SYBEX Bonus Chapter

Developing Killer Web Apps with Dreamweaver MX® and C#™

Chuck White

Bonus Chapter 1: Developing a Workflow

Copyright © 2004 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

ISBN: 0-7821-4254-0

SYBEX and the SYBEX logo are either registered trademarks or trademarks of SYBEX Inc. in the USA and other countries.

TRADEMARKS: Sybex has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer. Copyrights and trademarks of all products and services listed or described herein are property of their respective owners and companies. All rules and laws pertaining to said copyrights and trademarks are inferred.

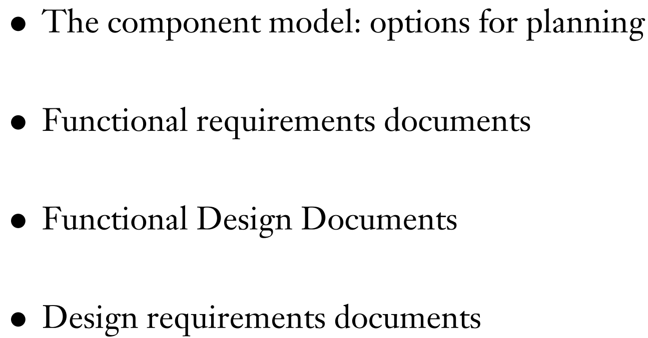
This document may contain images, text, trademarks, logos, and/or other material owned by third parties. All rights reserved. Such material may not be copied, distributed, transmitted, or stored without the express, prior, written consent of the owner.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturers. The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Sybex Inc.
1151 Marina Village Parkway
Alameda, CA 94501
U.S.A.
Phone: 510-523-8233
www.sybex.com

Bonus Chapter 1

Developing a Workflow



- The component model: options for planning
- Functional requirements documents
- Functional Design Documents
- Design requirements documents

Developing a web application is a process that requires planning, even when you're just a one-person operation. This chapter takes a team-oriented approach to the challenges faced by web application builders. Even if you're a one-person show, however, you should *act* as though you're a team and build your workflow accordingly. Your overall workflow should consist of four general areas:

- Your team's priorities
- Specifications for the components of your application
- Team management and direction
- A matrix indicating your project's progress

In addition, most large software and web development houses use a *functional requirements document* and a *design document* for managing workflow specifics. This chapter explains how even a single-person development team should use this planning technique. It also tells you how you can do this using Dreamweaver MX, which is especially important when working with .NET.

The Component Model: Options for Planning

.NET is a component-based architecture, so as you build documents in Dreamweaver MX, it's a good idea to think about planning the layout of these components. This may sound a bit like software engineering, and in fact, building web applications with Dreamweaver MX and .NET has more similarities to software engineering than to simple website design. If you're not comfortable with that notion and don't consider yourself a programmer, don't be intimidated.

Component-based architecture such as .NET is based on what software developers refer to as *abstraction*. This is really just a fancy word for hiding the intricacies of how things work and stuffing everything into little objects that you can play with. Another word for this concept is *encapsulation*. Software engineers love jargon, and you'll run into these two words frequently in discussions based on object-oriented programming (OOP). Like it or not, you'll become engaged in object-oriented programming once you drop your first Dreamweaver .NET object onto a web page using Dreamweaver's visual interface. Since the intricacies of how these objects work—not to mention their code—are hidden from you, all you need to worry about are the *definitions* of the objects. And that's why nonprogrammers needn't be intimidated by the fact that they've suddenly become software engineers when they're building web applications with Dreamweaver MX.

And although Dreamweaver MX helps you pull it all together quickly, you'll want to develop an overview of how all these components are put together. In other words, you'll want to develop a workflow. To do that, you'll need to begin the process of understanding what various components do.

.NET Controls and Components

As you're building web applications, many types of documents will cross your desk. You could conceivably have a .NET document that looks as simple as the one following, yet renders a substantial amount of data when delivering the final product in HTML format to the user's browser:

```
<%@ Page Language="C#" ContentType="text/html" ResponseEncoding="iso-8859-1" %>
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<asp:xml DocumentSource="orders.xml" ID="ledger" runat="server"
  TransformSource="orders.xsl" />
</body>
</html>
```

The component-based architecture of .NET means you can drop *components*, most often called *controls* in .NET lingo, into a .NET page. Each of these controls does a significant amount of work. Sometimes, a .NET control is even a mini-application. In Figure 1.1, you'll see a window that seems to bear little relation to the number of lines of corresponding code (the bold lines in the preceding snippet). However, the web control that is used in that code refers to an XML document containing data, and an XSLT (Extensible Stylesheet Language Transformation) that transforms that XML into HTML for display in a browser. Of course, the XML and XSLT involved in these arrangements may themselves be quite complex.

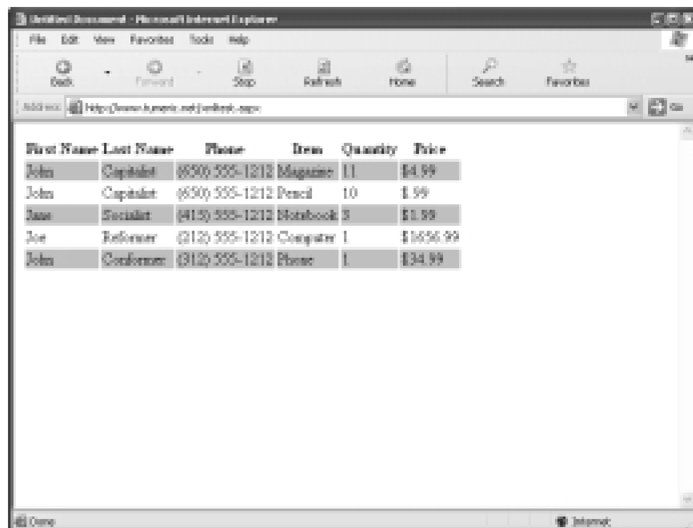


FIGURE 1.1: Very little code was needed to display this data.

There are three basic kinds of components in the .NET architecture:

HTML server controls These are the same as type as the HTML elements you're already familiar with, but server controls possess one significant difference: They are decorated with a special `runat="server"` attribute value pair that, when used in combination with an `ID` attribute, indicates to the web server that the element should be handled by the server and can then be controlled programmatically. Without the `runat="server"` attribute value pair, the element is assumed to be a client-side HTML element, just like the kind you've always used. For example, consider this HTML input element:

```
<input type="hidden" name="x" value="hello" runat = "server">
```

The element is exactly the same as an HTML element, except for the `runat="server"` attribute value pair. This tells ASP.NET that the `input` element will be evaluated by the ASP.NET engine.

Web server controls These are special controls developed specifically for .NET. Like HTML server controls, web server controls are handled server-side and also need the `runat="server"` attribute value pair in order to be successfully interpreted. In the preceding code accessing an XML document, the `asp:xml` element is a web server control that processes XML documents. Extremely robust and rich controls like the calendar control and the datagrid control provide an amazing amount of functionality with a minimum amount of effort (as compared to the old days, at least).

Custom controls These are controls you can build yourself using the language of your choice (C#, J#, JavaScript, or VB.NET). In the old days of ASP, you had to register a DLL on the server in order to use custom controls (which was a pain if you were working with remote hosting providers, because you had to ask them to do it for you). Now you can compile a DLL on your own machine and simply upload it to a `bin` directory, and .NET will find it and run it. Calling it is easier, too. No longer do you have to use the `CreateObject()` method within some VBScript code on an ASP page. Instead, you can drop the component in using element tags. The control is still registered, but it's registered on an ASP.NET page, not through the operating system.

I'll be exploring all of these controls in great detail beginning with Chapter 4.

Other Building Blocks

In addition to the new components found in the .NET framework, there are many other ways to develop a component-based architecture.

- You can link CSSs (Cascading StyleSheets) to your web pages so that a change in one stylesheet results in changes to, literally, if you wish, every page on your website. (You should strive to avoid HTML presentation tags such as `FONT`, because they're being

phased out of the language completely, and CSS provides a better way to control the look of your web page.)

- You can develop XSLT documents that allow you to easily change the rendering of data or content, customizing it to specific needs.
- You can store JavaScript in script files and provide what scripting professionals call *an abstraction layer*. This consists of objects defined through functions in the JavaScript file, called by the web page that needs them.
- You can use server-side includes to build headers and footers, a huge time-saver when you're working with anything more than a few web pages.

Even the Macromedia `DreamweaverCtrls.dll` component consists of several public classes, which are essentially objects with which you can manipulate databases. You'll never see these public classes, and you don't need to know anything about their underlying code structure, but they're there. The functions that make them work are hidden from view and compiled into the `DreamweaverCtrls.dll`, which is called as a component. And because they are "public" classes, the objects are callable and manipulated through simple elements, consisting of attributes and child elements, such as the `MM:Insert` element.

NOTE The functionality of the `DreamweaverCtrls.dll` is not completely hidden. You can actually view and even alter the source code (but, of course, not before you save it with a different name and compile it as something other than `DreamweaverCtrls.dll`, to avoid trashing the original functionality). The source code is in the following directory (the DreamweaverMX directory is usually in a directory named Macromedia that resides wherever you install programs): `Dreamweaver MX\Configuration\ServerBehaviors\Shared\ASP.Net\Scripts\Source`.

The payoff to planning ahead and thinking about how your components all work together is large, because when you use a component-based architecture everything is much easier to maintain. Working this way allows a great deal of scalability, which means that you can expand the capabilities of your system very easily.

TIP You can also use Dreamweaver Templates, which are predesigned, locked documents with editable regions, to further focus your efforts on planning ahead and maintaining design consistency in your site.

Separating the Data from Its Presentation

Thinking beyond components, even at the data level there is a way to create componentized architecture. A perfect example is through the use of *stored procedures* in a database. A stored

procedure holds cached data-manipulation procedures in a database. These procedures might select data, insert data, update data, or delete data, but since they're written at the database level and stored there, rather than within the web application itself, they're much easier to maintain. This is because, like all good components, they have a single point of update immediately available to all areas of an application. The stored procedures are also much faster than data manipulation tasks done at the web application level. Luckily, Dreamweaver provides support for working with stored procedures.

An example of a stored procedure is a simple SQL `SELECT` statement. SQL, of course, is the language databases use to manipulate and define data. At its simplest, SQL is very easy to learn. For example, this statement will select all the data from a table named Clients:

```
SELECT * from Clients
```

This could be an awful lot of data depending on the circumstances, and SQL lets you filter the data you want, but you get the point. (In other words, you'll want to avoid `SELECT *` in production environments because you will bring in data you don't need.)

When you use Dreamweaver MX, Dreamweaver automatically builds these kinds of SQL statements into the web page you are developing as part of your web application. There's nothing inherently wrong with this, but as you get more comfortable building advanced applications you may want more efficiency, and that's where stored procedures come in. Rather than build these kinds of SQL statements in the web application, you can build and "store" them in the database. The result is remarkable: Instead of pages that take their time loading and seem to hang, the pages you build with stored procedures will usually snap right up in front of your users, because the execution plan on the database server is very efficient when you use stored procedures.

I want you to get comfortable early with the notion of stored procedures. They are your friends. And using them is part of good component-based object design. They are also easier to manage from a security standpoint because you can assign specific rights to a stored procedure on a user or group basis. This is good for protecting the actual tables in the database; when you work with stored procedures you don't need to provide access to the tables themselves, just the stored procedures that are acting on the tables.

To sum up, when building .NET web applications with Dreamweaver MX, think components. Then, think about them again.

Functional Requirements Documents

Are you sold yet on the notion of thinking in terms of components? If not, read on, because it may make more sense when you are doing some actual planning.

Now I'm going to talk about something none of us like to do, because there's simply no fun in it, but *planning* is something that's absolutely necessary. Even if you have to resort to paper and pen, it's critical that you plan how you're going to make these components all work together. Most everyone's instinct is to skip this step, and just fire up Dreamweaver MX and start building web pages. But if you're not building a personal site for yourself—in other words, if you're being paid for your work—you're really burning away someone else's money if you don't plan ahead.

Remember that you're not building a simple website; you're engineering a web application. Dreamweaver MX makes it all pretty easy, but you're still building a complex piece of software whose successful operation relies on your anticipation of a wide range of possible user behaviors. You do, of course, have the option of changing your site when you've anticipated wrongly; even the best planning will not catch every possibility. Nevertheless, many possible actions can be predicted by simply diagramming them into a flowchart or some other schematic tool.

In this section, we will discuss the kind of things go into functional requirements documents. Keep in mind that web application development companies will each have their own way of doing things. What you encounter may be quite different from this, and if you're developing your own, you will likely find you want to do some things differently. These are just rough guidelines.

Workflow Fundamentals

The first step in building a functional requirements document for your web application is to think in the most basic terms by asking yourself a few questions:

- What are your site's objectives?
- Who's your audience?
- How will you speak to your audience?
- How will you structure your site?
- What will your site look like from a design standpoint?
- Who will handle site maintenance, and how?
- What is the plan for managing security policy?

Site Aims and Objectives

The most obvious question to ask when developing a website is, what is it for? What are the objectives the site is trying to achieve? The answer may seem obvious until you begin to examine the scope of the project closely. You may discover that, if you're selling toys, for

example, you need to go beyond the simple fact that you want to use the site to drive sales. You may want to use your site for driving the PR process, disseminating news, or building customer service applications.

Here's a brief list of goal-related questions to get you started:

- What is the site's purpose?
- What are the short-term and long-term goals for the site?
- Who is your target audience; if there is more than one, what is their order of importance?
- What do your competitors' sites accomplish and how do they work? What data are they displaying? What kind of navigation do they use?
- What kind of standardization and/or accessibility issues do you need to address? How will you accommodate Section 508 and disability issues? (Section 508 of the federal Rehabilitation Act is discussed in a later section.)

I'm sure you'll think of more questions as they pertain to your specific web application. The point is this: If you build an application without deciding on its objectives, the chances are good your site will reflect this lack of focus.

User Survey and Audience Definition

One of the most important aspects of developing your site is determining the target audience, because your content is going to need some kind of underlying theme and direction. If you're building a toy site, you want to know whose attention you want. Are you targeting children, in the hopes they'll pull on their parents' sleeves to get them to buy a hot new item? Or are you focusing on the parents themselves? Or both? Who else might visit the site?

You may have more than one audience, so you need to decide early about who is most important, because addressing more than one audience may have significant impact on your site's navigational elements. If your organization has the resources, you may even wish to consider a user survey on audience expectations for the site.

Content and Functionality

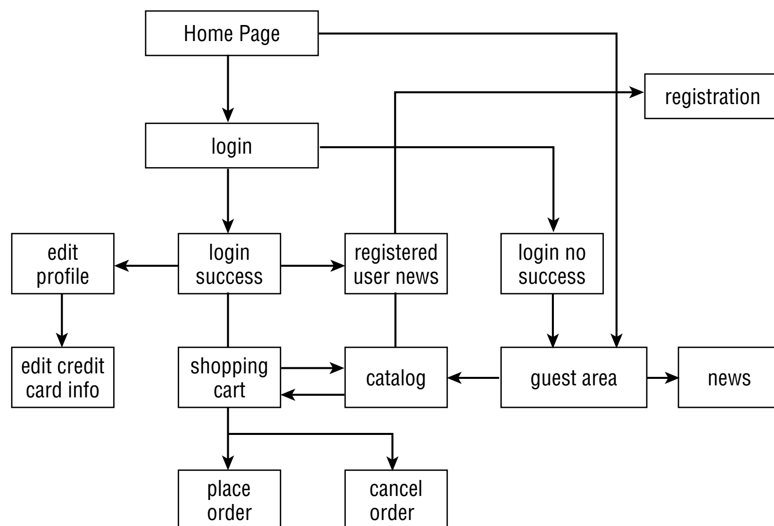
The next step is to think about how you're going to talk to your target audience. This is where you might want to develop a strong mission statement, and a brief overview of the audience demographics.

Then you need to decide on a "flavor" for the site. Obviously, content for an audience of children will be much different from content directed toward adults. A site geared toward youngsters may have games, puzzles, contests, and other youth-oriented activities. A more adult-oriented site may have discussion boards or forums.

Structure

At this point you're still in the formative stages of planning the site, but this is a good time to bring out the pencil and paper or the whiteboard and start mapping out a rough *schematic*. The schematic is a map of your site, similar to Figure 1.2, that shows a general structure. Don't go into much detail yet; just provide a nice overview, like the first stages of an outline, that focuses on the high-level elements of the site. You'll fill in the details when you build the Functional Design Document (FDD). All you're concerned with at this stage is the flow among the site's major sections.

FIGURE 1.2: It's not pretty, but anything is better than no flowchart at all.



In fact, you'll probably notice that Figure 1.2 isn't at all a model of professional perfection. But that's okay; it's just a sketch. As the functional diagramming process progresses within the scope of the Functional Design Document (discussed shortly), you'll improve the way it looks. If you're working on a very large, complex site, a sketch like this will give way to a professional looking flowchart or model diagram.

Design

Whether you design the site yourself or have a designer or design team working with you, you need to set up some parameters and design goals. Do you want a classic look (whatever that is)? Do you have any specific usability requirements the designer will need to consider?

What about Section 508 and disability issues? Many sites are subject to the Section 508 standards, and you should be looking into making your site compatible for impaired users whether it's mandated or not. You'll want to lay these out now, before work on the Functional Design Document is started.

Maintenance

Site maintenance is an area whose importance may seem so obvious that there should be no need to remind you to take it into consideration. How the maintenance is to be accomplished, however, is actually extremely crucial in the early going. This is because many websites, especially these days, are built to include a content management system (CMS). A CMS is an administrative section of your website that is available only to the individuals administering the site—probably your client (if the website is for you, I guess you can consider yourself a client, too).

Content management systems usually have a web-based interface that makes it possible for your client to update the site without any HTML knowledge. If your site includes such a system, you'll need to plan and develop it as a separate and distinct site apart from the one you're building for the general public. Not only that, but you should think early on about how you can make the CMS as generic as possible, so that you can reuse it without reinventing the wheel all the time.

Security Policies

I could write an entire book on security policies, except for one thing: I don't know too much about security. Security is such a fundamental issue that, if at all possible, you should bring in an expert to handle it for you. If your web application's budget can't afford to devote some extra resources to security, then I hope it's not *my* credit card you're processing! Even if you're on a tight budget, there are ways to establish substantial security policies because there are numerous third-party vendors who specialize in security implementations. A remote hosting service, for example, will frequently include a security component from a third-party source.

When building your security policies, you'll examine at least these four general issues focusing on the Who, What, When, and How of security:

- Who has access?
- What parts of the web application do they have access to, and what type of logs and monitoring system are you building?
- When do they get access?
- How is access controlled?

Section 508 and Disability Requirements

There are a large number of worldwide governmental regulations governing websites geared for people with disabilities. Generally, these guidelines are only enforced when a website interacts with a government entity in some way. It may be a contractor's website, or even a website built for the government in question. However, even if you're not mandated to build

accessible websites, you should—and it's easy to do if you plan ahead. Generally, designing for accessibility means paying attention to images and the like, and providing textual description alternatives so that, for example, a speech-based browser can read the web page to someone who is sight impaired.

In the United States, the main legislation concerning disabilities and web design is the 1988 amendment to the Rehabilitation Act, Section 508. Generally, you'll be in pretty good shape if you follow the guidelines in Table 1.1. For more information on other disability initiatives and legislation worldwide, visit the World Wide Web Consortium's website on Policies Relating to Web Accessibility at `http://www.w3.org/WAI/Policy/`.

TABLE 1.1: Achieving Section 508 (Disability and Accessibility) Requirements

Excerpt from Sec. 508 Standard	What You Need to Do to Comply
A text equivalent for every non-text element shall be provided (e.g., via "alt", "longdesc", or in element content).	1. Every image, Java applet, Flash file, video file, audio file, plug-in, etc., must have an `alt` description.
	2. Complex graphics (graphs, charts, etc.) must be accompanied by detailed text descriptions.
	3. The `alt` descriptions must succinctly describe the purpose of the objects, without being too verbose (for simple objects) or too vague (for complex objects).
	4. `alt` descriptions for images used as links must be descriptive of the link destination.
	5. Decorative graphics with no other function must have empty `alt` descriptions (`alt= ""`), but they can never have missing `alt` descriptions.
Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.	Multimedia files must have synchronized captions.
Web pages must be designed so that all information conveyed with color is also available without color, for example from context or markup.	1. If color is used to convey important information, an alternative indicator must be used, such as an asterisk (*) or other symbol.
	2. Make sure contrast is good.
Documents must be organized so they are readable without requiring an associated style sheet.	Style sheets may be used for color, indentation and other presentation effects, but the document must still be understandable (even if less visually appealing) when the style sheet is turned off.

Continued on next page

TABLE 1.1 CONTINUED: Achieving Section 508 (Disability and Accessibility) Requirements

Excerpt from Sec. 508 Standard	What You Need to Do to Comply
Redundant text links shall be provided for each active region of a server-side image map.	Separate text links must be provided outside of the server-side image map to access the same content accessed by the image map hot spots.
Client-side image maps shall be provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.	Standard HTML client-side image maps must be used, and appropriate `alt` tags must be provided for the image as well as the hot spots.
Row and column headers shall be identified for data tables.	1. Data tables must have the column and row headers appropriately identified (using the <th> tag) 2. Tables used strictly for layout purposes must *not* have header rows or columns.
Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.	Table cells must be associated with the appropriate headers (e.g. with the `id`, `headers`, `scope` and/or `axis` HTML attributes).
Frames shall be titled with text that facilitates frame identification and navigation.	Each frame must be given a title that helps the user understand the frame's purpose.
Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2Hz and lower than 55Hz.	Make sure no elements on the page flicker at a rate of 2–55 cycles per second, thus reducing the risk of optically induced seizures.
A text-only page, with equivalent information or functionality, shall be provided to make a website comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.	1. A text-only version is to be created only when there is no other way to make the content accessible, or when it offers significant advantages over the "main" version for certain disability types.
	2. The text-only version must be up-to-date with the "main" version. 3. The text-only version must provide the functionality equivalent to that of the "main" version. 4. An alternative must be provided for components (such as plug-ins and scripts) that are not directly accessible.

Continued on next page

TABLE 1.1 CONTINUED: Achieving Section 508 (Disability and Accessibility) Requirements

Excerpt from Sec. 508 Standard	What You Need to Do to Comply
When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.	1. Information within the scripts must be text based, or a text alternative must be provided within the script itself, in accordance with these standards.
	2. Either all scripts (e.g. Javascript pop-up menus) must be directly accessible to assistive technologies (keyboard accessibility is a good measure of this), or an alternative method of accessing equivalent functionality must be provided (e.g. a standard HTML link).
	Generally, you'll need to think about whether any Dynamic HTML you use is simply eye candy or if your navigation system and other functionality relies on it. If the latter is the case, your website is not considered accessible unless you provide alternate means of accessibility.
When a web page requires that an applet, plug-in, or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).	1. A link must be provided to a disability-accessible page where the plug-in can be downloaded
	2. All Java applets, scripts, and plug-ins (including Acrobat PDF files and PowerPoint files, etc.) and the content within them must be accessible to assistive technologies, or else an alternative means of accessing equivalent content must be provided.
When electronic forms are designed to be completed online, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.	1. Make sure all form controls have text labels adjacent to them.
	2. Make sure form elements have labels associated with them in the markup (i.e. the ID and form HTML elements).
	3. Dynamic HTML scripting of the form must not interfere with assistive technologies.
A method shall be provided that permits users to skip repetitive navigation links.	1. A link must be provided to skip over lists of navigational menus or other lengthy lists of links.

Continued on next page

TABLE 1.1 CONTINUED: Achieving Section 508 (Disability and Accessibility) Requirements

Excerpt from Sec. 508 Standard	What You Need to Do to Comply
When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.	1. Make sure the user has control over the timing of content changes

As you can see in Table 1.1, your primary concern is going to center around making sure that images have text descriptions, and that your web page doesn't rely on things like applets, DHTML, and CSS to achieve basic functionality. Think back to the earliest incarnations of HTML. If the web page you're building today could have worked in an ancient browser, you're well on your way to building an accessible web page.

When adding bells and whistles, make sure you include alternative ways for users to access the content. Like everything else, a bit of planning goes a long way in achieving this.

Functional Design Documents

A *Functional Design Document* (also referred to as an FDD) describes how the website should operate. The FDD is written for the people who have to actually write the code to make the site work, and provides an overview of website's functionality. Usually the FDD is a collaboration of three groups: the client, the Quality Assurance (QA) team members, and programmers who will be developing the site.

In a way, it's like putting the website down on paper. The FDD describes each page, what it does, how it behaves, how it connects to data and, what data it connects to. This document also describes the processes for validation—what needs to be validated and when—as well as the relationship between data and the user interface.

Searching mechanisms, registration pages, and member login pages are all examples of the functional content covered in the FDD, as is the description of exactly *what* these things do and how they do it. For example, you may have an entire section that describes just a tiny part of the front page, or that specifies an authentication controller that manages site traffic. The processes used for these elements all become part of the FDD.

Once you've become familiar with the `DreamweaverCtrls.DLL`, which is the main component you'll use in most of your applications, developing an FDD will be easy. You'll be able to focus on which Dreamweaver control elements to use and when you want to use them, as well as any specific attributes, parameters, or subelements you might need.

An FDD can be broken into two main parts: The Project Requirements Document, which contains an overview of project requirements, and a Section and Module Requirements Document, which describes the specifics of your component functionality.

Project Requirements Document

Earlier I described some of the considerations you'll take into account in determining your site's aims and objectives, and now it's time to flesh those out for the first phase of your FDD, the Project Requirements Document. Sometimes this piece is a separate document, and sometimes it is part of the FDD. It depends on the size of the project and the disposition of stakeholders. Roughly, it should cover requirements for the following aspects of your site:

- Mission and goals
- Audience definition
- Competitive analysis
- Site content, including a list of content items and groups of items
- User experience and interaction scenarios
- An overview of functional requirements
- Structure
- Design requirements
- Maintenance requirements
- Security policies

The length of the Project Requirements Document is going to depend on the complexity of your site and your time constraints, but you can use the preceding bullet points as a starting point. In journalism, there is a convention called the pyramid style of writing, which dictates that you start with the most important points and increase the detail as the reader gets deeper into the article. You can use the same principle here. Cover everything in broad strokes, and as time allows, go into more detail.

Section and Module Requirements

Once you've described your basic project requirements, you'll need to put more meat to the bones by describing the actual functionality of the web application. Since I'm pushing the notion of components so hard, it seems only fitting to describe the next part of the Functional Design Document as the part that contains the requirements for the sections and/or modules of the site.

There are two ways to represent sectional and module requirements, which are essentially the requirements for specific chunks of your web application. One way is to simply delineate

them within the scope of your FDD. You can create a group of separate documents named something like "XXX Module Requirements," where XXX represents the name of the module, such as "Catalog" or "Shopping Cart." You may have several of these, with a top-level document containing an overview. Another way to create the requirements is to build a flowchart or use a modeling notation such as UML (Unified Modeling Language).

In the following sections, we'll look at both methods for creating the module requirements.

Including Section and Module Requirements within an FDD

This part of the FDD describes the requirements of individual modules, with an overview of how they all work together, written by and for developers. These requirements may contain diagrams like the flowcharts described in the next section, just as a diagrammed model may contain limited documentation or text. Once again, focus on the broad strokes first, and work details in as needed as time allows.

Here's an example of the outline for a Functional Design Document's Section and Module Requirements:

1. Web Application Overview
2. The Authentication Module
3. The Catalog Module
4. The Shopping Cart Module
5. The News and Events Module

Of course, your own Section and Module Requirements will likely contain different modules. You may be developing an entire Content Management System as part of your site, which will possess its own set of requirements.

Using Flow Layout Documents and Languages

When you're using a diagrammed model for this part of the FDD, a sketch like the one in Figure 1.2 is what becomes your model (although a ridiculously simple one, in this case). You can schematically represent your site in any number of ways. Most organizations use either flowcharts or UML (Unified Modeling Language) diagrams. I'm going to focus here on UML because it has become popular with developers and because using it, even in a rough form, is very similar to flowcharting. If you don't know UML and don't have time to learn it, you can borrow some of its principles to help you organize your web application.

UML is a standardized modeling notation for representing software applications. You can use UML to model the way your web application is used by each user, or the way each component behaves when interacting with other components.

NOTE A very simple, easy-to-read reference to UML symbols and relationships can be found at `www.webreview.com/2001/05_18/developers/UML_checklist.shtml`.

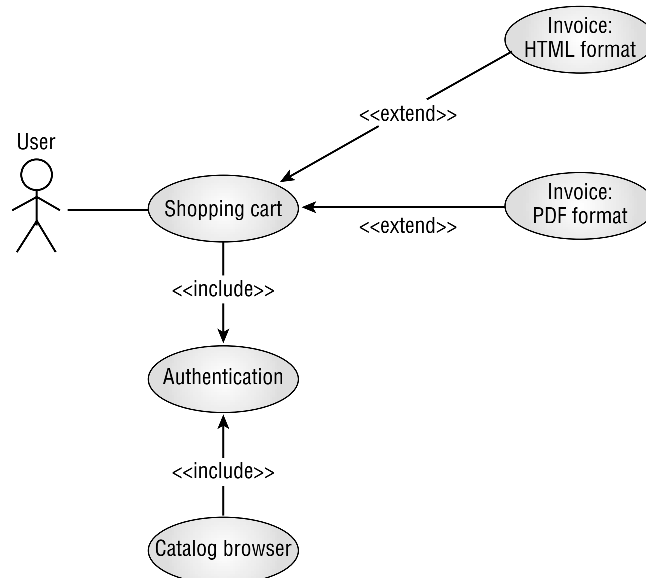
Navigation and User State

UML consists of two broad categories of diagrams: *Use cases* are diagrams walk someone through a typical user session at the website. *Sequential* and *collaboration diagrams* provide an abstract view into the relationship between the various objects in the website, particularly as they relate to each other. You could say they describe the navigation of the site, but they actually go considerably beyond that because they focus on the application-level functionality of the site.

Use Case Diagrams

A use case diagram details the user experience when they initiate a browser session on your site. This is a good place to catch problems before they develop. As you think about the general user experience, consider how your website will manage the various options a user will encounter. Figures 1.3 and 1.4 show use case diagrams created using UML notation, for two different applications. The little stick figures are called *actors* in UML parlance.

FIGURE 1.3: A use case diagram for a simple shopping cart



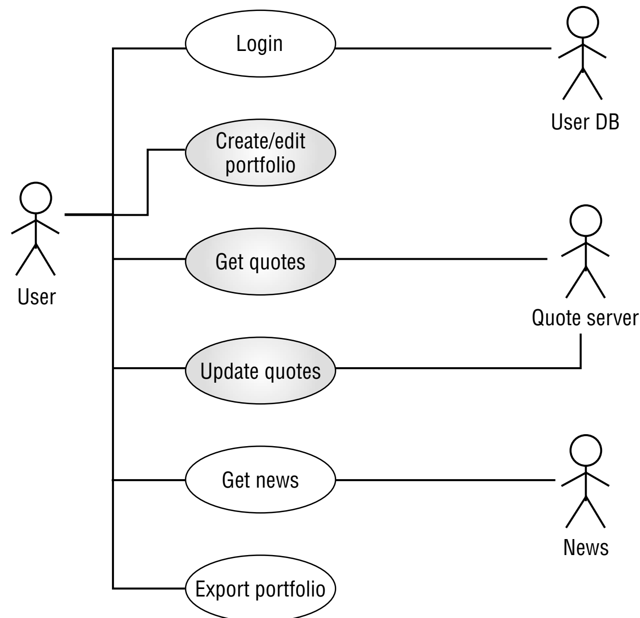


FIGURE 1.4: A use case diagram for a stock market portfolio web application

NOTE A tutorial on UML is beyond the scope of this book. Jim Conallen is an authoritative voice in this field, and any Google search with the term "UML Conallen" will yield results of his writings. You can also check out a whitepaper he wrote for Rational Software, makers of a high-end UML tool called Rational Rose, at `http://www.rational.com/products/whitepapers/100462.jsp`.

Sequential and Collaboration Diagrams

Sequential and collaboration diagrams describe the flow defined in a use case by detailing the way the various classes and/or components are implemented. Figure 1.5 shows a very simplified navigational system for a stock quoting service, and Figure 1.6 shows a navigational view of the very roughly drawn flowchart first seen in Figure 1.2.

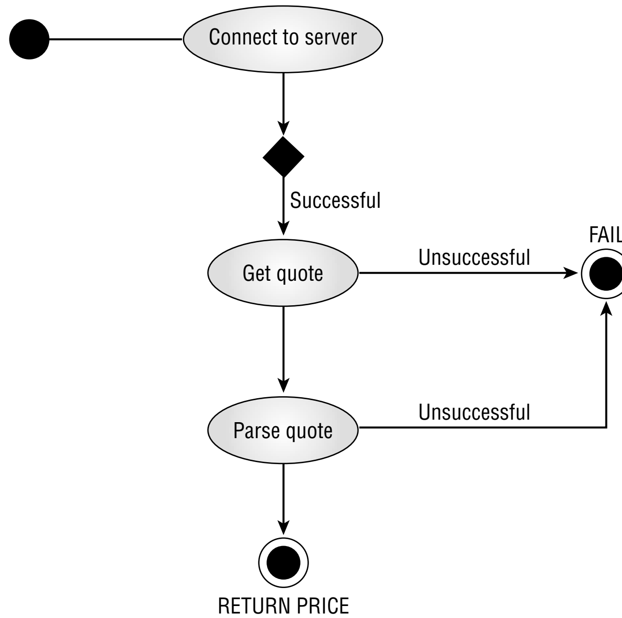


FIGURE 1.5:

Navigational diagram for a simple stock system

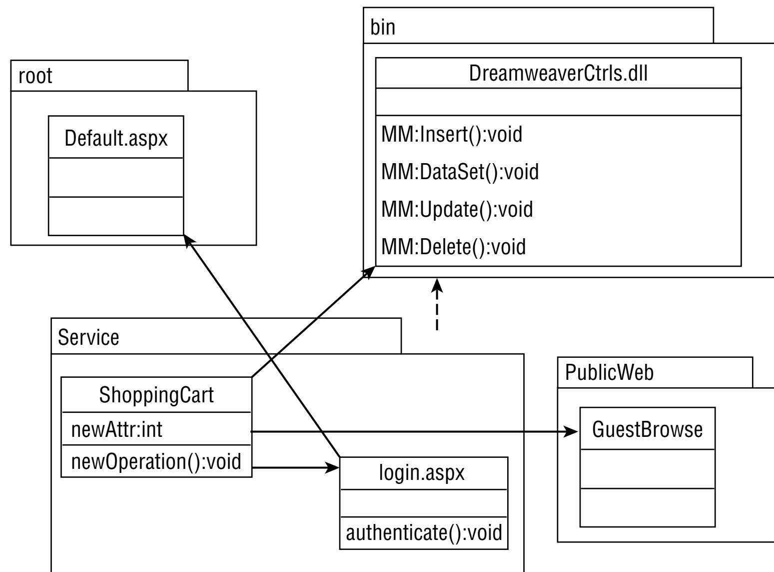


FIGURE 1.6:

The earlier rough schematic diagram takes on a bit more detail

Notice in Figure 1.6 that we've created a box representing the class files that make up the `DreamweaverCtrls.dll`. When drawing your diagrams, you'll want to note any dependencies on a control that certain parts of your site will have. You may also have other dependencies that you'll need to make note of, such as CSS stylesheets, JS (JavaScript files with a `.js` extension), and server-side includes.

TIP You can design more than navigation and user state diagrams with UML. In addition, you can construct class diagrams that describe the functionality of your classes in great detail.

Another kind of diagram similar to those just described is the *sequence diagram*. It focuses somewhat more on the user. Compare the diagram for the stock price quoting application in Figure 1.5 with that in Figure 1.7. Figure 1.7 uses a sequence diagram to track users' relationships with the functions or methods that take them to where they're going.

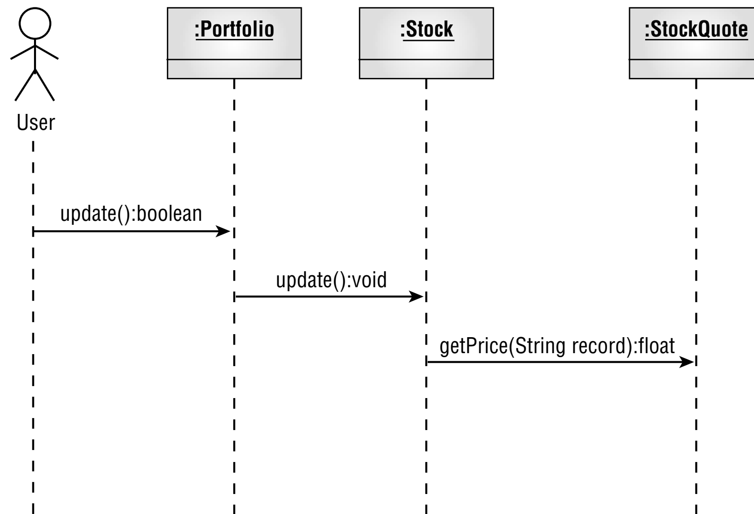


FIGURE 1.7: This sequence diagram describes the relationship between the user and the web application's functions and methods.

Other Types of UML Diagrams

Within the two broad categories (use cases, and sequential and collaboration diagrams), there are several other kinds of diagrams useful for FDDs. How they fall within the two main classifications of site diagrams depends a lot on who you talk to and which UML modeling software you use. They include activity diagrams, class diagrams, component diagrams, object diagrams, and state diagrams.

Generally, for our purposes, class building has already been done for us by the kind folks at Macromedia through the `DreamweaverCtrls.dll`, which contains a number of public classes useful for manipulating database-driven websites. So you won't need to diagram the specifics of the `DreamweaverCtrls.dll` classes. However, if you plan to build on them or to build your own controls, further exploration of UML's capabilities is a good idea.

NOTE The `DreamweaverCtrls.dll` is an integral part of the Dreamweaver IDE's relationship to .NET. You'll learn more about it in Chapter 6, "Working with .NET Validation Controls."

The complexity of your website will likely dictate how deep into UML modeling you'll want to go, as will, of course, your available resources. You should at least take a look at generating some basic diagrams that describe the flow of your application and potential hazards that can develop along the way. There is no rule that you have to use UML. An ordinary white paper napkin has been known to do the trick. But UML introduces a helpful, standard methodology for those who are inclined toward high-level organization and best practice.

Design Requirements Documents

I've talked a lot about the component-based architecture inherent in building applications using Dreamweaver MX. The real challenge comes when you are faced with trying to fit all these components within the scheme of your overall document, like the simple *design requirements document* that we'll look at next. Coming up with the design requirements document is really something of an art, but generally I've found that components can be dropped into a table cell, and you can design your pages based around the table. You can use placeholders when you don't have access to a live representation of the component.

There are specific steps you can take to increase your team's design productivity, and it's worth taking a close look at those.

Building the Mock-up

Mock-ups are an essential element to achieving the appearance you want for your website. In larger organizations, there are usually two mock-up stages. A graphic designer designs a mock-up of the site's pages in a graphics or page layout program, and an HTML developer writes the code to make them look exactly like the mock-up, to be presented as part of a design requirements document. The web design profession has a phrase for the goal at this stage: *pixel perfect*.

I remember working years ago as an art director for TMP Worldwide, parent company of the Monster Board. Generally, I designed ads and brochures, but I also designed company profile pages for the Monster Board. I submitted the mock-ups to the Boston office in the form of color prints. What came back was not pixel perfect, but rather the best HTML the developers could come up with. Today, with CSS and with Dreamweaver (or without, if you're a good hand-coder), you can create pixel-perfect renditions of design document mock-ups. When your code is developed properly, only the most ancient of browsers will fail to interpret your efforts well.

What I'm trying to say here is that even if you're a one-person operation, designing a mock-up in a graphics program is a good idea if that is where your creative juices flow best. If you're used to designing in Photoshop or even Quark XPress, go ahead and do it, and then switch over to Dreamweaver to implement your grand design.

If you're working on a large development team, this routine is probably built into the entire process, anyway. Let's take a look at how this is achieved. We'll start with a design document that includes the image seen in Figure 1.8.

FIGURE 1.8: The Design document mock-up

Building the Images

The first step in moving from mockup to HTML is building the images. This is simply a matter of cutting them out of the graphics program. Whether you use Adobe Photoshop, Macromedia Fireworks, or some other program, it is easy to cut out pixel-perfect images and build your web page from them.

Building the CSS

The next step in transforming the design document to a working web design is working with the Cascading Style Sheets. Let's just take one small piece of the mock-up graphic and build a portion of the web page from it to see how it works.

Take a look again at the original graphic (Figure 1.8) and notice the dotted line around the white login box. This is accomplished with a CSS border around the `div` element that contains the table.

1. Create a new Dreamweaver file. You can choose either a dynamic file or a basic HTML page. If choosing a dynamic file, choose Dynamic Page from the New Document dialog box category list (you may have to choose the General tab first). Then choose ASP.NET C# from the Dynamic Page list that appears after you click Dynamic page from the Category list (Figure 1.9). To choose a basic HTML file, choose Basic page from the category list, then HTML from the Basic page list that appears. Click the Create button.

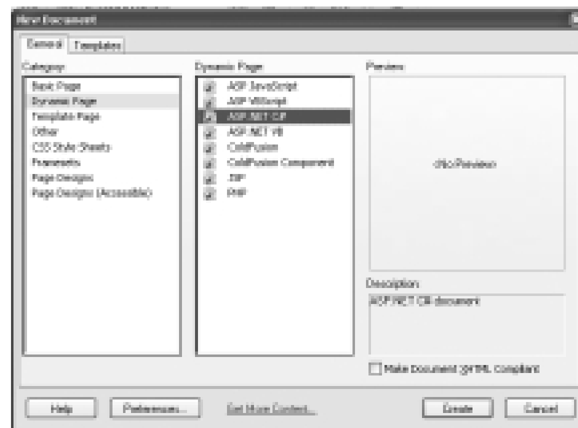
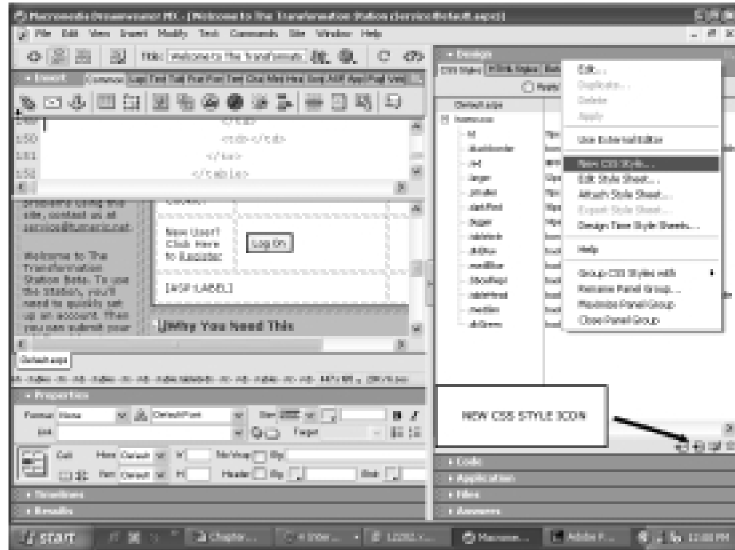


FIGURE 1.9: Creating a new Dreamweaver document

2. Once you have created a new page, you're ready to develop a CSS. In the CSS Styles tab of the Design panel, make sure the Edit Styles radio button is clicked, and then choose New CSS Style from the Design Panel menu (Figure 1.10). A new dialog box appears called New CSS Style. You can choose to either define the style sheet in a new style sheet file or define it within the document, by clicking one of two radio buttons in the dialog box. If your web page already has a stylesheet linked to it, the selection box next to the Define In radio button contains the name of the currently linked stylesheet. You can either use that or create a new one by clicking the selection box. The top selection is (New Style Sheet File). If you choose to create a new stylesheet, Dreamweaver will ask you where you want to save it.

FIGURE 1.10:

Creating a new CSS style



3. In the same New CSS Style dialog box choose the Make Custom Style radio button (which is selected by default).
4. Name the class **`.tablebrdr`**, making sure to include the dot at the beginning of the name. This lets the browser recognize it as a CSS class rule later when you call it into your code.
5. Choose the Define In (New Style Sheet File) radio button and click OK; a new dialog box will appear. Name the file **`home.css`** and save it to your site (if you have established it) and click OK. Dreamweaver creates a link to the stylesheet in the HTML `head` element that looks like this:

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

6. In the CSS dialog box, click Border in the category list. For the top border, set the Style to dotted, and keep the Same for All check box checked.
7. Set the Width to thin, keeping the Same for All check box checked.
8. Choose a color. When you click the little Color icon, a palette appears from which you can pick a color. I chose a very dark blue for this example (#000099).
9. Click OK. The new CSS rule definition should look like this in `home.css`:

```
.tablebrdr {
   border: thin dotted #000099;
}
```

10. Insert a table using the Insert Table dialog box, setting the Width at 100% and the Cell padding at 10. After you've inserted the table, select it with your mouse and use the Properties palette to change the background color to white (#FFFFFF).
11. In the Design palette, click the Apply Styles radio button.
12. Select the table, and click the style you just created (`tablebrdr`).

NOTE I used .NET form controls to create my form widgets because I knew they would be interacting with the .NET engine. You can do this if you want to, but it's not central to this specific exercise.

13. Let's move on to the Login button, which is designed somewhat differently than the traditional submit button. The important parts here are the background color and the font, so return to the Design palette. Click the CSS tab and make sure the Edit Styles radio button is clicked. Then create a new style called Login by navigating to New CSS Style as you did in Step 2.
14. Make sure the Make Custom Style (class) radio button is clicked, and enter **`.login`**. This will be the name of the style sheet you assign to the submit button named `login`.
15. Choose the Define In `home.css` radio button. This button will have a drop-down list next to it containing all the available style sheets for this page. There should only be one, but you can technically link more than one style sheet to an HTML page. (Note that Dreamweaver will add the required dot character before the name of your style sheet definition if you forget, but if you hand-code style sheets, don't forget to start the name with a period.)
16. When the dialog box appears, Type options should be displayed. A drop-down list of fonts appears. If a font you want isn't in the list, scroll down to Edit Font List. A text box will appear. For this example, enter **Frutiger, Verdana, Arial, Helvetica, sans-serif** into the Font text box and enter **11** in the Size text box, keeping the default measurement of **Pixels**.
17. In the Category panel, click on Border. Set the Style to Outset, the Width to 1 pixel, and the Color to Red, making sure that Same for All is checked above each box. This ensures that the left, top, bottom, and right borders all have the same style.
18. In the Category panel, click on Box. Enter **1** into the Padding Options panel, making sure that Same for All is checked.

19. Click OK. The new style sheet rule should look like this in the `home.css` document (which you can view in the Document editor by double-clicking it in the Files panel):

```
.login {
padding:1px;
background-color:#9999ff;
border:red outset 1px;
font-family:Frutiger, Verdana, Arial, Helvetica, sans-serif;
font-size:11px;
}
```

Listing 1.1 shows how to apply the style sheet and create a table that results in a look that matches the design document graphic. Figure 1.11 shows that its final rendering in the browser matches our design document.

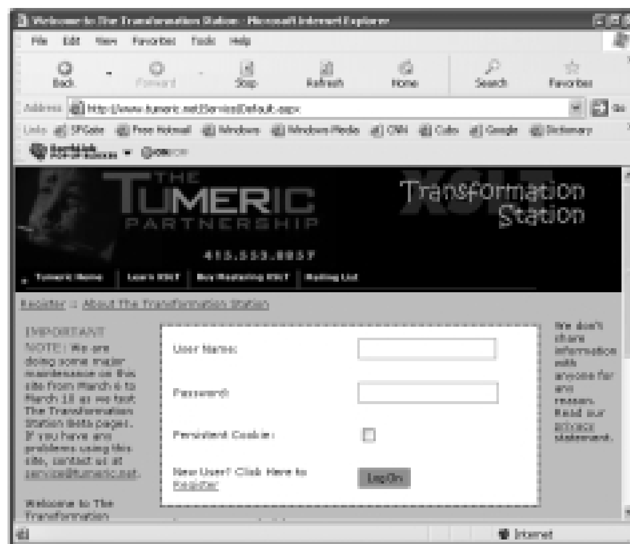


FIGURE 1.11: We have a match! The browser should render a page that looks like our design document.

Listing 1.1 A CSS-based table derived from a design document

```
<table width="100%" border="0" cellpadding="10" cellspacing="0"
bgcolor="#FFFFFF" class="tablebrdr">
                <tr>
                  <td><table width="100%" bgcolor="#FFFFFF">
                      <tr>
                        <td>User Name:</td>
                        <td>
```

```
                                <input id="txtUserName" type="text"
                                name="txtUserName" runat="server">
                            </td>
                            <td>*
                            </td>
                        </tr>
                        <tr>
                            <td>Password:</td>
                            <td>
                              <input id="txtUserPass" type="password"
                              name="txtUserPass" runat="server">
                            </td>
                            <td>*
                            </td>
                        </tr>
                        <tr>
                            <td>Persistent Cookie:</td>
                            <td>
                              <asp:checkbox ID="chkPersistCookie"
                              runat="server" AutoPostBack="false">
                              </asp:checkbox>
                            </td>
                            <td></td>
                        </tr>
                        <tr>
                            <td>New User? Click Here to <a
href="../createClient.aspx">Register</a></td>
                            <td><input class="login" id="cmdLogin" type="submit"
value="Log On" name="cmdLogin" runat="server">
                            </td>
                            <td></td>
                        </tr>
                        <tr>
                            <td colspan="2"><asp:label  ID="lblMsg" runat="server"
Font-Size="10" Font-Name="Verdana" ForeColor="red"></asp:label>
                            </td>
                            <td></td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
```

The only thing really notable about this code is that when you are in Dreamweaver, you'll first want to insert a table with a cell padding substantial enough (10 pixels in this case) to create a gap between the dotted line and the content, and then add your content table.

Wrapping Up

I hope I've succeeded in talking you into doing something you might not have wanted to do. Yes, workflows are boring. Dreamweaver is fun. It's easy to understand why workflows aren't always initiated at a project's start. However, seasoned developers rely on them because we've all learned the hard way. I've seen projects grind to a halt because there was no advance planning.

Your workflow begins with the most basic of starts: defining your objectives. From there, it proceeds to mapping out a strategy that defines your website through a combination of requirements documents, functional design documents (FDDs), and design documents. Then it's up to team members to implement the site. The developers will review the functional design documents and write the code according to the specs in the FDDs, and the graphics designers will review the design documents and match the designs in those.

The final result will be a web application with far fewer surprises than you would otherwise find. Surprises will always come, of course, but good planning means there will be fewer of them.