Website Interface HTML Development Workflow

How to Streamline Web Interface Design for Developers and Designers

Pierre Berryer
Web Designer

Original Date: August 2000
Version 1.02

Website Development Workflow

Topics

Workflow Improvements

Business Team to Developer Team Transitioning

In regards to interface design and development, the focus is to facilitate the simultaneous and synchronized development of the visual design and backend coding by helping both sides understand and meet the needs of the other. This would be accomplished via the implementation of a set of basic standards and suggestions that are easily understood and used by all technical web developers. This includes HTML/Layout creators using Dreamweaver, JSP and ASP programmers.

One critical overlooked aspect of the AMS eCycle is the concept of Inter-Team Transitioning for web interfaces. The role entails Inter-Team communication management, with an emphasis on utilizing best practices dealing in transitioning deliverables from business teams to development teams. In practice, they should help justify and implement the increased quality and ease of implementation of a professional interface.

Working with the development team in helping them to convert HTML to JSP or ASP, though vital to the success of the project, its inherent potential for unexpected intricacy

is often overlooked in project plans. JSP and ASP developers are often unfamiliar with HTML and need an expert resource to ensure seamless conversion. Also, if the team is unfamiliar with the level of effort associated with implementing an interface paradigm they may prepared to abandon the idea of implementing a compelling design in fear of out of scope development time.

Skills beneficial to this role include a good understanding of the project cycle and project management concerns. An understanding of the limitations of web interfaces and the ability to communicate these effectively is a must. Also, a good understanding of more detailed developer concerns during the project cycle in relation to working with source material provided by a project's Web interface design team.

Dreamweaver: Visual HTML Editor

This centers around increasing the usage of Dreamweaver's higher level functions.

Note: When incorporating Library Objects or Templates in a site while using the remote site functions and Check In/Out capability, all files that will be updated must be checked out before the changes can be automatically made to them. Otherwise, the files will be seen as read only and no changes will be made to them.

Note: The HTML/Web Designer should make use of an Editing Guide that lists all the information pertaining to editing the site in the future, both by code editors and Dreamweaver.

Library Objects

The HTML Pages will be broken down into parts that can be more easily modified by the Teams who are working on them as well as allow changes in one file to be effective across all files. This reduces the time spent by development teams in searching through an entire page's HMTL just to edit a single database provided field. For instance, whenever a dynamic website needs to show the user's name, the design team responsible for the "look and feel" of the site's pages can insert a library object in place of the developer code that would normally call the viewer's name from within the database. Because the designers do not know the code that would be used to access the database, they can use the library file as a placeholder while continuing to edit the pages. Once the library file code has been modified to dynamically pull the info from the database, neither the developer nor the designer needs to go back to each page and update the HTML. Dreamweaver will automatically update every instance in which the library file was used in a web page.

The library object can then be edited by the developers. Rather than sifting through the entire page's HTML contents looking for where the name is entered, they would be sifting through several lines of HTML and replacing portions with the database code.

Regardless of whether or not the developers or designers are proficient in coding HTML, smaller chunks of code are much easier and faster to edit without introducing errors.

Incorporate the use of Library objects:

- Library files will allow the simultaneous creation of the design and development of the site.
- Library objects are chunks of code that are stored externally in files. Dreamweaver tracks each library file to the HTML files it is used in and when the library file is changed, the effect is that all pages using it are changed as well.
- If library files are named well, developers should be able to know what database content needs to be provided in each one. The library files can also be commented in detail using the standard
  <!-- HMTL Comment tags > style of commenting.
- The use of library files requires a decision by the design team to store the elements which will be displayed dynamically into a library object. This decision should take place during the development phase of a web project, whether it is just a presentation or otherwise. This dynamic storage of site elements allows the designers to more easily migrate a "presentation only" site into a functional one.
- Developers may still need to deal with chunks of HTML. The difference is that they will be viewing and editing chunks rather than entire pages. This represents a significant time savings during editing and debugging of a project site.
- Many elements can be stored in a library file. For example, the designers can store a form's text input box in a library file, comment its usage and role in the site and continue to design the page. The developers would then only need to modify the specifics of that box such as its name in order to make the page's form submit the information correctly to the server.
- Calls to library files can even be nested within another library file. For example, this would allow the designers to craft a library file as a side navigational tool. Should the developers need to add dynamic entries to this bar, those sub-portions of it could be stored in a library file that would be edited by them in smaller chunks.

Library File Storage and Tracking:

The library objects should be stored in a central area accessible to all members involved in the project. This can be facilitated by allocating a central project workspace in which the designers can place the contents of a Dreamweaver site.

This allows a member of the project to oversee the library file creation scheme and act as an intermediary between the design and development teams. This person should track the library files, making sure that the development team is aware of their use and

making sure that project members are kept current. This person should also be responsible for making sure that the developers understand what content needs to be in each library file. While it is the designers responsibility to create well labeled library files, it will also be up to the designers to ensure that all required user data is presented. Hence, they will generate library files as the need dictates.

Maintaining readability/graphical display on non-Dreamweaver systems.

1. The contents of a library file are stored in each HTML file in which that library file is used. The contents are wrapped in comment tags which DW uses to determine the location of the library object.
2. The library file, when seen from within the DW editing window, cannot be edited from within the document it is used in. The library file must be edited directly and the changes automatically update across the site. This prevents accidental modification of the library file elements used within a page.

For example a library file would be shown within the HTML document in which it is used via the following convention:

```
<BODY>
 <HTML
  <TITLE>
   Library Object Test Page
  </TITLE>
  <P>
   This page will display a library file in which an input box
is stored.
  </P>
  <FORM METHOD="POST" ACTION="SENDTOSERVERPROGRAM">

  <!-- #BEGINLIBRARYITEM "/LIBRARY/LIBRARY FILE NAME.LBI" -->
     <INPUT TYPE="TEXT" NAME="LASTNAME">
  <!-- #ENDLIBRARYITEM -->

  <INPUT TYPE="SUBMIT" NAME="SUBMIT" VALUE="SUBMIT">
  </FORM>
 </HTML>
</BODY>
```

Technical Limitations of Library files within DW

- Library files cannot be used to replace elements within an HTML tag. When used, they must contain opening and closing tags or any plain text. You could not, for instance, use a library file to replace only the contents of the “name=” field in an <input type="text" name="??????"> tag.

- This does not circumvent the need to enter database data names by hand. The developers must still adhere to the naming conventions used for the database software being used. While they still need to use precise capitalization in the field names, the chance for error is reduced. If any dynamic information does not display properly, generally only one file will be to blame and only one edit will need to be made.
- In the future, custom programming may allow for an environment in which standardized and properly commented Library files could be modified dynamically using the exact information provided by the database.

Dreamweaver/HTML Integration with JSP

JSP can use server side includes formatted to appear similar to ASP tags when seen from within Dreamweaver. For instance, a template page may contain JSP includes placed to call in the appropriate navigation elements, and content for the page. This system doesn't perfectly mesh with DW's typical implementation of Library objects. The problem is that a typical Library object contains the exact HTML code to display that element. One solution is to use nested Library objects. The problem is that this can confuse the developers looking at the code in the HTML pages and the assortment of Library files unless they fully understand what is happening beforehand.

Directory Structure when using JSP includes:
A common directory structure can make using Library Objects as JSP includes much easier. The files called by JSP includes are pretty much exactly what Dreamweaver creates when you convert a section into a Library Object. These snippets of code can be seamlessly pasted into a document. When developers ask for JSP includes to be created, the Library Objects can be directly copied from the site's Library directory with no modifications if the developers have chosen a directory structure which meshes with Dreamweaver's. For instance, if the Library objects reference an image in a parallel level directory, their links will be shown as "../GraphicsDirectory/filename.gif" or something similar. If the developers mimic this directory structure by placing the JSP includes in a directory on the same level as that of the images, the newly copied Library Objects will continue to display properly without having to modify all their links.

Try to create a directory structure that can be shared with both types of files, JSP and HTML. For instance, they can use the same graphics directory and CSS directory and files. This way, changes by the designer are seen instantly for the developers as well. There should be a separation of the actual HTML and JSP files, however. One idea is to have an HTML directory with the identical subdirectory structure as the JSP directory. This allows the designers to work and experiment without potentially damaging the developers' work.

Form Submission via JavaScript: Avoiding onClick
This is a constant factor in developing JSP based interfaces. Javascript is used to validate the form before it is submitted to the server. This approach can lead to

difficulties in browser compatibility, both between IE and Netscape as well as between versions of the same browser, when done improperly. onClick is also unavailable for use in ADA compliant sites. If the user is using a non mouse based browser, the onClick event may not work.

One of the best ways to deal with this is to use an <A href="…"> tag where the … is set to the JavaScript to run your function and submit your form. This method eliminates reliance on the onClick event which has been known to cause browser compatibility issues, as well as being detrimental to ADA efforts. Non-mouse-based browser systems cannot always trigger an onClick event. The additional benefit to using this method to submit a form is that the Netscape browser will now change the mouse cursor to the traditional pointing finger indicating the user can click on this element.

For example, the following would create a function that will check to see if the form has been validated then submit the data:

```
<SCRIPT>
     function submitOK() {
          if (CustomValidateFormFunction()) {
               document.yourFormName.submit();
          }
     }
</SCRIPT>
...
<a href="javascript: submitOK();">OK</a>

or

<a href="javascript: submitOK();"><IMG SRC=”image.gif”></a>
```

For an image submit, this method would force the browsers to display the "Click" finger icon over the image as opposed to Netscape's default arrow over standard image submits. Variations on this can be done by actually skipping the A tag and embedding the HREF="…" directly in the IMG tag. This is not recommended, however.

Cascading Style Sheets (CSS)

Cascading Style Sheets - Explanation

> This specification defines Cascading Style Sheets, level 2 (CSS2). CSS2 is a style sheet language that allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of

> documents from the content of documents, CSS2 simplifies Web authoring and site maintenance.
>
> CSS2 builds on CSS1 (see [CSS1]) and, with very few exceptions, all valid CSS1 style sheets are valid CSS2 style sheets. CSS2 supports media-specific style sheets so that authors may tailor the presentation of their documents to visual browsers, aural devices, printers, Braille devices, handheld devices, etc. This specification also supports content positioning, downloadable fonts, table layout, features for internationalization, automatic counters and numbering, and some properties related to user interface.
>
> *Excerpt from: Cascading Style Sheets, level 2 CSS2 Specification Guide*

Benefits

CSS centralizes the storage of formatting information for a web site. The formatting codes (styles) are stored in one file that is linked to each of the web site's pages. The styles are defined and applied to a tag (i.e. a header tag such as <H1>), a Class, or an ID name. To use the formatting information, elements merely need to use the tag, apply a class to a tag or name an element's ID. When implemented well, it becomes easy to alter the entire site's look from within one document. Because the formatting is independent of the web pages, be they HTML, JHTML, XML, or DHTML, it can be controlled and modified by the designer while the content is still being worked on.

Implementation of CSS is facilitated through the use of graphical web editors such as Macromedia Dreamweaver. This software provides excellent support for the editing and display of CSS. The editor can also be configured to format HTML code as desired to maintain human reading and editing ability. Yet, once the user is aware of its use, CSS coding remains easy to modify by hand. Because CSS is straightforward to understand and implement, it can coexist with static pages as well as dynamically created pages. Also, through the creative use of templates and library objects from within Dreamweaver, a web site's pages can be broken up into individual pieces which can be integrated and used to update the entire site in one pass.

Attributes

CSS allows the layers and positioning information to be stored in a style sheet, rather than inline in the HTML. This allows absolute positioning, changeable afterwards via one document. This also allows for the HTML code to be cleaner because the detailed positioning information for sections, headers, images, text, etc. can be entered externally and kept free of accidental positioning changes in the editing of individual pages.

Style sheets can affect the font and position of text in ways previously impossible in straight HTML. From a designer's perspective, this enables much more precise control over the "look" of a site during all stages of development. The positional layout control

enables interface designers to alter the placement of repeated elements such as content areas, headers, and buttons without needing to modify pages individually.

The CSS instruction set can be used in part or completely, with varying degrees of success dependant on the browser being used. Non CSS browsers still show pages correctly if they've been designed to degrade gracefully. This can be a big benefit for sites adhering to the Americans with Disabilities Act (ADA) standard for web site design. When the browser does not support CSS, the pages can still display as traditionally appearing HTML pages. If properly designed, they can degrade to work in older browsers very well and maintain the logical machine readability that many handicapped individuals require. If the site uses styles to look great when seen, yet is still able to be browsed and accurately represented by a non visual web device, the design succeeds. By applying styles to standard tags, a designer can show section headers (<H1> etc.) in any font variation, yet, because they are web standard headers, a text to speech web browser would still understand their role in the organization of the page's contents.

CSS can also integrate with HTML or XML and Javascript to form DHTML, or Dynamic HTML. Because the CSS attributes remain modifiable even after the page loads, its design can be dynamically modified in real-time. Elements such as the look of the fonts, color of the page, or location of the layers can all be modified in response to user interaction, given a supporting browser. This allows simple usability effects such as text links that change their appearance when you roll over them, yet are still in accordance with the ADA. Layers can also be animated and triggered via user input. Macromedia Dreamweaver facilitates the implementation of the underlying code.

An interesting aspect of CSS to keep in mind is that text can be forced to appear at a certain point size, regardless of the browser's font size settings. This can be ideal to ensure that the site is seen as intended during presentations or technology demos and that text fits as planned within layers. The unfortunate side effect of this is that the user is limited to viewing the site at a font size that may be too small. Issues like this should be kept in mind when designing sites that look good, but maintain effective usability. Specifying text in variable units like x-small, etc. let's the user adjust the text size.

Cons

Like any newer Internet technology, CSS is not without its problems. The primary concern is that it is not yet fully supported in Internet Explorer 5 (IE) or Netscape Communicator 4.7 (NC). While they both tend to display text formatted relatively accurately, NC does not adequately support layers. It exhibits one bug in particular, in which, after the page has loaded, the layers narrow to the width of the smallest word, regardless of the size specified in the HTML document or the style sheet. The current workaround to this is to place a single cell table with the correct size inside the layer and to nest the layer's contents in that table or a transparent single pixel image sized to the width you want. This forces Netscape to widen the layer appropriately.

CSS also tends to display differently in different browsers, whether using different software programs or different operating systems. And as CSS cannot currently be disabled in all browsers, the users are forced to view it in one way, which may not be the way the designer's browser shows the site. This, however, is a problem inherent in all aspects of HTML design, regardless of CSS.

When text size is specified in pixels, points, picas, etc., it cannot be changed from within the browser. The solution to this is to use variable text sizes such as small, x-small, large, ems, etc. These will resize depending on the base font size of your browser.

CSS Commenting

Remember to enter Comments within a style sheet. This can't be done from within Dreamweaver and must be done through a text editor. Be careful when editing your CSS file externally from DW as you may accidentally overwrite your changes if you edit the file while DW is still open.

CSS Comments are structured as follows: ***/*** comment contents *****/***

Links

Webmonkey | StyleSheets: http://hotwired.lycos.com/webmonkey/authoring/stylesheets/
Webmonkey | **Mulder's Stylesheets Tutorial**
http://hotwired.lycos.com/webmonkey/authoring/stylesheets/tutorials/tutorial1.html
Stylesheets let you control layout like never before, right down to the spacing between letters. Mulder shows you how it's all done. *16 Sep 1999*

Cascading Style Sheets, level 2, W3C Official Guide
CSS2 Specification: http://www.w3.org/TR/REC-CSS2/

Cascading Style Sheets, level 1 (CSS1) [Recommendation - 17 December 1996, revised 11 January 1999]
http://www.w3.org/TR/REC-CSS1

Cascading Style Sheets - Workarounds

Anchor Link Style Control

CSS can control link colors during four states: Link, Highlighted, Visited, Active. This can be a problem when using multiple classes to control link colors in different parts of a page. If the primary A:LINK and A:VISITED attributes are set, it may override classes applied to specific <A> tags. This results in inconsistent color control.

The best solution is to avoid the CSS selector states (link, visited, hover, active) for the primary <A> tag and instead, create selectors for specific occurrences of the <A> tag

and a class only. For instance, A.ClassName:hover results in hover parameters being set for a class only whenever it is used in an <A> tag.

If the link and visited colors are set the same, they should be removed and the basic <A> tag should be modified alone. This allows a class to be applied to it in the HTML, which will then override its default style.

If the link and visited link colors are to be different, this will cause some problems. If the pages are being generated and the head of each HTML page is controlled via one file, an easy solution would be to use the default link and vlink attributes to the BODY tag in each page.

To override bugs in assigning a class to the <A> tag, insert a <SPAN> between the <A>…</A> link and then apply a class to the span. This will not always change the underline color from the default A:LINK or A:VISITED settings.

To actually get A:LINK and A:VISITED selectors to behave in a site that uses <A> tags with the possibility of multiple classes being applied, the solution is more complex. Every class that will be applied to a an <A> tag must have the corresponding selector state versions as well. This means a link and visited version of each class in the form of A.ClassName:selector

Classname Bug
When naming your classes, avoid using the underscore character. Netscape 4.61 and below do not support it and will ignore these classes.

Cascading Style Sheets - Theoretical

Details
CSS allows percentage based as well as absolute values for positioning and sizing of elements within a style sheet.
CSS will (someday) allow the user to create their own stylesheet to process pages they're viewing
Allows specific control over web page margins.

Advanced CSS usage allows Image filtering commands (in IE 4+).

- Alpha channel: Allows opacity even on a gradient
- Motion blur
- Transparent color on JPEGs (chroma)
- Drop Shadows
- Image Flipping
- Glows (text and cutout images)
- Grayscaling
- Color Invert (negative)
- X-Ray (like Invert but black and white)
- Masking

Wave (Can it be changed dynamically via javascript?)

http://hotwired.lycos.com/webmonkey/98/15/index4a_page7.html?tw=authoring

Trick No. 1: Use Styles on Similar HTML Tags
Ie, use <B> to house your Bold style sheet info and, <I> for italicized text.

Hypothetical solution for Layer Based Site developer interaction problems:
Goal is to integrate an externally referenced: side Navigation bar, Top nav bar, and content area. The content area then integrates text, image headers, and image buttons.

Problem: With absolute positioning of layers, programmers and designers constantly butt heads as the three elements undergo positional changes throughout the revision process. Margins change size, buttons are offset, and the coders are left to sift through machine generated code (sloppy Dreamweaver inline layer code).

Answer: Store the positioning code in a style sheet. Put a (<DIV CLASS = “Category”>) container around each of the three primary elements and give each a different (class) name. For the headers and buttons, give them each a category (i.e. CLASS = “headers” and CLASS = “buttons”).

The style sheet contains the positioning coordinates for the elements, referenced by their category (Class) names.

Remember that the layers can be created in any order as well. This results in code that is not necessarily laid out from top to bottom to match the output of the page. Because each layer is absolutely positioned, you can put elements like the content area, which would be dealt with by the developers, at the top of the HTML document. The code to generate the interface could be placed lower in the file, lessening the excess code developers need to sift through.

Requirements:
Good communication between the development teams. They have to know the styles available in the style sheet and avoid entering absolute inline values within the document being worked on. This means being careful not to let Dreamweaver insert them while working on the page.

Inline positions are those coordinates that are entered directly into the <DIV> tag that creates each layer. The alternative, as discussed is to have the <DIV> tag contain a class = "???" In which the corresponding class name contains the positional coordinates in the external CSS file.

There must be someone who regulates the style sheet, informing all project members of newly entered styles and ensuring that all developers are using the most current style sheet when test displaying the project. Because the style sheet is referenced externally on page load by the browser, as long as the developers use styles (classes) that are listed in the style sheet, they can focus on the code for the section they are working on even before the look and feel is approved. The interface designer can then focus on the visual aspect of the site and integration of the different elements. Bear in mind that a list of styles should be kept and updated, even if they are not yet defined. This is because the developers will need to create style names as they build their dynamic pages.

Dealing with Netscape Problems:

Problems with Netscape are a time consuming issue when trying to achieve multiple browser compatibility or Browser Independence. The following are some notes to help with this issue. They deal with Netscape 4.x as version 6 uses a different Document Object Model (DOM).

JavaScript Debugger: If you are experiencing JavaScript problems, Netscape provides a basic console that will display the errors and the lines they occur on. It can be accessed by typing javascript: as the URL.

Classname Bug:
When naming your classes, avoid using the underscore character. Netscape 4.61 and below do not support it and will ignore these classes.

Any class using an attribute that is incompatible with Netscape will result in that entire class being ignored. To set a class to use Internet Explorer only attributes, break up the class into two parts. The first, consisting of the class name only, should contain the standard CSS attributes as supported by both browsers. A second tag specific class could then be listed consisting of the IE only attributes. The following example would create a class to be used on images that would have a special effect for IE:

```
.YourClass {standard attributes}
IMG.YourClass {IE only attribute}
```

Invisible Form Fields:
If the form elements display in Internet Explorer and not in Netscape, try this first. Netscape requires form elements to be placed between <FORM> … </FORM> tags. Also, make sure that intersecting tags do not break the form. For example do not open a form, then open a table, then close the form then close the table. The following is a simple example:

```
<FORM>
     <TABLE>
          <TR>
               <TD>
                    <INPUT>
               </TD>
          </TR>
     </TABLE>
</FORM>
```

The following is an example of incorrect <FORM> tag insertion:

```
<TABLE>
```

```
      <TR>
            <TD>
                  <FORM>
                        <INPUT>
                  </TD>
            </TR>
      </TABLE>
</FORM>
```

Table colspans and rowspans: Netscape sometimes has trouble dealing with complex tables that contain merged cells. The safest bet is to use tables that have a unified number of columns/rows. If you need a table with a one column, one row cell on one side and a multi row cell on the other, you may be better off putting a nested, multi-cell table in one cell of a simple table. This is another of Netscape's erratic problems that may or may not affect your page.

Table Backgrounds: Netscape does not properly support images in table backgrounds. The results are erratic. One possible quick fix is to apply your image to the table and to avoid having it repeat in each cell is to have each cell's <TD> have the attribute <TD background=""> and this may work. You still may have the problem of a nested table adopting the background settings of the parent table or even the page background.

CSS in Tables: Netscape will often not display the font attributes when a class is applied to a table, cell or row. If you are **creating tables**, try to enclose the text within the <TD> cells in a <P></P> and apply a style to the <P> tag in the CSS. This will ensure that the text will look at least in sync with the rest of the site should a user run into Netscape's random problems formatting cell text via a class applied to the table elements. Try to avooid applying a class directly to the TD or defining the TD tag in the CSS as this makes it difficult ot modify that's cell's attributes without adding a class to the cell. For instance, if you define the <TD> tag to have any text aligning attributes, you cannot override those attributes without applying a class that counteracts the original settings. The align=??? Settings cease to function.

For the **table background colors**, try to use a system where the background colors are pulled from a style while the text inside the table is handled via the <P> tag. Exceptions to this are when the table contains lots of data and needs a smaller font applied to the <P>.

You can **apply a style to the table** in order to color the whole thing in one shot. Then apply a style to the title cells as needed or any nested tables within cells. Netscape can handle this. Do not rely on the <TD> type applied class to give their font information to Netscape. For that, you must apply a style to the contents of the cells (like the <P> tag that holds the contents).

Pages not loading (and displaying a different URL):
The sign of this happening is if the page fails to load in Netscape and displays an error message while redirecting the browser's URL to a different filename than the one entered. The most likely cause of this problem is that a file called by the page is not found. For instance, if the page calls an external JavaScript file and it is not found, Netscape will return an error and fail to show the page. Internet Explorer, however, will continue to display the page. This problem also can occur with linked stylesheets that are missing.