as shown above, and be sure to include the equivalent `<param>` elements to those specified in the `<embed>` element. The only exception to this statement is that with `<object>` elements all sharing the same `"console"` value, the `"src"` parameter is only needed in one `<object>` element. Every `<embed>` element in the `"console"` group must have the same `"src"` value.

Lastly, given Duke's use of streaming media through RTSP (Real-Time Streaming Protocol), it is important to clarify what URL to use when referencing a media file hosted on Duke media servers. For all media for which you would like to embed a Real player in a browser window, the most efficient solution is to use the RTSP URL to point directly to the media stream, e.g.:

```
<param name="src"
value="rtsp://rtspserver.duke.edu/sample_movie.rm">
```

This is the most effective way to point to the file and serve it up using the secure RTSP stream. However, if you wish to launch the media file in an external Real

Player instance (see "Option 2: Launching Real Media in a Real player"), then you will need to use a normal, and fully-qualified, HTTP URL, using the "RAMGEN" option (which dynamically creates a Real media reference file pointing to the RTSP server). A sample URL of this would be:

```
<a  
href="http://rtspserver.duke.edu/ramgen/sample_movie.r  
m">Link</a>
```

### **Option 2: Launching Real Media in a Real player**

Launching Real media from a web page so that it plays in its own player is much simpler than embedding Real media on a web page. Typically the only HTML code necessary is a simple link to the Real media reference (which can be an actual ".rm" file, or an intermediate specification file that points to web-served Real media, such as text ".ram", ".rpm", or ".smil" files):

```
<a href="sample_movie.rm">Link to Real media</a>
```

If you are using Duke's Real streaming server, then you will want to use a fully-qualified HTTP URL, using the "RAMGEN" option (which dynamically creates a Real media reference file pointing to the RTSP server). A sample URL of this would be:

```
<a  
href="http://rtspserver.duke.edu/ramgen/sample_movie.r  
m">Link to Real media</a>
```

If the user's machine does not have a version of the Real player installed, then the browser will usually prompt the user to pick an application from which to play the Real media file. Because of this, it is always advisable to provide links to the download page for the Real player on any page that provides Real media.

## Useful Tips: Ways to Keep Your Site Clean and Easily Maintained

As most web developers have experienced, there are as many methods to organize a site and to make it browser-friendly as there are web developers out there, and no clear guidelines for what approach is better or more compliant. Rather than provide a single directive for these issues, this section describes a few different techniques employed by both Duke web developers and larger web groups that work to achieve efficient sites that provide optimal rendering in as wide a browser audience as possible. As you will notice, some of the examples are from a valuable online resource, [A List Apart](#), and many of the techniques described here rely heavily on using Cascading Style Sheets (CSS). While CSS takes a while to master, it rewards the developer by allowing considerably more flexibility than using HTML alone.

Regardless of technique, however, the only constant in all of these approaches that bears repeating is this: **the more you write your code compliant to the standards you employ, the easier it will be to maintain your site and make it degrade nicely.** (Conversely, the more you tailor your site to browser-specific features, the harder you will find it to make your site accessible to everyone.)

- **Use CSS to provide scaling sophistication in your site.** If you keep your web pages as simple as possible in terms of content encoding and reserve presentation aspects to CSS, you can develop a hierarchy of external stylesheets for your site that enable advanced style definitions only for compliant browsers, and that still allow an older or non-compliant browser to display the page adequately. This is a very elegant approach to the concept of graceful degradation, and it also requires a considerable amount of planning to implement correctly. For more information: ["Alternative Style: Working With Alternate Style Sheets"](#) ["A Backward Compatible Style Switcher"](#)
- **Use CSS stylesheets to provide alternative (or simpler) formats.** A variation on the technique described above is to use CSS stylesheets for specific media and/or content-browser types. This technique allows for "printer-friendly" formats that could also be used by audiences whose browsers cannot display the primary web browser format effectively. The biggest benefit to this approach is that you don't have to duplicate content to get multiple output formats. For more information: ["Designing for Context with CSS"](#) ["CSS Design: Going to Print"](#)
- **Replace table-based formatting with CSS layout.** Properly coded, websites written without tables (or with minimal tables) can provide an enormous amount of flexibility in design and approach, and allow developers to avoid having to touch HTML web pages to change the way the site looks. Moreover, well-designed sites built on CSS tend to degrade more nicely than table-based site on older browsers. For more information:

### *Site Design and Maintenance Highlights:*

- Write code compliant to the standards you employ
- Use CSS to affect the design and impact of your site
- Use CSS to make your site gracefully degrade
- Consider server-side or template technologies to lower your site's maintenance overhead
- Use subdirectories to organize your content
- Create a style guide for your site to make maintenance tasks easier
- Use a footer for every page
- Use frames sparingly

["CSS Layout Techniques: for Fun and Profit"](#)

["CSS Zen Garden"](#)

- **Use template-based approaches to designing your website.** If your web server makes it available, technologies such as server-side includes (SSI) can dramatically reduce the overhead involved in changing the look-and-feel of your website. Web content documents can use SSI commands to reference stand-alone template code, and allow web developers to make changes to template code documents that propagate throughout the site. Currently the [www.duke.edu](http://www.duke.edu) web server does not have SSI technology enabled.

If you cannot access SSI, another useful alternative is to use a web development software package such as Dreamweaver that allows you to use template functions. Although much of the template management in Dreamweaver is offline, the process of pushing changes throughout a site is similar. For more information:

["Server Side Includes \(SSI\) Tutorial"](#)

["Apache Tutorial: Introduction to Server Side Includes"](#)

- **Use subdirectories to organize your site.** One of the more common strategies for keeping sites manageable and maintenance-friendly is to organize different files in a site into different directories. For medium to large sites--generally sites with 15 or more pages--it is usually a good practice to create content subdirectories where various files with similar content can live together. Aside from creating less cluttered directories, this strategy also has the benefit of providing the option of giving clues in the URLs themselves about the nature of the content being viewed.

In addition, it is generally good practice to place some categories of files in site directories by themselves. Images, stylesheets and scripts are three of the most common categories. Usually files in these three categories are placed in three directories below the root directory, named "**images**", "**css**", and "**includes**". This will ensure that all references to these files, from any location in the site, will point to a common directory that can easily be referenced and is not likely to change. This is particularly useful when using root-relative references, for example:

```

```

- **Create a style guide to document your site's style and conventions.** One of the most beneficial long-term investments you can make in your site is to document your site in a style guide. There is no formal structure to a style guide, but typically the style guide includes basic information on the file structure of the site (its architecture), the visual structure of the site (its layout), and the technical details on the site's fonts, colors, images, styles and common headings used in the site (the specifications). Developers often find it useful to include any programming notes or hints in the style guide, as well as comments on intended audiences or browsers or other assumptions about the site. If the site will transfer ownership from one group to another, the style guide should also include relevant

maintenance instructions and any other filesystem-related issues.

For examples of previously-created Duke style guides, see:

[Sulzberger Lab sample style guide](#)

[Law School sample style guide](#)

- **Use a footer on every page in your site.** Although this advice won't make the site any easier to maintain, having a footer on every page with relevant information--such as contact information, links to services like a site map and search pages, and links to related entities--provides a degree of consistency for visitors to the site that should benefit them enormously. If you provide data such as last-modified dates for each, the footer can also be used to reinforce the "freshness" of your site.
- **Use frames sparingly, and only when appropriate.** Frame technology in browsers can allow you to convey a considerable amount of information in a small space, and can ease complex navigational issues in websites. However, some of the most inelegant, inaccessible and difficult sites on the Web are that way precisely because they use frames. Frames can also pose problems for search engines; Duke's Google appliance, for example, requires that a site using frames must follow the frameset rules strictly, and end-users cannot control whether its search results may point to a frameset or to a specific frame page. Another limitation to frames are that pages within a site are very difficult to reference using URLs, which is a considerable problem if you'd like to have others link to content on your framed site.  
If you choose to use frames in your website, be aware that you should use a different "flavor" of the XHTML standard, namely [XHTML 1.0 Frameset](#). This standard is quite similar to the [HTML 4.01 Frameset](#) standard. Accordingly, whenever you use frames, follow the [frameset guidelines](#) established by the W3C group.

## Reference: Compatible Browsers and Platforms

The Duke-wide web standards have been developed and tested for a range of browsers and platforms that reflect the vast majority of entry points for our audiences. Accordingly, Duke-based developers writing to these standards should test browser behavior in as many of the following platforms and browser versions as possible. While not every browser listed here will render your site perfectly, they should display your site adequately (particularly Netscape 4.7x, which should minimally display the content of your site in a meaningful way).

To reinforce a guiding design principle for web developers: **the easiest way to make your site browser-compliant and browser-friendly is to write your code compliant to the web standards you employ**, from (X)HTML to CSS to JavaScript. You will always need to test for proper degradation once you have written to the standards, but this practice will also ensure future browser versions will be able to render your site.

### Platforms:

*Windows:* 2000, XP and higher

*Apple:* OS X (10) and higher

*-Nix platforms:* Sun Solaris 9 build, Linux@DUKE build (current version)

### Browsers:

*Internet Explorer:* 5.5 and higher (Windows)

*Netscape:* Communicator 4.74 (basic functionality only), Navigator 7.x and higher

*Opera:* 7.x and higher

*Mozilla:* 1.x and higher

*Safari:* 1.x and higher

*Konqueror:* 3.x and higher

The platforms and browsers listed here are subject to change based on web usage data.

### **Browser Compatibility Highlights:**

- Write code compliant to the standards you employ, and browser compliance will follow more easily

## Reference: Duke's Web-Services Offices

Most major divisions, schools and departments within Duke have their own staff and offices dedicated to providing some form of web-development services and/or standards. Consequently, the standards outlined here represent only a starting point for websites developed within one of these organizations at Duke. Please contact your appropriate office for more information on how your group augments or replaces the web standards outlined in this "Production Standards and Best Practices" guide.

*Office of Information Technology:* [Office of Web Services](#)

*School of Arts and Sciences:* [Cynthia Sulzberger Interactive Learning Lab](#)

*Duke University Libraries:* [Department of Web Services](#) (IT Services)

*Fuqua School of Business:* [Applications/Web Development team](#)

*School of Law:* [Web Services](#) (Educational Technologies)

*Pratt School of Engineering:* [IT Services](#)

*Nicholas School of the Environment:* [Information Technology Department](#)