Laszlo Naszodi

CREATING FAST WEB APPLICATIONS

Selected lectures for not-so-dummies

Edition 5.13

Published by

ScriptWell.net

First edition 2005

Fifth edition 2007

Table of Contents

I cannot associate corporate culture with cultural values more than the culture of fungi that makes the milk sour.

About the Author and His Views

In most of my life I have worked for small companies or for even smaller, for myself. With a complex occupation of being a systems designer/ developer/ programmer/ QA analyst/ database analyst/ technical writer at the same time, I have operated in a simple and flexible way. I knew my financial limitations and that of my customers, mostly small business owners. Instead of having my customers buy Macromedia's DreamWeaver, Coldfusion Studio or Microsoft's Visual Studio for one thousand dollars each, I developed reusable, open-source codes that functioned the same way as those few necessary features of the heavy-duty commercial software.

Same with databases. Unless the job or the customer required a really big database, I haven't utilized an expensive SQL Server or Oracle. In most cases I could go far without any database or with an inexpensive MS Access solution. Everything seemed simple. When I needed a permission or access to certain resources, I just told my coworker at the next desk to set up the necessary rights for me. He was the systems administrator, the database manager, the network administrator and the web master, without having these titles. He was Joe, the master of his domain. I might have had to mention to him that my problem was that I could read but could not write in a certain folder or in a certain database table. That was it. He knew what I needed and how to meet these needs with a minimal compromise. He lifted the restriction that obstructed me and I could continue my work in a minute.

The first time I worked in a corporate environment, I was shocked to see how difficult the same process could be. The one-minute task to gain write access to a folder or a table took a month; a busy month for many people. I heard the clatter of fire brigades running up and down. I had to fill out lengthy forms, have my supervisors authorize them, and answer questions that I had nothing to do with. The request went through an unclear labyrinth of bureaucrats and semi-educated "specialists". The process was covered by a unit called Help Desk. I could not look behind the desk to figure out who is "helping", i.e., holding me back. The systems administrator did not know anything about databases, the database manager did not deal with system issues and the security administrator had his own pigeonhole, too. Everybody was content with the hard and expeditious work but me. They saw a complex process, I saw inefficiency. Instead of removing the roadblock, they kept tossing the ball back to me. I had to be rude to make myself clear: "*A small part of my job is to create new records, but I am restricted to write in my database. I have to do dozens of other tasks to finish my job. You do whatever you must do to solve this well-defined access problem. I don't know and I don't care whose responsibility it is inside the IT department. Just don't return my request to me.*" Needless to say that I was out soon. I did not fit in the corporate culture where people were too busy showing how important they were.

The second time when I was employed by a large corporation, I was hired as a technical writer. By that time the PR lingo changed the word corporation to the better sounding organization. Still, disorganization ruled. Departments had conflicting interests and they fought each other, even at the cost of damaging common interests. Our section needed a Web application. We asked the IT department if they could develop it. We got a half-year, six-person, half-a-million dollar offer, which was unacceptable. I mentioned that I could complete the project alone in two months for my regular salary. The IT people who were supposed to cooperate with me in the project did not forgive me for taking their opportunity of easy money making. They set up rules and restrictions for me to fail. I was supposed to follow their unreasonable standards, using their ugly, bloated software tools, their hulking development environment and their over-priced database server. When I agreed to do so, they fell short to provide me with those documents and resources.

While I was waiting for the resources, I created a prototype that did not use anything but a Web browser. My development environment was a text editor, slightly smarter than Notepad. It colored my program with the intelligence level of the Homeland Security alerts. I could only use text-based elements: HTML, CSS, JavaScript and XML. I did not get access to a database or a programming language to write server-side scripts. I could not even have an Internet service, like IIS, Apache or Personal Web Server. After two months I had a working application that did not need server-side scripting or a database. As a matter of fact, it did not need a server at all. It was fully functional and those who had access to a shared folder on the Intranet and had a Web browser on their workstations could run the application. It was fast and smart and low maintenance. Fast, because it did not make a roundtrip to the server at every user intervention as if I developed the app with their standard methodology. Smart, because it could find and display all the information from XML files as if a relational database was running behind the scene. Low maintenance, because it was independent from service providers ***who*** needed maintenance. And fast, smart and low maintenance, because its size was a fragment of the similarly smart applications' size that were developed with the usual large enterprise technology.

The malfunction of the big organization made me follow a technology that I have done before as a small business owner and someone who contracted with less resourceful small businesses. The prototype went into production before I got the green light from IT to start the development. I am thankful to this large company for making me prove that the cost-conscious craftsman's approach can be efficient in a wide enterprise environment, in a relatively large project, too.

Corporations are not and cannot be as customer centric as small businesses. The main goal of all big companies is to please their shareholders with the highest available profit. They cannot act primarily for their clients. It's more profitable to spend on marketing, delude and make us ***believe*** that their products and services are good, than to spend on quality improvement. I'm not anti-market, I'm pro-consumer. I would like to see free competition prevailing by raising quality and lowering prices, beneficial to the people. Trends of merges point to the other direction, to restrictive practices of monopolies. We only get the bad part of competitive market. We have to put up with the raising prices, with the marketers via phone, TV, papers, magazines, radio, Internet, bill boards, drinking cups and anything and everything that has a surface, is packaged, is moving or not. In general, while running around the market-driven world, we shouldn't forget why all the Canossa-going, compromise, perpetual back noise of advertisements, and lies-with-smiles are for. Big businesses are for the largest possible market share and not for us, people. They like us if it's profitable and they kill us if it's more profitable.

Big software companies go to China and agree to help the oppressive government to pin down human rights activists. Not that these companies were against human rights but because this is how they can make big business before better, more conscientious companies. You may think that I am opinionated. Is the executive director of Human Rights Watch opinionated, when he says: "…If it implements its pledge, Yahoo will become an agent of Chinese law enforcement." Is the European Commission also opinionated in its March 2007 statement? It says: "... Microsoft has established ***unreasonable*** prices for its protocol licensing of its server technology in Europe." The Commission characterizes Microsoft's proprietary server software protocols, which is protected by patent, copyright and trade secret law, as containing ***virtually no innovation***.

I do not hate big corporations. I may dislike them. Not only because I must wear a suit there and shave every morning although I usually spend eight hours with a computer staring at each other in a cubicle without meeting a person. How would a policy maker feel if he would be forced by his own rules to the extremes, to wear jacket and tie in the bathtub where nobody sees him?

I cannot associate corporate culture with cultural values more than the culture of fungi that makes the milk product sour. Nothing is wrong with fungi. After all, we like fungus-produced cheese and wine. Should we like spoiled milk products, too? Corporate-supported process of innovation is like fermentation. The final product can become either delicious or simply stinky. Corporations help innovations if and only if the proper personnel thought that the particular innovation was profitable, regardless if it makes sense. Half of the innovative efforts aims at making products proprietary, i.e., to hide information even at the cost of making the product clumsier. Corporations are not necessarily the champions of progress. It is ***not*** true that what is good for Ford is good for America. A man may be respected for his inventive ideas but he should be ashamed of others. Henry Ford might have been a genius but he was also an ignorant, arrogant, and insensitive man. But let's not go in there. I am talking about a technology that enables us to create high quality expert systems without using costly tools during and after development. Does it work for large enterprise projects? In certain cases it does not. However, it works in many projects, where most developers and organization leaders think that a pricier toolset and approach are necessary.

I would like to show here that completing great projects doesn’t necessarily need expensive tools. I am not saying that we should always fight a big beast with a stone. However, we should not use heavy artillery when a slingshot would do it. Go David, go!

The Web developer is a device that turns coffee into Web pages

Preface

To Whom and About What I Am Talking

Before I started to create Web applications, I read some books and articles on the topic. Some good, some bad and some really ugly. There are countless publications for beginners detailing how to color the page or make a sentence bold. A systematic study of the basics is useful but one should not spend too much time on learning those few HTML tags. What I needed was the tricks and tips of the trade, which make a Web site fast, usable, maintainable and aesthetic. I am offering the reader a fast track. Once you have a basic knowledge of HTML, CSS and JavaScript, you can jump on my wagon. Following the examples of ***Creating Fast Web Applications*** you will become an advanced Web developer much faster than if you keep reading those beginner's books.

The alphabet soup

Have you heard about RAD, as Rapid Application Development? Yes? Good. How about DRA, as Development of Rapid Applications? Yes? I doubt it. I just made up this term. Creating fast applications is out of scope of the IT industry’s mainstream. If you use a developmental environment and the resulted application runs slowly, they say that it’s not their fault; you should buy a faster, more powerful computer. They can convince you to buy their software, which will quickly do the developing job for you. If you fail to produce the application as quickly as the smiling guy from the demo, you won't complain. But if they promise that the resulting app will be fast and in fact, it will not be, you may have the right to ask your money back. That's why there is no such thing as DRA for sale.

The title and the content

What a lame book title! my son said. At least, why don't you change the adjective to ***slick***? My son's remark made me think. Creating ***fast*** Web applications is very important but speed is not the only characteristic of a computer program. These lectures are focusing on speed but they concern about aesthetics, maintainability and usability of browser based applications as well. In short, slick covers the topics better than fast, but it sounds too populist to me and it connotates with superficial, not just with fast, smooth, compact, nice and smart. As so many times, my son was right but after considering all aspects I neglected his advice. I did not want to imply that my collection was one of those "***How to Become a Pro with No Brain***" books. That's why the second part of the book has been included. It deals with the second most important aspect of slickness after speed; with usability. The third part is about both: How to make an application fast and usable with some ideas of AJAX.

A title must be captivating, as short as possible, still distinctive, descriptive and precise. My first choice was one of my articles' titles: ***Info On the Fly***. It is catchy and short but does not give a clue about the topic. A title like that can hide anything from a book on gossiping to airplane flying instructions. I knew exactly what I wanted to write about but I could hardly select the first word. Should it be building or creating? I relate the word ***building*** to the persistent diligence and effort of the construction worker and the word ***creating*** to the vision and inventiveness of the engineer. The job of Web design and development requires both brain and buttocks. What is more important? I chose the word creating because I thought that developing applications require more brain than endurance. Actually, several What-You-See-Is-What-You-Get Web site building software take the most part of burden. If one has imagination, one can build nice looking static Web pages in no time, with no sweat. The trouble starts when one visits these sites. First, one does ***not*** get what you have seen. Second, these pages open much slower than those created without using the page editor's WYSIWYG feature. And new problems rise when you want to create some dynamism on the pages. When I say dynamism, I don't mean dancing signs and flashing pictures. A dynamic page has to do something useful; to calculate, to select, to give context-sensitive choices. The WYSIWYG mode is not efficient for creating dynamic sites and applications.

I struggled with the last word of the title, too. Should it be site or application? People associate the Web with Web sites and they are almost completely right. However, a ***Web application*** is not necessarily a site on a network. Web application is a term for programs that use a browser as their graphical user interface. It should have been called something like browser-based application. It includes but is not limited to Web sites. A Web site can be a set of static pages full of pictures of a pet. It is just a Web site, not a real Web application even if it shakes and barks. We can expect from an application to be dynamic. Well-written Web apps run in any modern browsers, in almost any computers with almost any monitors. As opposed to most software applications, they are platform independent because their codes are in text format, readable by any operating systems. They don't have to reside on a network. A so-called Web application does not necessarily need a Web server, either. I can't change an industry-wide established terminology and I will interchangeably use the terms Web application and Web site as others. However, I always mean a set of interconnected pages which opens in a browser. Nothing more and nothing less.

Let me explain the subtitle, especially the reference to the target audience I described as ***not-so-dummies***. In the last decade the average computer literacy shifted from beginners' toward intermediate level. A generation grew up on the "For Dummies" series. Amateurs learned how to handle various computers, software tools and build Web sites by moving the mouse around. Most people took advantage of the convenient approach. They grabbed one of the Web site building software with the corresponding "For Dummies" book, turned on the WYSIWYG mode, and formed their pages without knowing, what is going on behind the scene. They can create nice pages with limited functionality now. I want to reach those who already learned the basics and feel the urge to move further, leaving the children's play of pointing-and-clicking behind.

Web pages written in a What-You-See-Is-What-You-Get development environment may become dysfunctional or even unusable sometimes. Most of all, they are always slower compared to the ones edited in source code view. A crucial review of the automatically created source codes is necessary to speed up these pages. It's not a developer but a senseless software that writes the code. A program, which does not care about superfluous repetitions or file size. Names and id's, like x139 and Table07, automatically given by the software, don't mean anything and there is no person to ask how and why a code segment was created. The generated code becomes more and more difficult to decode as it grows. You can start a project in WYSIWYG mode, show off with the fast progress but you cannot finish it. At a critical point you must stop pointing and clicking and must write and rewrite substantial portions from scratch.

The last 5% of the project will need more time then the first 95%. Why can't we hand code from the beginning then?

An example

Let's say you have a restaurant at a ski resort and you want to display an accurate menu and wine list next to the entrance. Every time you change the special of the day, run out of an item or a new wine delivery arrives, you need to update the list. Would you retype the new list, go out to the freezing street and replace the sheet of paper three times a day while the snow blows in the open window of the menu box? Or get an old, stand-alone computer for a $100 that has a browser, place its monitor in the menu window, and as soon as you reedited a simple text file inside the building, the new menu is shown in a fancy format outside. The program can automatically update the price list according to the part of the day, before breakfast, lunch and dinner time. This is a simple Web application, but the periodical updating function would be hard to write with a WYSIWYG tool. Using the samples of this book, you can hand-code a dynamic price list application in Notepad and run it in a browser, without having a Web site or a network. Do you want to put the price list on your Web site? You can do it, too, without extra reformatting efforts. The online and offline versions will be the same.

Inspiration, perspiration

Writing the first application from scratch is time consuming. During my programming career I had to switch systems and languages several times. At these times my mentor used to say: "Never write your first program in a new system. Start with the second." He meant that we should pick a source code, any working code in the particular language, and rewrite it. This book presents enough sample code to rewrite.

Working with source code is a great intellectual challenge, and it always delivers mental satisfaction to me. The challenges make me excited and sometimes sweat. These selected lectures are for Computer Science/IT teachers and students, for professional and amateur developers who want to develop themselves, too, for small business owners who can't afford to buy expensive development environments and office suites. For those who already see the limitations of the code generating tools but don't have the time to cope with picky languages and disagreeing browsers. I hope that the ideas make the readers as excited as myself and the given solutions help to spare their sweat. However, if you do have time and urge to sweat, you can find practice questions and problems at the end of each lecture.

PART 1. SPEED

Trust me. It won't take long and it won't hurt.

Web Design from Scratch

Intro

No, I don't want to launch an N+1st Web design course for dummies. I don't want to make the false impression that Web development is an easy job. Yes, if one could format a document in Microsoft Word, one could create an HTML version with a click on the "Save As Web page" option. The result can be a nice page with a messy source code, full of repetitions and inconsistencies. In general, point-and-click environments generate ill-formed code with syntactical errors in many times. And last but not least, automatically generated pages open significantly slower than similar pages created in HTML view with hand coding.

Why not from scratch then?

Theoreticians of information technology say that if you need to build maintainable complex applications fast, you must use an Integrated Development Environment. When they say Web site builder IDE, they mean a code-generating software. While the developer points-and-clicks, drags-and-drops in a graphical user interface, the software writes the HTML code. IDEs may be useful for beginners, amateur developers, and for designers, who want to show off with fast partial results. These tools meet the expectations of having a short learning curve for simple projects similar to their samples and for people with minimal programming background. Unfortunately no IDE can completely exclude the manual writing of some elements from the development process. At the end of this lecture I give an elementary formatting task that one cannot solve with a site builder IDE by pointing-and-clicking. One must write the code manually. Problems come up at the high end, too. An IDE's half-solution becomes non-maintainable because during the short learning period the "developer" did not learn what happens behind the scene. Sooner or later a real expert has to look at the code anyway and has to rewrite other's code, namely, the code generated by a senseless software while a trained monkey pressed the mouse buttons in the right sequence. Names and id's don't mean anything in that code and there is no person to ask how and why a segment was created. Decoding, understanding and maintaining the generated code become more and more difficult as the code grows.

Average projects that require about a quarter-year effort of a qualified developer can usually be completed without an IDE at least as fast as with an IDE. If you understand what you are doing, then the handcrafted application will be at least as maintainable as if it was built with an IDE. Of course, one can build terrible solutions manually, as well.

Writing code from scratch is not easy but rewarding. One should focus on creating elements that are generally applicable, not just for that particular case. WYSIWYG editors don't know what other pages and similar elements are in your mind. They are programmed to create that specific page or page element only. In code view, on the other hand, you can create general reusable elements that build Web applications, rather than just solve the actual problem. Your tools are not proprietary, as opposed to that of the IDE's. You can modify them easily and reuse them freely. The superiority of this approach in education is also without doubt, provided that the goal of education is to produce thinking individuals, not trained monkeys. Nevertheless, the products created from scratch will run much faster than the ones created in the WYSIWYG view of an IDE.

Once you built a few small sites and applications of your own elements, you would hang on to your toolset, independently from others' heavy-duty machinery. You may switch to an IDE later but it does not work in the opposite direction. If you never tried the craftsmen's method, you cannot fix the problems that come from the automation. And they will come, no matter what the IDE maker says. That is why I want to start with writing the very basic structure of an HTML document manually. Trust me. It won't take long and it won't hurt.

The basic template

As scientific disciplines must proceed through their evolution, including some dead-end experiments, the history of Web design and development has its own natural path as well. I whish that in our territory theory of evolution was not applicable, and the smartest, not the strongest survive. Most point-and-click Web design software builds the layout of a page by creating cells of rows of tables. What happens if a page element needs internal rearrangement? The software creates cells of rows of tables inside the cell that contains the element. The page will be populated with complicated formatting objects that should be separated from the content.

Even though I have not used point-and-click or drag-and-drop interfaces of development environments for serious projects, I too fell for the easy way of designing layouts. My standard template of a Web page always had a table and it looked something like this:

```
<html>
   <head>
        <title> Title comes here</title>
   </head>
   <body>
     <form>
       <table>
         <tr>
           <td> Content element 1 comes here
           </td>
           <td> Content element 2 comes here
           </td>
         </tr>
         <tr>
           <td> Content element 3 comes here
           </td>
           <td> Content element 4 comes here
           </td>
         </tr>
       </table>
     </form>
  </body>
</html>
```

Code 1. An old HTML template using a table for arranging the content.

I placed a form in the top level of the body, just to be capable to use form elements, like input boxes, form attributes and methods throughout the page. Then I put a table in the form to manage the layout. Lately I read publications that convinced me not to format pages with tables. Two of the best were published at about the same time, in May 2003. One is **Designing with Web Standards** by Jeffrey Zeldman. The other one is Dan Shafer's **HTML Utopia: Designing Without Tables Using CSS**. If you are as cheapo as I am and want to check out free chapters first, you can find the respective links at the end of this book. And if you are as reasonable as I am, sooner or later you will be convinced and buy at least one of them. They explain well that tables are for displaying rows and columns of related data, not for arranging objects on a page in general. I agree that there are other tags, like `span` and `div` that select sections of a page and are more suitable for formatting page sections. If we use tables as means for arranging unrelated elements, we violate one of the ***Web Developer's Ten Commandments,*** which says: *"Thou shalt not mix content with presentation."* One who understands why the `font` and `center` tags have been deprecated can understand that I misused the `table` tag here.

Browsers render tables much slower than other elements. Most HTML elements are interpreted and displayed sequentially, from top to bottom of the source code. However, browsers cannot display the first row of a table correctly until they read the last row. So, they go through the tables twice, which doubles the time of presentation.

Another reason I had to get rid of the table that surrounded the content was that I frequently created sequences of similar page elements. HTML does not facilitate looping, and I had to write `td` and `tr` tags in the page with scripts. These tags were missing from the view of HTML validators. I wasted time on searching non-existent errors in syntactically correct and well-running pages because validators skipped the tags written by scripts. I could have replaced the commandment about separation of form and content with one that says: *"If it ain't broke, don't fix it!"* but I did not. I am not that kind of reformer...

I caught another children's illness of Web development. I structured my pages in a frameset. Framesets have an incontestable advantage. You can put constant page elements in frames and update only one or a couple of frames that change content while navigating through the site. If your Web site displays many constant elements, such as menus, header and footer, a side section with table of content, you may save significant download time with placing those elements in steady frames. On the other hand, frames have their problems, too. A frameset's URL that you see in the address bar does not necessarily correspond with the content of the page you see. The displayed address is usually the address of the opening state of the frameset. It does not change when the content of a frame changes. I have not completely given up using framesets. I found a solution to overcome the above problem of unmatching addresses and contents. The article ***Got Lost On the Web?*** helps you decide to keep or abandon your frames.

To format a page, we should use Cascading Style Sheets at the first place. In the next template I included some simple styling. I also replaced the `form` tag with `div` because I don't use form elements in this page:

```
<html>
 <head>
    <title> Title comes here
    </title>
    <style>
      body {text-align:center; background:#eee}
     .content {background:white; width=680px; text-
align:left}
    </style>
  </head>
  <body>
    <div class="content">
      Content elements come here<br>
     in an arrangement you like.
    </div>
  </body>
</html>
```

Code 2.a An HTML template using a division for arranging the content.

Even a simple page like this looks substantially differently in different browsers. IE 6 centers the content, FF does not. One way to make it look the same is that we declare its document type. Once we put a standard XHTML doctype in the first line, we also take some obligations. We need to add a namespace to the html tag, a type attribute to the style tag, and we need to close unclosed `<br>` tags as `<br />`:

```
<!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">
    <head>
        <title> Title comes here
        </title>
        <style type="text/css">
            body {text-align:center; background:#eee}
            .content {
            background:white; width=680px; text-
align:left}
        </style>
    </head>
    <body>
        <div class="content">
            Content elements come here<br />
            in an arrangement you like.
        </div>
    </body>
</html>
```

Code 2.b An XHTML template using a division for arranging the content.

Many Web sites condense their pages on the left side of the window. In a high-resolution screen they look as if they are close to tip over and sink to the left. However, it is not easy to evenly distribute content because the user's screen's width can range from 640 to 1600 pixels and windows can be resized. That's why I prefer centering the content. The `text-align` property of the body does not mean here what it says. It aligns the enclosed object, not just the text, to center. Unfortunately and fortunately it only works in Internet Explorer. Unfortunately, because the unsolved problem inspired me to find a general solution, and fortunately because I could come up with a cross-browser solution that does what it says. I don't have to be aware that the solution becomes obsolete when Microsoft corrects its semantic error in the next version of Internet Explorer. (Note: I was right in the first edition of this book. Since then IE 7 came out and the solution works as expected.)

I like to display the content in an area of fixed width and text left-aligned. That is why I set the form's width to 680 pixels, wide enough but not exceeding the width of the still used 800 by 600-pixel monitors. I reset the `text-align` property of the `div` that surrounds the content to the left.

To really separate content and presentation, we need to create a style sheet file and link it to the page. This single style sheet file can later be linked to all pages that we want look similarly. Also, in an optimal setup the once loaded file is cashed in memory and the browser doesn't have to download it again and again when it opens other pages that use the same CSS file. Here are the separated HTML template and the CSS file.

```
<html>
 <head>
   <title> Title comes here
   </title>
   <link href="scripts/basic.css" type="text/css"
rel="stylesheet">
 </head>
 <body>
    <div class="content">
      Content elements come here
    </div>
 </body>
</html>
```

Code 3. An HTML template containing link to the style sheet file.

```
 /* basic.css*/
body { text-align:center; background:#eee}
.content {background:white; width:680px;
text-align:left}
```

Code 4. The basic.css style sheet file.

Code 3 is a modification of the quirky Code 2.a, which displays differently in IE 6. Returning to the problem of centering the content, a cross-browser solution exists, which works in both quirks mode and standard mode. The critical part of the style is highlighted in the following code. The idea is to move the left edge of the content to the center and then back with half of its width. As opposed to many other browser-dependant solutions, this one does not run a script to query the width of the window. Everything is set relative to the actual window size and everything is absolutely simple. The CSS file contains my additional formatting preferences, too. This file, about 1k size, can be the overhead of a whole site. Take its items as a shopping list, not as a book of law. It takes care of the most important tags and classes. You may change their properties according to your taste.

```
/* basic.css*/
body {margin:0; padding:0; border:0;
  background:#eee; font-size:12pt;
  font-family:Arial,Geneva,Sans-Serif}
.content { width:680px; position:relative;
  left:50%; margin-left:-340px;
  margin:0 auto; text-align:left;
  padding:0 25px; border:0; background-color:white;
  background-repeat: no-repeat;}
h1 {text-align:center; font-size:22pt}
h2 {text-align:left; font-size:17pt; margin:0;
    line-height:100%}
.float {float:left; text-align:center;
  width:65px; padding:8px 5px}
.code  {font-family:Monaco,
  "Curier New",monospace; font-size:11pt}
.code1 {font-family:Monaco,
  "Curier New",monospace}
.hilite{background:yellow}
a{text-decoration: none;margin:0; padding:0;
  border:0}
a:link{color:blue; font-weight:bold}
a:active{color:red}
a:visited{color:#909; /*lila */}
a:hover{color:black;background:silver}
```

Code 5. The final version of basic.css.

The rest is needlework. Just make a dozen copies of the HTML template, give them titles, fill them up with contents and you developed a platform independent, resolution-independent, cross-browser Web site. More detailed, XHTML compliant templates are shown in a subsequent article, in ***The Ultimate XHTML Template***.

Summary

Starting and completing Web design projects from scratch is simple. As opposed to working in a WYSIWYG environment, your code remains maintainable and loads much faster than the one created with a point-and-click, drag-and-drop editor.

? Practice questions and problems

1. Have you been using WYSIWYG Web design software? Then here is a simple task for you. Paste the content of the first twenty lines of the http://www.scriptwell.net/fastweb/scratch.html page in your Web development tool. No, not the source code, only the unformatted text. Reformat the text with the point-and-click way to look like the beginning of my page. When you are happy with the result, check the following:

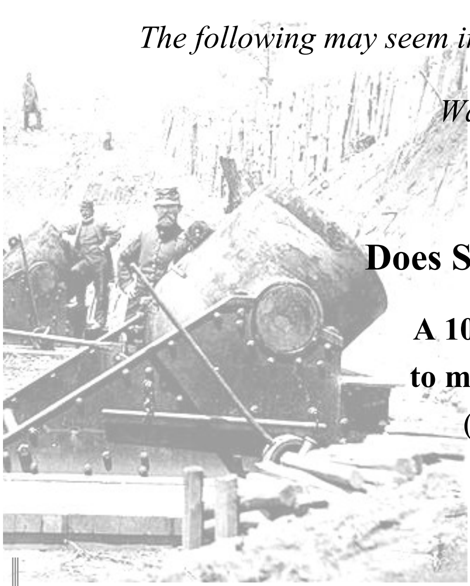
    - Resize the two comparable windows to full size, to medium and to smaller than the text width. Do they still look alike? If not, go back to the design desk until they do in all three window sizes.
    - Open my page and your page with two other browsers. If they look the same, let me know what development environment are you using and I take my hat off. If they look different, try to modify the page in WYSIWYG mode until they look the same in all the three browsers as mine. After you are

done, reconsider the label "fast" of your development tool.

- How large is the formatting overhead of your file? Does your tool duplicate this overhead or a significant part of this overhead in a similar page that you create with your tool? Don't you think that it is too much overhead for that short, one thousand character text?
- Validate your page with an HTML validator, for example, with the W3C Markup Validation Service. If it is proved to be an XHTML strict document, I give my highest respect to the makers of your development software. If it is a valid HTML 4.0 Transitional document, I still bow. If it is not valid, rethink using that Mickey Mouse toy as a development tool.

2. What are the basic advantages of creating Web pages from scratch as opposed to using WYSIWYG development software?
3. Let's say that you prefer the arrangement of pages where the content starts on the left, with a margin of 25 pixels. Would you change the HTML file of the above page? If not, what would you change?
4. Write a template that centers the content of the page vertically, too, regardless of the window size. For simplicity, the height of content of the page is expected less than 400 pixels but unknown. Isn't it a simple and basic design? Can you complete the development with your WYSIWYG tool? (I don't think so!) The solution should work in three browsers, for example in IE6, NS7/Mozilla/FF, and Opera7.

The following may seem irrational, unreasonable but educational.
Waste more – make more.



Does Size Matter?

A 10-minute cure to make it shorter
(or longer)

Intro

Call me superstitious but 10k in file size is a magic number to me. As most of us, I can read and correct small codes fast and easily. To me the trouble starts around 10k. It's OK to put 20k textual content in a page but a 10k file full of HTML tags, CSS styles and JavaScript can become unreadable. Beside perspicuity, a file under 10k can load in about a second, even via regular dial-up connection. I usually shave off 30-60% of my first working version's size. Whenever I realize that my file is too big to trace, I start to scrupulously compact, and rewrite repeating blocks into functions.

What's Wrong with Obfuscators?

I usually don't compromise readability, usability or maintainability for smaller size. I don't delete comments that make sense and I don't write several statements in one line to spare some new-line characters. Unlike some obfuscators, I would not rename a variable from sAnswer to x to save some bytes. An obfuscator program makes me keep two versions: One for reading and editing and one for uploading and running. Can you trust an obfuscator that it delivers exactly the same page as the original version? If you have a development site locally, a test site and a live production site, which ones would contain the obfuscated version? The live site is supposed to be a replica of the last successfully tested state of the test site. What would you check when the obfuscated version fails and reports an error in line 132? If a linked file has an original and a condensed version, which version would you invoke in the original and in the condensed version of the main file? Obfuscators don't like that I don't put ";" in front of "}". Would you change your coding habits to satisfy the obfuscator's special needs? I am not an obfuscator fan. A nitpicker? Maybe.

I am picky about repeating lines. It is trivial that one should replace repetitive blocks of alike JavaScript codes with calls of one function. Less obvious is that repetitive chunks of HTML can be replaced with the call of a JavaScript function that contains `document.write` commands. Means of dynamic HTML can be used in static HTML pages, too. Let it be a large part of HTML text showing twice or a small part showing more than two times in a file, I functionalize it but obfuscators don't.

I found a file, OPEN.HTML of 11k size, filled with JavaScript, well structured, well organized and probably well tested, because it is part of a Microsoft presentation. On the Windows 2000 Professional CD, you can find it in a folder called DISCOVER. It is readable, relatively concise and does not really need my personal touch. I learned a lot from it about how to animate a page with simple JavaScript instead of using ten times larger flash files or other prepackaged animation technologies.

What's Wrong with the Market Economy?

I too prefer simple but significantly shorter solution to any machine-gun-against-the-sparrow solutions that have the false advantage: A beginner who has an expensive development environment can deliver impressive effects without knowing much about computers. But this is a bigger issue in our society. In general, a trained monkey with resources is more appreciated by decision makers than talent, knowledge and hard work. You cannot get a fat government contract just because you can write fantastic applications in a short time on a low budget. "Social engineering" beats engineering. Hire a lobbyist or a salesman for a 95% share, who can convince the big buyers that they have to spend 20 millions on the solution, and you justified your modest, 5%, one million dollar income. ***Waste more make more***.

The private sector is not better either. Today's market economy is not about competition for excellence. The race for the better, more economical turned to the race of squeezing out the competitors with non-technical means, like with patenting an obvious feature, a key-stroke, to prevent the competition from using it. Differences in values or efficiency of the product have secondary importance. Efficient manipulation with false ads of a half-baked proprietary product wins over quality work. Smart marketers and lawyers make a product profitable at first place, not technical experts who know the trade in and out.

I don't categorically deny the value added by non-technical persons. In many cases issues like usability should be discussed with experts in human sciences rather than relying merely on the opinion of the application's developer. However, if the solution runs slow because of its size, in our culture the big buyers incline toward purchasing a faster and bigger computer system rather than asking for a solution that is aware of resources. ***Worst of the worse, some employers measure developers by the line they add. You must work against efficiency if you want to get paid.***

But my topic is not rationality today. I am about to make a nice little code even smaller just because my fingers are itching from a file size exceeding 10k. It is irrational, unreasonable but educational. Take it as I took, a short exercise that can lead to a larger saving in size and bandwidth in another case.

A mini case study

I decided to spend 10 minutes on shaving 10% off the already concise OPEN.HTML file without ruining its readability. I tried first what obfuscators do at first place. I cleaned extra spaces by replacing ' = ' with '=', ' + ' with '+', etc. After five minutes of mindless processing with automated *Find and Replace* the file size decreased about 500 bytes. It helped me with reading chunks of codes better but others may see the result less readable. Then I took out extra tabs. For example, all function blocks were indented with two tabs instead of one. Again, minor change in size.

Then I went for little brainier actions. I deleted useless commented lines, like `//alert (...)` left there in the testing phase. An obfuscator program would not know, which comment was trash, which was valuable.

So far the code has not been changed in a sense that its interpretation by a browser has been unaffected. The following alterations may change the parser's work but the outcome still must be the same.

If I see references to an object more than two times in a block, I usually rewrite the script using the with statement. In case of repeated long expression in addition or concatenation, I prefer the '+=' operator. For example,

```
document.all.Marketing.filters.alpha.opacity =
document.all.Marketing.filters.alpha.opacity+EndV;
document.all…
```

is not only longer but less readable than

```
with(document.all){
    Marketing.filters.alpha.opacity+=EndV;
    ...
}
```

Applying the above two principles at only two places reduced the size to 10k. Now we are talking about size reduction.

Next I just selected all HTML sections and hit Shift/Tab about five times. This action eliminated all the indentations and dropped the size to 9.7k. You may say that I definitely ruined readability. However, this was not an irreversible change. I use Visual Studio's editor that has a sometimes-annoying feature: when I switch between design view and source view, it reformats the source, regardless if I want it or not. (Actually, this is the only reason that I switch to design view.) Other advanced web page editors also have the auto-formatting feature. We can get back our indentations in no time. The file size went below 10k before my preset 10 minutes was up. I stopped tinkering the file, although I had some more ideas to make it shorter and more readable. Here are two ones concerning the `if` statement.

Conditions, my way

When I check the true/false value of a condition I never write down statements like `if(Found==true)`.

To me the shorter form `if(Found)` is more readable. Nevertheless, leaving out the comparison operation may make the code confusing in other situations. JavaScript is supposed to return false, if the object in the condition does not exist or is null. Browsers disagree on the interpretation of these cases but developers deliberately use the comparison-less shorthand

`if (object)` and `if (!object)`

regardless of testing the condition for true value or for not null value. I met highly knowledgeable developers who, of course, are aware of the difference between 0 and null. Yet, they use the above shorthand in both cases, because it happens to work in IE. Usability and maintainability are at stake when the script is being converted to a language where false is equivalent to the integer type value 0 and true is the same as not false, i.e. not 0. I also met interpreters that defined false as 0 and true as -1. Is +1 true or false in this system? Not distinguishing between false and null may work in one environment but strikes back in another. So, please compare with null if you mean that. Because of a special feature of Netscape 6, you may also need to compare with the value undefined. To keep readability in these cases, I use the wide-spread, cross-browser function to testing for null:

```
function isNull(obj) {
    return(obj==null)||(obj==undefined)
}
```

The comparison-less shorthand becomes

```
if(isNull(object))
```

and not the also correct but longer and less readable

```
if(isNull(object)==true)
```

Several techniques are available that compress the code more dramatically but they may work against readability. One is called the ***terse C syntax*** of the conditional statement. Instead of

```
if(name=="Eggenbegger") {
    x=4;
}
else {
    x=2;
}
```

you may write:

```
x=(name=="Eggenbegger")? 4: 2;
```

The latter is shorter but is it more readable? You decide. I am not that fanatic believer in the beauty of petite. But, without extreme compression actions, I pushed the file size under 10k. Mission accomplished, as the President used to say when a minor, partial and marginal task had been finished. If you are paid by the line and want to show off with a really large size of code, open your HTML document in Word, and save it as an HTML. You get result faster then ten minutes in terms of fattening your code.

And a joke

I think I made my point about the Web developer's situation in the market environment. Still, I can't resist telling a related story. The mushroom expert comes across the old woman who collects mushrooms in the wilderness. He takes a peek in her basket and anxiously says. *"Lady, these mushrooms are not edible." "Don't worry, young man"*, she says, *"I would never eat them. They're for sale."*

Summary

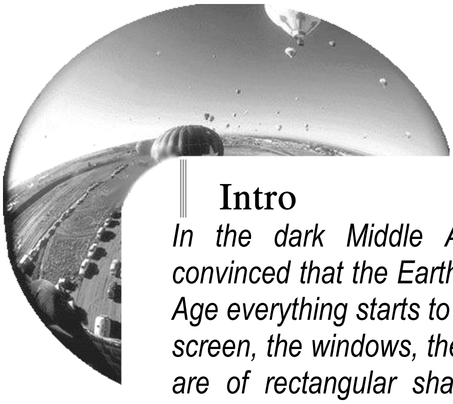
You can make it shorter or longer, depending on the demand. File size matters but extreme size reductions ruin readability. I show a few tricks that make the size of an HTML / Javascript file smaller, while improve readability. I also draw attention to a common mistake in shortening conditional statements.

? Questions

1. Where do you stand? Are you an anxious mushroom expert or one producing software for sale, not for consumption?
2. Do you think that the software market is full of half-done products?
3. Have you ever felt that your development is not done yet but financial or organizational pressure made you publish it prematurely? What did you say and what did you think then?

Now I found that the World is round...
Bee Gees

Let's Make the World Round Again



Intro

In the dark Middle Ages people had to get convinced that the Earth is round. In our Computer Age everything starts to become square again. The screen, the windows, the data sheets and tables all are of rectangular shape. One cannot create a rounded border for a table with pure HTML. You need at least a few images to make the frame of an object round. Fireworks and other Web design tools provide us with help to create non-rectangular frames. Using point-and-click software is as simple as peeling bananas but the created code will most likely be huge and clumsy. Let's avoid monkey business and go for the pure solution.
This article helps to create those rounded corners from scratch with HTML, CSS and Javascript, without Web designer software. I will present a very small addition to the page and a reusable Javascript code that will do the job. In the meantime I want to show you how and why style sheets and scripts should be introduced in your Web design practice.

If you are aware of speed and bandwidth issues, you want to avoid using large images. The following solution includes four replicas of a simple picture of a quarter-circle. Their size is about 200 bytes each. You can create one with any drawing program, e.g., with MS Paint.

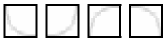


Figure 1. The four corners, bottomleft.png, bottomright.png...

If you read this in electronic format, you can save the pictures by right-clicking on them, or you can draw a new bottom left image and three reflections will create the corresponding bottomright.png, topright.png, and topleft.png images. I am also using a picture for vertical and horizontal lines. It is called graydot.png, which is nothing but a 1 x 1 pixel gray dot.

The idea is to place the frame elements in the outer cells of a table. The first and last rows contain the top and bottom parts of the borders, including the curvy corners. The first and last columns of the middle row hold the sidelines of the border and its inner three columns contain the object, in our case some text that we want to surround with the border. I considered that using five columns instead of three was important because this way the sidelines of the border would occupy only one-pixel wide cells and the surrounded object could align closer to the sidelines.

Here comes the first, table-based version of a page with a text framed by a rounded corner border:

Now that you read the table of content and some light passages of the book, you can buy the "hard stuff", as a drug dealer would say. I'd say the same in a polite manner: If you like what you have read, please buy the complete book. You will not regret it. Thank you.

```
<html>
 <head>
  <title>Rounded Corners ver. 1.</title>
 </head>
 <body><br>
  <table align="center" cellspacing="0" border="0">
   <tr><!--Row 0 presets the widths of columns-->
    <td width="1"></td>
    <td width="14"></td>
    <td></td>
    <td width="14"></td>
    <td width="1"></td>
   </tr>
   <tr valign="top"><!--Top with upper corners-->
    <td colspan="2"><img src="images/topleft.png"
    width="15" height="15" alt="/"></td>
    <td><img src="images/graydot.png" width="570"
    height="1" alt="."></td>
    <td colspan="2"><img src="images/topright.png"
    width="15" height="15" alt="\"></td>
   </tr>
   <tr><!--Middle of the frame with the sidelines-->
    <td background="images/graydot.png" width="1">
    </td>
    <td colspan="3"><!--The object placed inside
        the border-->Content comes here...</td>
    <td background="images/graydot.png"
    width="1"></td>
   </tr>
   <tr valign="bottom"><!--Bottom & lower corners-->
    <td colspan="2"><img src="images/bottomleft.png"
    width="15" height="15" alt="\"></td>
    <td><img src="images/graydot.png" width="570"
    height="1" alt=".">
    </td>
    <td colspan="2"><img src="images/bottomright.png"
    width="15" height="15" alt="/"></td>
   </tr>
  </table>
 </body>
</html>
```

Version 1. Rounded borders created with pure HTML and with 5 minimum size images.

Remarks

1. There is a 0th dummy row in the table before the first row that does not display anything but sets the widths of the cells. They could be set in the first row if there were only single-column cells without `colspan`.
2. The vertical sidelines come in fact from the default feature of repetition of the single dot as a background image. We cannot repeat the dot horizontally the same way because the cell's height is greater than 1 pixel and the repetition of the dot would fill the whole height of the cell. The length of the vertical borderlines automatically grows as the height of the row grows with the size of the center cell.
3. The length of the horizontal borderlines, presently 570, should be set at least as large as the width of the framed object. Otherwise the border will be discontinuous.

There is a big drawback of the above solution. Whenever the size of the framed object changes, the size of the horizontal borderlines should be changed at multiple places. A better solution should offer a single location where the width of the border can be set. Also, the HTML code contains the size of the corner images at several places. What if someone wants to use other corner images of different pixel size? The next version contains a simple CSS code that sets the size of the horizontal line at one place.

Let's replace the tag

```
<img src="images/graydot.png" width="570" height="1">
```

at both places with

```
<img class="horizline">
```

where `horizline` is defined as a CSS class:

```
  .horizline {width:570px; height:1px;
        background-image:url(images/graydot.png);
        background-repeat:repeat-x;}.
```

Now we can set the width of the horizontal line at one place. Similarly, the class

```
.vertline {width:1px; background-image:
url(images/graydot.png); background-repeat:repeat-y}
```

would eliminate the width setting in the tags

```
<td background="images/graydot.png" width="1">
```

at two places by rewriting the above tags in the form of

```
<td class="vertline">.
```

Unfortunately, we must handle the sidelines differently as the horizontal lines: Their length (height) is difficult to tell in advance. Although the described class works well with IE, Netscape disregards the

```
background-repeat:repeat-y
```

property without the explicit specification of the object's height, so, we will draw the vertical lines the old way.

However, to converge the `width=15 height=15` settings in one place, we can define a corner class as

```
.corner {width:15px; height:15px}
```

and replace the tags like

```
<img src="images/topleft.png" width="15" height="15">
```

with

```
<img src="images/topleft.png" class="corner">.
```

The new version looks like this:

```
<html>
 <head>
  <title>Rounded Corners ver. 2.</title>
 <style type="text/css">
   .horizline {width:570px; height:1px;
      background-image:url(images/graydot.png);
      background-repeat:repeat-x;}
   .corner {width:15px; height:15px;}
 </style>
 </head>
 <body><br>
  <table align="center" cellspacing="0" border="0">
     <td width="1"></td>
     <td width="14"></td>
     <td></td>
     <td width="14"></td>
     <td width="1"></td>
    </tr>
   <tr valign="top">
     <td colspan="2"><img src="images/topleft.png"
     class="corner">
     </td>
     <td><img class="horizline"></td>
     <td colspan="2"><img src="images/topright.png"
     class="corner">
     </td>
    </tr>
   <tr><!--Middle of the frame -->
     <td background="images/graydot.png"
     width="1"></td>
     <td colspan="3">Object inside round border</td>
     <td background="images/graydot.png"
     width="1"></td>
    </tr>
    <tr valign="bottom">
     <td colspan="2"><img src="images/bottomleft.png"
     class="corner">
     </td>
     <td><img class="horizline"></td>
     <td colspan="2"><img
      src="images/bottomright.png" class="corner">
     </td>
   </tr>
  </table>
 </body>
  /html>
```

Version 2. Rounded borders created with HTML and CSS.

Remarks

Although the repetition of background images is default, setting the repetition in the style is a good idea. Don’t forget that browser defaults can be set in several levels.

In the previous version the horizontal lines were in the foreground and the vertical lines in the background. (The $100 question: Why was this setting necessary there?) With the class definitions both lines are in the background.

Now almost all the size parameters are moved from the disperse parts of the body to the style segment of the head, except from the notorious row 0. Let's move those parameters to the head, too, by developing a little Javascript that will write the HTML code of row 0. The first 20-some lines of the code are as follows.

```
  <html>
  <head>
    <title>Rounded Corners ver. 3.</title>
    <style type="text/css">
     .horizline {width:570px; height:1px;
        background-image:url(images/graydot.png);
        background-repeat:repeat-x;}
     .corner {width:15px; height:15px;}
    </style>
    <script>
      var curveWidth = 15;
      function dispRow0() {
        var temp = curveWidth-1;
        document.write('<tr><td width="1"></td>');
        document.write('<td
        width="'+temp.toString()+'"></td>');
        document.write('<td></td>');
        document.write('<td
        width="'+temp.toString()+'"></td>');
        document.write('<td width="1"></td></tr>');
      } //end function dispRow0()
    </script>
 </head>
 <body><br>
 <table align="center" cellspacing="0" border="0">
      <script>dispRow0();</script>
      <tr valign="top">
… The rest of the code did not change …
```

Version 3. Rounded borders created with HTML, CSS and Javascript.

Since version 1 the source code grew about 6 lines but it became simpler. Wouldn't it be nice to surround an object of a Web page with a round border just by adding a few lines to the Javascript? Then we can create as many framed objects in the Web site as many we wish, without extra overhead.

Once we started to write Javascript, we became close to an elegant version. Why don't we consolidate the rest of the border drawing into the Javascript and put the script in a separate file? Let's "can" the border drawing in a Javascript file that the Web designer does not even have to read and understand. This way one can add round borders by adding only a few lines to the original HTML source code.

Here is the "canned" version of our project:

```
<html>
 <head>
   <title>Rounded Corners ver. 4.</title>
   <style type="text/css">
     .horizline {width:570px; height:1px;
     background-image: url(images/graydot.png);
     background-repeat:repeat-x;}
     .corner {width:15px; height:15px;}
   </style>
     <script>
       var curveWidth = 15;
     </script>
     <script src="scripts/roundborder.js"></script>
</head>
<body><br>
      <script>dispRow0();dispTopAndLeft();</script>
Here comes the object placed inside the round border
     <script>dispRightAndBottom();</script>
 </body>
</html>
```

Version 4. Rounded borders created with HTML, CSS and Javascript.

The setting of size parameters is concentrated in the style and script sections of the head. The body contains only two extra lines. The above solution works with the five picture files in the images subfolder and the following `roundborder.js` script in the scripts subfolder:

```
function dispRow0() {
   var temp = curveWidth-1;
   document.write('<table align="center"
     cellspacing="0" border="0">');
   document.write('<tr><td width="1"></td>');
   document.write('<td
width="'+temp.toString()+'"></td>');
   document.write('<td></td>');
   document.write('<td
width="'+temp.toString()+'"></td>');
   document.write('<td width="1"></td></tr>');
} //end function dispRow0()
function dispTopAndLeft() {
   document.write('<tr valign="top">');
   document.write(' <td colspan="2"><img
src="images/topleft.png" class="corner"></td>');
   document.write(' <td><img
class="horizline"></td>');
   document.write(' <td colspan="2"><img
src="images/topright.png" class="corner"></td>');
   document.write('</tr>');
   document.write('<tr>');
   document.write('<td
background="images/graydot.png"
width="1"></td>');
   document.write(' <td colspan="3"
align="left">');
} //end function dispTop()
function dispRightAndBottom() {
   document.write('</td>');
   document.write('<td
background="images/graydot.png"
width="1"></td>');
   document.write('</tr>');
   document.write('<tr valign="bottom">');
   document.write('<td colspan="2"><img
src="images/bottomleft.png"
class="corner"></td>');
   document.write('<td><img
class="horizline"></td>');
   document.write('<td colspan="2"><img
src="images/bottomright.png"
class="corner"></td>');
   document.write('</tr></table>');
} //end function dispRightAndBottom()
```

Addition to Version 4. The script `roundborder.js` contains a size variable as a single parameter that can be set outside the code.

Are we done yet? Well, almost. The setting of the size parameters is still scattered: the curve size shows at three places and the length of the horizontal line at a fourth place. Why don't we move them to one place by letting a Javascript function create the style section, too? Here is the latest version. The creation of the latest and greatest is waiting for you. See the homework at the end of the article.

```
<html>
 <head>
   <title>Rounded Corners ver. 5.</title>
   <script src="scripts/roundborder.js"
type="text/Javascript">
   </script>
   <script>
       var curveWidth = 15;
       var horizWidth = 700;
       scriptRoundStyle();
   </script>
 </head>
 <body><br />
   <script>dispRow0();dispTopAndLeft();</script>
      Object to place inside the round border
   <script>dispRightAndBottom();</script>
 </body>
</html>
```

Version 5. The addition of the highlighted lines make an object bordered with a rounded frame.

To make the above solution work, we need to add the `scriptRoundStyle()` function to our Javascript:

```
function scriptRoundStyle() {
   document.write('<style type="text/css">');
   document.write('.horizline {width:' +
horizWidth.toString() + 'px; height: 1px;');
document.write(' background-image:
url(images/graydot.png);
   background-repeat:repeat-x;}');
document.write(' .corner
{width:'+curveWidth.toString()+'px;
height:'+curveWidth.toString()+'px;}');
   document.write('</style>');
} //end function scriptRoundStyle()
```

This function should be added to the script roundborder.js.

The ultimate solution

We learned some techniques to make a page more compact and more maintainable. Now that we know the steps and reached a solution that works in HTML quirks mode, let's throw it out, and restart the development of a standards-compliant, XHTML, tableless solution by using the prior experience.

```
<html>
<head>
    <title>Tableless Rounded Corners ver. 1A.</title>
    <link href="scripts/roundborder.css"
rel="stylesheet" />
</head>

<body>
  <div class="roundBox">
    <div class="roundedge">
<!--Top of the frame with the upper corners-->
      <img src="images/topleft.png"
       style="vertical-align:text-top"
       alt="/" /><img class="horizline"
       src="images/graydot.png"
       style="vertical-align:text-top;" alt="." />
    </div>
    <div style="border-left:solid 1px #ccc; border-
right:solid 1px #ccc;">
            Content comes here...
    </div>
    <div class="roundedge"
        style="background:url(images/bottomright.png)
no-repeat top right">
        <img src="images/bottomleft.png"
            style="vertical-align:top" alt="/" /><img
class="horizline"
            src="images/graydot.png"
            style="vertical-align:bottom" alt="." />

            Content comes here...
    </div>
    <div class="roundedge"
        style="background:url(images/bottomright.png)
no-repeat top right">
        <img src="images/bottomleft.png"
            style="vertical-align:top" alt="/" /><img
class="horizline"
            src="images/graydot.png"
            style="vertical-align:bottom" alt="." />
      </div>
  </div><!--end roundBox-->
</body>
</html>
```

The Ultimate Version.

The highlighted lines should be added before and after the content that you want to surround with a rounded frame. It is important that the consecutive image tags be written back-to-back. If we write them in separate lines an extra space will show between the round frame elements.

Where are the top-right and bottom-left corners? You may ask. Those and the horizontal lines are defined in the linked roundborder.css:

```
.roundBox { width:730px; font-size:14px;
    background: url(images/topright.png) no-
repeat top right
}
.roundBox div, img {padding:0; margin:0;
border:0;}
.roundedge {width:100%; height:15px; line-
height:100%;/* for Strict*/}
.topleft {vertical-align:top}
.horizline {width:700px; height:1px}
```

The linked roundborder.css.

Where to go now? We may consolidate the dozen lines that display the frame into two JavaScript functions, topFrame() and bottomFrame(), similarly to dispTopAndLeft() and dispRightAndBottom() in version 5. I leave this job to you. You can perfect the project in several other ways, like making the line width of the border adjustable, or, making the whole frame a clickable button. I am looking forward to see your ideas.

Summary

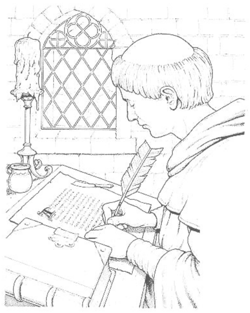
Screen objects, like windows, frames, and buttons are usually of rectangular shape. Do they poke your eyes? It's not too difficult to round the objects' corners.

In five steps from pure HTML via CSS and Javascript to a style-writing script, I'll show you the creation process of a reusable code that frames an object with rounded borders.

Problems

1. The ultimate solution can use one Javascript function call before and one after the object that we want to frame. Write those two functions and replace the two highlighted code blocks with the two function calls.
2. If you can't be sure that your users enable Javascript, you may write one line before and one after the object that include the necessary html lines. Replace the two highlighted code blocks with two single lines that include the proper HTML in the page on the server side. Create the two files to include. What else is necessary to make the server-side include work? For hints google the words html file include.

We want those nice letters, too,
but we don't have the time.



The Initial Initiative

Intro

Sure, we like those decorative initial letters, and no, we don't want to spend hours to individually paint them. Once you found your favorite font, size, color and arrangement, why can't you just throw a new fancy letter, say, a shady G in your page by calling a function, say shady(G)? Again, most graphical design software is capable to create fancy pictures of those letters but we want a simple, fast and bandwidth-saving solution. Forget the pictures and their thousands of pixels to transfer. Let's go for the pure solution "canned" in CSS and Javascript as we did previously.

The technology is out there, you just have to pick it up. Setting the initial letter of a paragraph is carried out by putting the initial letter in a span of class d3, named after 3-dimensional. The span should include two other spans that hold the front and background letters. The HTML code of this paragraph should start like this:

```
<span class="d3">
<span class="front">T</span>
<span class="back">T</span></span>he technology is
out there...
```

A letter is a letter stored in one byte and its shadow is the same letter, shifted and grayed a little. The two spans have to be in one line, otherwise a white space gets between them. The front and back classes are defined in a style sheet as

```
.d3 {float:left; font:bold 34px Georgia;
       position:relative; line-height:107%}
.front{position:relative; top:-10px; left:2px;
       color:black; z-index:2}
.back {position:absolute; top: -8px; left:0px;
       color: gray; z-index:1}
```

The above definitions set the colors of the front and shadow letters black and gray, and the relative shifts of the shadow down by 2 pixels, and left by 2 pixels. (Remember, measures start from the top left corner. That's why top and left mean down and right.) Naturally, we should play with the shifting parameters ***after*** we set the font type and size. You need to find a balanced set that works for the narrowest letter I and the widest letter M. Once you found your favorite arrangement, stay with it and don't change it inside a site or application.

The solution already works but it is not as elegant as I promised in the intro. The HTML code of a better version should be as simple as

```
<script>shady('T')</script>he technology is out
there…
```

The rest should be hidden in the JavaScript code:

```
<script type="text/javascript">
  function shady(letter) {
    document.write('<span class="d3"><span
class="front">'+letter+'</span>');
    document.write('<span
class="back">'+letter+'</span></span>');
  }
</script>
```

To make the scripted version work the same way as the original version, one should copy the function exactly. A minor reformatting, addition of white spaces like a new line or a tab would ruin the composition.

Playing with the three classes in the style definitions gives you an opportunity to live up your artistic talent and to design initial letters the way you want. The setting of the background, size, and multiple letter layers yield endless possibilities.

Now that we arrived to a simple solution, I give you an even simpler one. The W3C has a recommendation for a CSS pseudo-element called first-letter. For example, if you want to use a special style for the first letter of a paragraph, or of all paragraphs, you can start the paragraph with `<p:first-letter>` and close it with `</p:first-letter>`. The style definition of the tag should look something like

```
p:first-letter {text-weight:bold; text-shadow: silver
2px 2px 0}.
```

You can look for the description of the text-shadow property's parameter on the W3C site www.w3.org. The main reason that I don't go into details is that no major browser implemented this element yet. But watch out, my nice solution for the fancy initial letter may become obsolete soon. Until then, use it if you like it and improve it if you don't.

Summary

Graphically composed texts are nice but they show up much slower than literal texts. A non-graphical technology is suggested for creating attractive letters with dropped shadow. This solution saves design time as well as download time of a page. With half a kilobyte of code you can cover all alphabets, let them be Latin, Greek or Cyrillic.

Practice questions and Problems

1. You may want to display longer than one character texts with dropping shadow. Would the shady function work without change if one gives a longer than one character literal constant as parameter to it? If yes, why; if not, why not?

2. You may want to make the decorative letter drop the shadow to its right. You increase the font-size in class d3 and realize that the font suppresses the next letter. Would an extra space after the last </span> fix the problem? If not, then what would?
3. In the above solution the principle of separation of content, structure and style prevails. The content (the letter) emerges in the HTML document. Its style and the structure are described separately in a CSS and in a JavaScript function. In some cases, passage between the separated elements is necessary. Wouldn't it be nice to set the font size and the offset of the shadow in the HTML file when calling the JavaScript function? Can you rewrite the function that its call shady('G', 300, 5) would generate a letter G of 300% size, with a shadow of a 5-pixel offset? The solution must work the same way as the original one when the additional two parameters are omitted in the function call.

Do you think the colonel goes to chase the chicken
after you order the spicy thighs at KFC?

No! At the time of order the bird is already there.
With skins and bones, but dead and breaded.

Info On The Fly

Intro

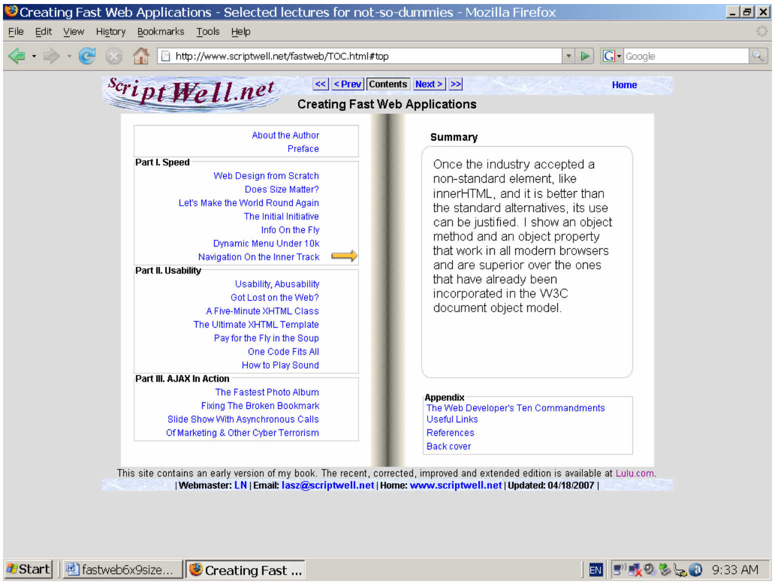
I have always been impressed when a system serves the information immediately as I need it. Satisfactory speed in the fast food industry is more common than in the information industry. Maybe because no speed is fast enough in the latter. Some technologies drill down in the pile of data while the inquirer composes the request. Others use brute force and serve everything in advance before one may specify one's needs. To serve the info fast, it has to be prepared and close to the user at the time of request. Like the breaded bird in KFC's kitchen.

A traditional way of finding a specific item is to show a containing "box" where the specific item might be stored and show the selection of items in that box. In a book one can check out the titles of chapters then the section titles inside the chosen chapter. On the Web, we click one of the visible tabs, buttons, menu items or links and hope that the click will lead us to the place we want to go. It would be nice to look behind the link before clicking, to see what we can expect.

Before I get to the point, let me talk about some historical moments of the browsers' evolution. Unlike today, early browsers did not behave similarly at all. Since both Netscape and Internet Explorer had their attractive features, writing cross-browser codes was a necessity. And a nightmare. Significant parts of the used Javascript codes dealt with finding out what the type, operating system and version of the client's browser were. For years mostly three major species must have been recognized and supported: Netscape version 4, Internet Explorer 5 and 6, and Netscape version 6. To me, dealing with the first one required the most effort. As Netscape skipped version number 5, it already has its version 7. Although Netscape 4.0 has been the pioneer of dynamic HTML and layer setting, people should stop driving that ancient vehicle. It is not only outdated but also full of bugs. To support it is as obsolete as to support a pulse tone telephone. Not having a ten-dollar touch-tone phone is ridiculous. (Imitating that you don't have one may be efficient because sometimes this is the only way to get a live representative on the other end of the line.) Not having a newer browser for free is unforgivable.

Let's return to the original topic. We need an HTML object, the content of which can be changed fast, according to the user's request. We also need an event that can handle the object's content dynamically. The object I chose is the inline frame, `iframe` in short. The event we will utilize is the `onmouseover` event. The latter is what it says. It happens when the mouse gets over an image or over a linked object. With its help, before the user clicks on a link, we can show what to expect after the click. The info will be displayed in an inline frame immediately as the mouse gets over the link.

You could see an example on the *Table of Contents* page www.scriptwell.net/fastweb/TOC.html :


Creating Fast Web Applications - Selected lectures for not-so-dummies - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.scriptwell.net/fastweb/TOC.html#top
Google
ScriptWell.net
<< < Prev Contents Next > >>
Home
Start
fastweb6x9size...
Creating Fast ...
EN
9:33 AM

Creating Fast Web Applications

About the Author
Preface

Part I. Speed
Web Design from Scratch
Does Size Matter?
Let's Make the World Round Again
The Initial Initiative
Info On the Fly
Dynamic Menu Under 10k
Navigation On the Inner Track

Part II. Usability
Usability, Abusability
Got Lost on the Web?
A Five-Minute XHTML Class
The Ultimate XHTML Template
Pay for the Fly in the Soup
One Code Fits All
How to Play Sound

Part III. AJAX In Action
The Fastest Photo Album
Fixing The Broken Bookmark
Slide Show With Asynchronous Calls
Of Marketing & Other Cyber Terrorism

Summary

Once the industry accepted a non-standard element, like innerHTML, and it is better than the standard alternatives, its use can be justified. I show an object method and an object property that work in all modern browsers and are superior over the ones that have already been incorporated in the W3C document object model.

Appendix
The Web Developer's Ten Commandments
Useful Links
References
Back cover

This site contains an early version of my book. The recent, corrected, improved and extended edition is available at Lulu.com.
| Webmaster: LN | Email: lasz@scriptwell.net | Home: www.scriptwell.net | Updated: 04/18/2007 |

Over there, the summary of a particular article showed in a frame in the moment the mouse got over the article's title. No search in a database, no download time. The `iframe` container made the prompt response possible. During the download of the page, another HTML file that contained all the summaries loaded into the frame. The `onmouseover` event does not have to initiate any data transfer, just the vertical positioning of the summaries inside the frame. How many summaries can we retrieve promptly this way? As many fit in an HTML document. The practical limit is the list of articles on the parent page, the summaries of which should be displayed. In an Intranet, a reasonable page size can be up to 50k. The recent year's summaries of a publishing company, or the company's phone directory can fit in this size for sure. All in all, this technique is excellent for providing a small-to-middle size quick reference.

Let's see how the page is set up. It has a table with one row and two cells. The right cell contains the `iframe`, the left cell contains the list of titles of the articles. For simplicity, I took out the codes that draw the round frame around the summaries and simplified the articles' file names:

```
<table align="center" cellspacing="0" border="0">
 <tr><td width="330px">
  <a onmouseover ="moveTextTo(1)"
href="article1.html">
  Let's Make the World Round Again
<img class="arrow" alt="<" id="arrow1"
src="images/ARROW.gif"></a>

 <a onmouseover ="moveTextTo(2)"
href="article2.html">
  Dynamic Menu Under 10k
<img class="arrow" alt="<" id="arrow2"
src="images/ARROW.gif"></a>

 <a onmouseover ="moveTextTo(3)" href="article3.htm">
  Got Lost on the Web?
<img class="arrow" alt="<" id="arrow3"
src="images/ARROW.gif"></a>

 <a onmouseover ="moveTextTo(4)" href="article4.htm">
 The Initial Initiative
<img class="arrow" alt="<" id="arrow4"
src="images/ARROW.gif"></a>

 <a onmouseover ="moveTextTo(5)" href="article5.htm">
  Info On the Fly</a><br>
<img class="arrow" alt="<" id="arrow5"
src="images/ARROW.gif"></a>
 </td>
 <td>
 <b>Summary:</b>
  <iframe style="left:6px; position:relative"
marginwidth="0" frameborder="0" scrolling="no"
id="sum" name='sum' src="summaries.html"
width="300px" height="140px">
  </iframe>
  </td>
</tr></table>
```

Code 1. The HTML code of the table that displays the list of articles and the summaries.

The above code refers to the arrow class, defined in a style section as

```
.arrow{width:52px;height:20px; vertical-align:bottom;
border:0; visibility:hidden;}
```

The moveTextTo function is responsible for moving the arrow image, and scrolling the text of summaries inside the frame to the appropriate entry:

```
function moveTextTo(block){
 with (document) {
  for (var i=1;i<6;i++){//switch all images to
invisible
getElementById('arrow'+i).style.visibility='hidden';
    }
getElementById('sum').src='summaries.htm#bl' + block;
getElementById('arrow'+block).style.visibility=
'visible';
  }
}
```

Code 2. The Javascript function turns all images invisible, scrolls the text of summaries in the frame to the appropriate anchor and switches the selected arrow back to visible.

Thanks to the `getElementByID` function that works in Internet Explorer 4.0 and above, in Netscape 6.2 and above and in Opera 4.0 and above, the solution covers 99% of present Internet users. Unfortunately Netscape 4 does not understand `iframe`, although it knows a similar container, `ilayer`. With `ilayer`, a similar effect could have been created as with `iframe`. However, Netscape 4 has other deviances. For example, it does not support the `height` style element, it does not render the `border` element well and more importantly, it has a terrible behavior at manual resize of the browser's window. In a nutshell, these are the reasons why one should give up on Netscape version 4. I could write pages being compatible with NS 4. I just don't want to. Are you still a Netscape version 4 rider? Go download version 6 or 7! (And buy a touch-tone phone for ten bucks!)

Summary

A technique is shown for displaying info promptly at the time the mouse hovers over a Web object. If the object represents a link the user can decide from the info if he/she really wants to click on the object. The necessary code is smaller and faster than that of other pop-up solutions.

? Practice questions and problems

1. You are assigned to build an online phone book of a company. What would you do differently if the number of employees is one thousand or if it is ten thousands?
2. What are the basic differences between the sketched technology and a database application that uses form input?
3. Did you notice the repetitions in code 1? Good. Time to put the table's creation in a Javascript loop. If you read the previous articles, you don't need more help. OB Send me the solution as a non-reusable assignment of an array of titles and a reusable function, the size of which is not larger than 300 characters.
4. The solution uses table cells for arranging the page elements. Rewrite the page using `div` tags instead of table cells.

Talk is cheap. So is storage space.
Then why bother? I tell you why.
('Cause talk is cheap.)

Dynamic Menu Under 10k

Intro

Talk is cheap. So is storage space. You can easily find a provider who is willing to host your Web site for free, if your site is smaller than 50 Megabytes. How much is 50 Megs? For comparison, the total text that an average person reads during a lifetime would easily fit in 50 Megabytes. Dealing with pictures is a different story. One good quality photo may take as much storage space as the full text content of a book.

Still, one can build a nice and graphically rich site within the free size limits. The problem starts when you happen to be lucky and people start to visit your site. After a certain amount of traffic the provider will ask you to pay for the bandwidth and the free site is not free anymore. **Keeping the site small keeps the necessary bandwidth small and you from becoming a paying customer.** OK, you may say. On the other hand, if you don't show storage-extensive graphics, your site stays free of charge because it is so unattractive that nobody wants to see it. Right? Wrong. **Your visitor's first impression greatly depends on the loading time of your opening page.**

Many pages on the Web take much more storage than they should. Let's take a simple menu bar of four selection buttons created graphically. A button may have three versions: one for normal, one for pressed state and a third for the state when the mouse is over the button. A simple graphical button may have the size of 2k at least. So, the four buttons with three versions easily occupy 24k. With a regular modem connection, loading 24k takes about 5 seconds. 5 seconds of your visitor's time and of your bandwidth.

There is a good old programming principle that is forgotten by many Web designers: Repetition should be managed with a loop rather than repeating a chunk of code. The blame is not necessarily on the developer. Pure HTML is not capable to manage loops. But Javascript is. Displaying a bar of four dynamic buttons should not require four times as many graphics and four times as big code. Showing three different versions of a button may **add one small image of the altering part only**. In the following I show you a solution for the above menu bar problem. It will display graphical buttons with the above three states. The number of selection buttons can be arbitrary and the required amount of transfer stays below 10k, regardless of the number of buttons. **Adding a new button would add a few bytes, but not kilobytes to the transferable file.**

Let's create an opening page with two frames. The top frame will contain the menu bar and the selected page will appear in the bottom frame.

```
<html>
 <head>
  <title>Home frame</title>
 </head>
 <frameset rows="90,*" border="0" framespacing="0"
frameborder="no">
  <frame src="top_frame.html" name="menu"
scrolling="NO" noresize>
  <frame src="page0.html" name="content" noresize>
 </frameset>
</html>
```

Code 1. index.html

When you modify top_frame.html, set the highlighted size of the first frame the way that the bottom of the frame coincide with the bottom of the tabs. For the present height of top_frame.html, the above value works well in Internet Explorer 6, Netscape 6 and Opera 7. Here is the code of the top frame:

```
<html>
  <head>
    <title>Top Frame for Choices</title>
    <link href="menuDrawerStyle.css"
type="text/css"
       rel="stylesheet">
    <script language="Javascript"
src="scripts/menuDrawer.js">
    </script>
    <base target="content">
  </head>
  <body>
    <h1>Home Page</h1>
    <table cellSpacing="0" cellPadding="0"
border="0">
       <tr>
         <script language="Javascript">DrawMenu();
         </script>
       </tr>
    </table>
  </body>
</html>
```

Code 2. top_frame.html

top_frame.html is as simple as index.html. Its head contains a link to a style sheet and a link to a Javascript file. Its body contains a table with cells holding the images of the menu tabs. They are created via calling the Javascript function DrawMenu(). The fun starts with the Javascript:

```
/*menuDrawer.js ver.1.0. Creates tabs in a menu frame to
switch content in a center frame. L. Naszodi 2004.01.14. */
var tabSelected = 1;
tabText = new Array();
//Your set of menu items come here:
tabText[0] = 'Applications';
tabText[1] = 'Publications';
tabText[2] = 'Favorite Links';
tabText[3] = 'About Me';

function changeSelected(tab) {
 if (tab!=tabSelected) {
 //Change previously selected tab back to non-selected:
 objID='obj'+tabSelected;
document.getElementById(objID).src='images/toprightNotSel.p
ng';
 //Change newly selected tab to selected:
 objID='obj'+tab;
document.getElementById(objID).src='images/toprightSel.png'
;
 tabSelected = tab;
    }
} //end function changeSelected()

function DrawMenu() {
with (document) {
 for (var i = 0; i < tabText.length; i++) {
writeln('<td>');
writeln('<table       cellPadding="0"       cellSpacing="0"
border="0">');
writeln('<tr>');
writeln('<td>');
writeln('<img                  src="images/topLeft.gif"></td
align="bottom">');
writeln('<td class="tabtop"></td>');
write('<td>');
write('<img id="obj'+i+'" align="bottom" src="images/');
  if (i==0)
    write('toprightSel.png">')
   else
```

```
    write('toprightNotSel.png">');
writeln('</td></tr>');
writeln('<tr>');
writeln('<td class="tableft"></td>');
writeln('<td class="tabmid">');
writeln('<a onclick="tab='+i+';changeSelected(tab);"');
writeln('
href="option'+i+'.html">'+tabText[i]+'</a></td>');
writeln('<td class="tabright"></td>');
writeln('</tr></table></td>');
  } //end for
 } //end with
} //end function DrawMenu()
```

Code 3. menuDrawer.js

For the tab-like buttons I used six small images, with a total size of about 1k. The names of the first three are highlighted here, the other three show in the style sheet as background images. Only the top right segment of the tabs changes when one or another tab is selected, and only this segment has two versions that swap when a tab is selected. In function `changeSelected(tab)` the image objects are referenced by their ids, rather than by their names. This way the site works in Netscape 6, too, not just in Internet Explorer 6 and in Opera 7. In the latest W3C recommendations of migrating pages to XHTML, a stricter HTML, names have been completely expelled. You can read a short introduction of XHTML in this book.

At some places I had to use the `document.write()` function instead of `document.writeln()`. It is very important not to place a white space (a new line) between the tags `<img>` and `</td>`. If you do, the images of the buttons will disintegrate to their graphical elements in Internet Explorer and in Netscape.

Here is the last piece of code, the linked style sheet file:

```
 /* menuDrawerStyle.css version 1.0 creates tabs in a
menu frame to switch content in a center frame.
Author of version 1.0: L. Naszodi Jan 14, 2004. You
are allowed to use and modify the code freely but the
author of the original version must be mentioned in a
comment. */
body{margin:5px 5px 0 50px;/*top right bottom left */
  padding:0 0 0 0;
  font-family:'Times New Roman',Times, serif; font-
weight:bold;
  font-size:14pt; text-align: left;
  background: url(images/tagEdge.gif); background-
repeat:repeat-x;
  background-position: bottom; line-height:100%;}
h1 {background-color: lightblue; text-align: center;
font-size:20px;}
a { font-size: 18px; font-weight: bold; text-
decoration: none; margin: 0 0 0 0; padding: 0 0 0 0;
border: 0;    text-decoration: none; }
a:hover { background: silver; color: darkblue;}
td.tabtop { background-image:url(images/topbar.gif);
   background-repeat:repeat-x;}
td.tabright{background-
image:url(images/rightSidebar.gif);
   background-repeat:repeat-y}
td.tableft {background-
image:url(images/leftSidebar.gif);
   background-repeat:repeat-y}
td.tabmid {background: white;/*whitens the body's
background-image*/}
```

Code 4. menuDrawerStyle.css.

To avoid transferring more graphics, the effect of the mouse-over event is handled by the `a:hover` style element. It works very well in all the three investigated browsers. One minor and one major change is necessary to customize the tool:

At the beginning of `menuDrawer.js`, change the values of `tabText[]` array to the texts that you want to show on the tabs. No dimensioning is necessary. Just start the indexing from [0], give them the captions of the tags, and the code will know the number of tabs to create.

Write your content in `page0.html`, `page1.html`, etc. Sorry, you're on your own here. I can't do this for you.

Done? Then you have a complete dynamic menu-driven Web site. The rest of changes is a matter of taste. For example, the widths of the tabs are variable, depending on their captions. I like the way it is but if you don't, just put a couple of `&nbsp;` (hard spaces) to the front and the end of the texts of the `tabText[]` array, for example, in the short 'About Me' caption, and the tab widths become about uniform.

Do you want to change the image of tabs while the mouse passes over them? Modify the `a:hover` entry in the style sheet. However, avoid any modifications that change the size of the text. Otherwise the altering size will make the graphics jumpy.

Good luck, and let me know about your improvement ideas.

Summary

The suggested semi-graphical menu bar avoids lengthy opening of your Web page. The buttons have all the usual dynamics, such as change at mouse over, mouse out and click events. The number and size of applied images and the size of reusable code are very small regardless of the number of menu elements.

? Problem

Because of educational and historic reasons, the above solution uses a table for arranging the buttons of the menu. Rewrite the solution so that it uses `div` tags instead of table elements.

The only downside of innerHTML is that someone else invented it, not me.

Navigation On the Inner Track

.

Intro

In a previous article I encouraged the reader to follow the W3C standards and the rules I collected in the Web Developers' Ten Commandments. Before you draw the final conclusion that I am preaching to always obey the rules, I show some exemptions. I also mentioned that half of the innovative efforts aims at making products proprietary, i.e., to hide information even at the cost of making the product clumsier. Let's look at the other half, which includes inventive ideas that may depart from the standards but make the Web page faster and still maintainable.

A Non-standard Navigation

The `window.history` object has been introduced in Netscape 2 and in Internet Explorer 3 and it works in all newer versions since. Still, it has not been incorporated in the World Wide Web Consortium's document object model. Should we avoid it then? Of course we shouldn't. If an element survived three major version changes of two major browsers each, we don't have to be afraid that it will not be supported in the future.

The simplest way of creating customized "back" and "forward" navigation buttons is to utilize the history object. Here is the code:

```
<span class="button" onclick="history.back()"><a>
< Back </a></span>
<span class="button" onclick="history.forward()"><a>
Next > </a></span>
```

You can spare some characters if you replace the `.back()` and `.forward()` methods with `.go(-1)` and `.go(1)`, respectively. Since `window` is a default object, we may but we don't need to call the methods as `window.history.back()`, `window.history.forward()` or `window.history.go()`. This laxity is allowed but others are not. In situations where we don't declare the document type and the browser is forgiving enough, the page shows as intended. But we want to create enduring solutions that last in times when standards become stricter and sloppiness becomes less excusable. HTML 4 and XHTML are less forgiving as earlier HTML standards. Still, this article is about cases when we can depart from the official standards.

The two spans are clickable but don't really look like buttons yet. We must make them in a style description. Here is a simple but aesthetic one:

```
.button {border-style: outset}
```

If you want it work in IE 4 or 5.0, don't put it in an inline style, though. (You wouldn't do it anyway because you want to separate presentation from content, right?) And don't try it with Netscape versions earlier than 6 at all. I already gave up supporting Internet Explorer versions older than 5.5 and Netscape versions older than version 6. The above deficiency is at the bottom of the list of reasons.

Automated Creation of Menus

Many times we need to dynamically create page elements and change their contents. Starting from the above solution of the back button, here is a useful example for object creation. Let's say, you need to display a menu in several pages. Different pages should show a menu of the same look but of different menu items. You can make one common style sheet and link it to all pages. You can create separate lists of items for the different pages or one data set for all lists. The lists can be stored in a server side database, in an external XML file, in XML data islands embedded in the individual pages or in individual arrays in each page. For simplicity, I will use the individual array per page approach but the technique of menu creation is independent from the source of data. If the items come from an XML file, data set, or from a database, you can dump them in an array and from there the process is the same as below. We can use multi-dimensional arrays, if we also want to store other information in it. For example, each item may correspond with a file name. Then the set of file names can be stored in a second column of the array. However, the corresponding file name can be simply the button's label with an .html extension. In this case, a second column of the array is unnecessary because the file names can be dynamically generated from the item strings. This will happen in the example.

Let's fill up the `Items[ ]` array with some menu items:

```
var Items as new Array;
Items[0] = 'Home';
Items[1] = 'Arts';
Items[2] = 'Books';
Items[3] = 'About';
```

We can create corresponding menu buttons in a static HTML code, similar to the ones in the previous section:

```
<span class="button" id="btn0"><a href="Home.html">
Home </a></span>
<span class="button" id="btn1"><a href="Arts.html">
Arts </a></span>
<span class="button" id="btn2"><a href="Books.html">
Books </a></span>
<span class="button" id="btn3"><a href="About.html">
About </a></span>
```

If we need 20 buttons, we should write 20 similar lines and manually paste each array element in them, twice in each line. This is the solution that we can do with a stupid text editing program by copying, pasting, and modifying 20 lines. It is a boring process. In the meantime we can listen to music and pray that we won't mess it up. But if we know the routine, why don't we make the computer do the boring stuff? When I say computer, I don't mean the Web editor program in the computer. The difference between the code generated by the Web editor program and ours is that ours will be shorter, faster and more intelligent. The following script will create the same buttons:

```
function createButtons() {
  for (var i=0; i<Items.length; i++){
   document.write('<span class="button" id="btn' + i
+ '"><a href="');
   document.write(Items[i] + '.html"> ' + Items[i] +
' </a></span>')
  }
}//end function createButtons()
//Invoke:
createButtons();
```

The size of the Javascript code is about the size of the HTML code of four buttons but the JavaScript's size does not grow with the number of buttons while an editor-generated HTML code's size does. Our script is written once for all present and future pages, as opposed to their HTML code. Also, it loads once in a browsing session and executes from memory as many times as many new page openings invoke it. Naturally, the individual pages can have different set of items in their arrays, without rewriting our button-generating script for each new button set on each new page.

Fast Dynamic Changes

As I mentioned in ***One Code Fits All***, all major browser makers have accepted the `innerHTML` object but it is still not a W3C standard. The reason is simple. It does not fit in the W3C document object model. As Peter-Paul Koch, a freelance Web developer in Amsterdam pointed out in QuirksMode.org, changing a tag's content with `innerHTML` is much faster than with the corresponding W3C DOM method. The change of the above created buttons occur when a new page is loaded with the same style, same

Javascript but with different Items array. In fact, it is not a change; it is recreation. (Not in a sense of leisure but of regeneration. But it isn't hard work either, is it.) Can we change a button's label and its linking target without jumping to a new page? The answer is yes, yes. I say yes twice because I am going to show you two solutions. The first will use a standard method, the second the non-standard `innerHTML`.

Here is a live problem to solve. The opening page of a Web site is usually called index.html or default.html, and not Home.html, as the linked file name would be generated in the above solution. When you enter a URL in the address bar, like www.scriptwell.net, it opens the default.html from the site's root directory. How can we change the link of the Home button from Home.html to default.html in all the other pages? The process is straightforward. First, create the button as shown above. The difference is that we need to identify the anchors with unique `id`'s, because we want to change their `href` attributes. The core of the modified `for` loop will be as follows:

```
document.write('<span class="button" id="btn' + i +
'"><a href="');
document.write(Items[i] + '.html" id="anc' + i +> ' +
Items[i])
write(' </a></span>')
```

As a second step, after the `for` loop reset the `href` attribute of `anc0` anchor:

```
id = "anc0";
document.getElementById(id).href = "default.html";
```

Are we done yet? With the first solution, we are. But here is another solution with `innerHTML`. It will rewrite everything between the start tag `<div id="btn0">` and the end tag `</div>`. Skip the first step of the previous solution and replace the code in the second step with this:

```
id = "btn0";
document.getElementById(id).innerHTML =
    '<a href="default.html"> Home </a>'
```

The opportunities that `innerHTML` offers are limitless. We can make AJAX-like effects without knowing AJAX. Once you can change the HTML between tags, you can do anything and everything, without utilizing the Document Object Model. Imagine that you are changing the content between `<body>` and `</body>`...

Summary

Utilizing non-standard elements, like `innerHTML`, can be justified. It is absolutely proper once the industry accepted the element and it is better than the standard alternatives. I show an object method and an object property that work in all modern browsers and are superior over the ones that have already been incorporated in the W3C document object model.

Questions and Problems

1. Can you apply one-dimensional arrays for storing the labels and the linked file names if a menu item consists of more than one word? (Help: Write a function that generates one-word filenames from the multi-word items by cutting out the spaces between the words.)
2. Can your solution link to "Yahoo!"? The problem here is that a file name cannot contain an exclamation mark. Consider other possible file name limitations, like all-lowercase letters, disallowed characters, etc.

This page has been left intentionally blank.

PART 2. USABILITY

Microsoft Internet Explorer
Be prepared
Unfinished software is coming to the computer near you.
OK

Don't terrorize me!
If you want a string in lower case,
make it lower case yourself.

Usability, Abusability

The Wrong Message

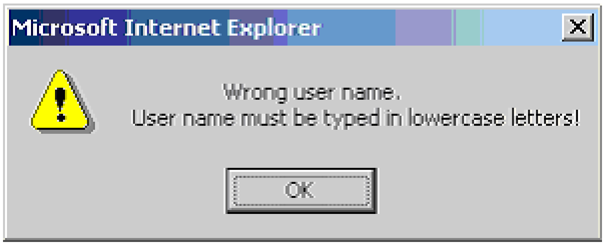
Intro

As the Developer's Bible says: "Thou shalt not leave loose ends in thy programme." But are all those branches fixed that end with messages? My message to those developers who think that a message can replace their responsibility to finish their jobs is: "We don't eat half-baked bread. Why must we consume half-baked programs?"
In this article I will use the word ***stupid*** *several times. Those who think of this word as four-letter, i.e., too offensive, stop reading now. Stupid, stupid, stupid. Stupid.*

It would be ridiculous if an application stopped in the middle of a calculation and instructed the user to pick up a pocket calculator, sum a row of numbers seen on the screen and enter the result in the computer. We bought a computer system because we want it to add numbers for us and not vice versa. Do programs make us do things that they are supposed to do? Unfortunately, yes. The following example of an annoying application is real. The messages have been recreated, their headers modified to hide their sources and to avoid legal consequences. Successful companies with unintelligent developers may have very smart legal representatives.

The wrong first impression

The login instruction of the program tells me to enter the user name in all lowercase letters. Even worse, the instruction comes after the first attempt of logging in:



Wrong user name.
User name must be typed in lowercase letters!

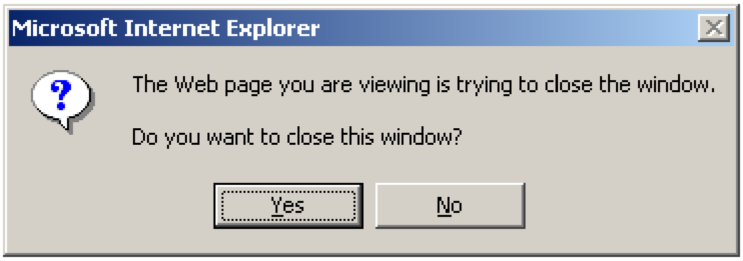
We get these kind of ignominious messages so often that we feel guilty daring to capitalize our own names or not to check the Caps Lock key before typing. Most of us feel stupid compared to the genius who can program a logon screen. But can he really? My message to the programmer would be this: *"Do not terrorize me! If you want the string in lower case, make it lower case. Instead of instructing us to reenter, why don't you modify your stupid program?"* A single line in Javascript, like

```
username = username.toLowerCase()
```

would make it right. The form of the statement may vary, depending on the language used. But when I say language, I mean programming language and not the language that does not use the word "stupid". You may say that this is not the case with case sensitive user names and you are right. But the above example is different. A user name that must use lower case letters only is not case sensitive, only its internal representation is. Similarly, user entries of e-mail addresses and Web page addresses should never require case sensitive input. The program should not distinguish between the entries of MeStupid@MyStupidSite.com and mestupid@mystupidsite.com. It should accept both variants and should take care of the conversion internally if necessary.

The wrong developer

Then another problematic message pops up as proof of the developer's dilettantism. Several variants of the following have been discussed on Web forums since years. When someone wants to close a window programmatically, the browser may display the message:


Microsoft Internet Explorer

The Web page you are viewing is trying to close the window.

Do you want to close this window?

It is pretty annoying. The developer should handle the case when the user clicks ***No***. Many avert responsibility saying that it's an "IE thing", not the developer's fault. Some forum members say that one cannot rid of the message because it is related to some kind of security issue. Others suggest hacking solutions that work with certain browsers and don't work with others. The worst suggest to tell the user in the manual or in a previous screen to answer ***Yes*** to this question. A question should not come up if the user does not have a real choice of answers. These "experts" should know that there is a cross-browser solution. In short, a

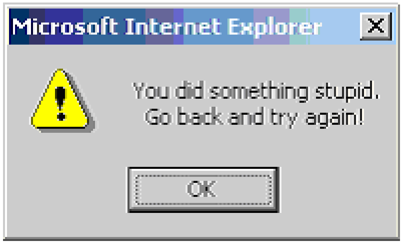
```
window.opener=top
```

Javascript statement should precede the

```
window.close()
```
command and the annoying message simply does not show up; the window closes without bothering the user with a question that he has nothing to do with.

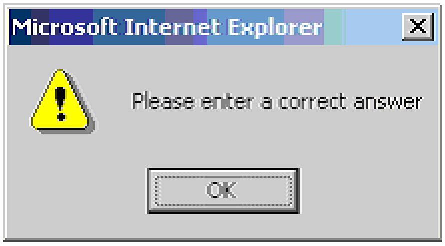
The wrong message

Many programmers ignore the importance of correct messages. Deep in their heart they would just notify the user:


Microsoft Internet Explorer

You did something stupid.
Go back and try again!

Of course, the user is dear; the message must be polite and technical. Beside the courteous language, the following real text


Microsoft Internet Explorer

Please enter a correct answer

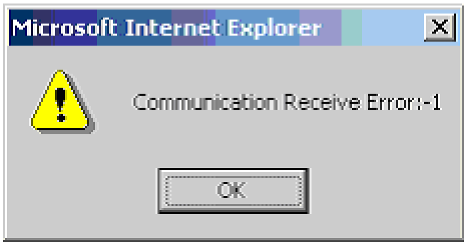
carries about the same message as the previous one and does not help us much, either. After the obnoxious login incident and the pointless error message a user with self-respect may say to the developer: *"I am not stupid. But you are. If you know that my answer is incorrect, why don't you* *tell me why? Did I type a longer string than expected? Did I enter a decimal point where you wanted an integer?"* But the developer has a great advantage: Unlike you to him, he can send out annoying messages to you.

I must use another application every day that explicitly says at a point: *"Do not enter a command manually."* Because the particular command I must issue is not programmed in a function key or in a menu, I have to manually enter it. There is no text box because the program is not prepared for data entry. Wherever the cursor is on the screen, I have to start typing. How do I know about this "feature"? I participated in a training where I was told so. To me, a program is not a program yet if it requires training, where experts explain what to do when a controversial message shows up. Same with user's manuals. I created complex expert systems that do not require training or manual. When a new user misses them, I usually ask: *"Can you use Windows without a manual? Then you don't need a manual to my program, either."*

Complexity is a private matter and it should stay between the developer and his product. The user interface must be self-explanatory, regardless of the complicated internal processes. The developers should delete most things from the manuals and put them in the program, where they belong. Context-sensitive help, meaningful captions, reasonable messages and finished programming tasks are the way to go. An application needs some more work before distribution, if it has a paragraph in its manual: "When you see the message ***Press any key*** you may press any key, except the Tab key or any of the function keys."

The wrong address

The torture does not end here. After a common action a new message pops up:


Microsoft Internet Explorer

Communication Receive Error:-1

Getting this error, the benign user assumes that something happened outside the application, for example, the network connection has been lost. However, in our case the message relates to an internal problem of the application that the developer did not handle properly. The message obviously targeted the program tester, not the user.

When the postman drops a letter in my mailbox with my address but with the name of the previous tenant, I can send the mail back with a "Wrong address, return to sender" note. What can I do with a message like this? The developer could have done something. He had the problem coming to him and he did not finish the branch properly.

The application is accompanied with long manual, including interpretation of error messages. If you are lucky, you may find the explanation, what to do in case of certain errors. Many times the problem solving process could have been incorporated in the program, rather than displaying the error message and telling us to look up the solution in the manual. So, if you get a message that makes you read the manual or one that instructs you to pick up your pocket calculator, return the program to the sender. It is not ready for sale yet.

Summary

A program is not done yet if it instructs you to do things that could have been automated.

A program is not done yet if it displays messages that do not apply to you.

A program is not done yet if it forces you to read the manual to understand the messages.

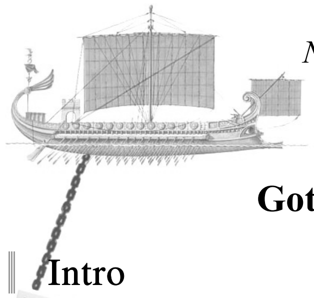
A program is not done yet if it has a large manual.

A program is not done yet if it needs extensive training.

Questions and Problems

1. Does your program display messages instead of doing what the user is instructed to do?
2. Does your program have a large manual? Are there paragraphs in the manual that could be preceded with a sentence: "If you can use Windows then you don't need to read this"? If you took out these paragraphs, is the manual still bulky?
3. Have you ever felt that your development is not done yet but financial or organizational pressure made you give it out prematurely? Did you "finish" unfinished branches with messages then?

Navigare Necesse Est ...
Pompeius (First Century, B.C.)



Navigating on the Web is necessary.
From a job description
(Twenty First Century, A.C.)

Got Lost on the Web?

Intro

The ancient Romans did not have Internet but they already knew one of its biggest problems: to recapture the location where your navigation led you.

You are an experienced Web surfer. You just found an interesting article on 'Human Stupidity' but your boss is approaching your desk. A double-click highlights and selects the site's address in the URL box, a Ctrl/C copies it to the clipboard and another click sends the browser to the home page of the corporation's Web site. You may keep on surfing while your boss is breathing in your neck or wants to see something on your computer. Once he left, you retrieve the saved address by pasting it from the clipboard to the address bar but the article of interest just isn't there.

Where have all the pages gone?

Take a simple scenario. You want to share the found article on 'Human Stupidity' with your brother. Sending the copied URL of the visited page in an email will take him to the same article that your browser is displaying now, right? Wrong. OK, you may say, in our fast revolving world sites change fast. They probably restructured their site before your brother could get there. Right? Wrong again.

The above scenarios would only restore simple pages. However, online news sources and in general, most Web sites build their pages in frames. The address shown in the

URL box is that of the frameset. It does not refer to the actual contents of the frames. The present state of the browser is a good example, if you opened it with my book-imitating frame. The frame's file name is `default.html`. No matter how you go back and forth among the articles, i.e., how you change the content of the main frame, the address bar will show the same reference, `default.html`. If you copy and paste the URL from here or from the previous article, you will get a third content, i.e., the opening state of the frame.

There are ways to pick the address of the very article you are seeing now. In Internet Explorer, you need to right-click on the article and take the URL from the Properties window, or, click on the Create Shortcut item that adds an icon to your desktop, which links to the article. Either way, reopening the link will not show the whole page what you have seen, only the content of the right-clicked frame. This may be good in case you don't want to show the surrounding ads to your brother, and may be bad if some other frames contain valuable details. In slightly different ways, one can take the address of the frame's content in Opera and in Netscape, too. In all cases, the problem is that the surfer should know that he is dealing with the content of a frame, not the entire page, and he has to apply special tricks to get the address of that content.

Let's look at the issue from the viewpoint of a Web developer. Do you want to disappoint a visitor of your site, who is intelligent enough to know how to copy and paste the URL of a simple page but not technical enough to know about HTML frames at all? Of course not. Some nice designs don't even show the visitor that the page is constructed in a frameset.

The contradiction comes from the fact that the URL box shows the frameset's address only, no matter how the frames changed their contents after a few clicks on links.

There is a way to create a mutual and unambiguous correspondence between the URL and the actual content. In Javascript terms, the URL box shows the value of the location object. The location object may end with a so-called search string. The search string always starts with a '?' character. Even if there is no search string attached to the location string, the search string has an internal value, namely a '?'. We can store and retrieve the actual state of the frame in the search string. Javascript can access the search string as `document.location.search`. The rest is finger play. After the file name of the frameset, we can programmatically put a distinctive search string in the URL to reference the content of the changing frame. When pasting the extended URL to the URL box and hitting the Go button, the loaded HTML file should evaluate the search string and load the corresponding file into the adequate frame.

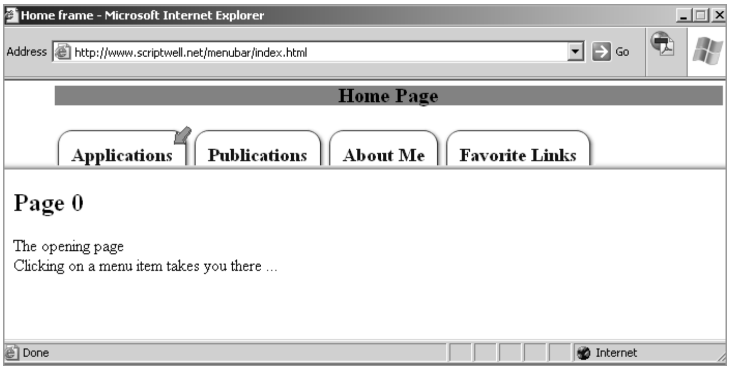
Again, the first part of the URL should contain the address of the frameset file as before and the second part should somehow, not necessarily with its file name, refer to the file that should be displayed inside the frame. Usually, a link that changes the content of a frame does not contain the frame's address and a click on the link does not change the URL box. Now, in the `href` element of the link we need to provide the frame's address, because we want the URL show it along with the reference to the linked page. It means that links that previously pointed to content files, should point to the frameset file with a search string referring to the content file.

Let's modify the sample files of a previous article, ***Dynamic Menu Under 10k***. It has a frameset called index.html, with a top frame that contains links and with a bottom frame that displays the contents according to the clicked links:

```
<html>
  <head>
    <title>Home frame</title>
  </head>
  <frameset rows="90,*" border="0" framespacing="0"
frameborder="no">
    <frame src="top_frame.html" name="menu"
scrolling="NO" noresize>
    <frame src="page0.html" name="content" noresize>
  </frameset>
</html>
```

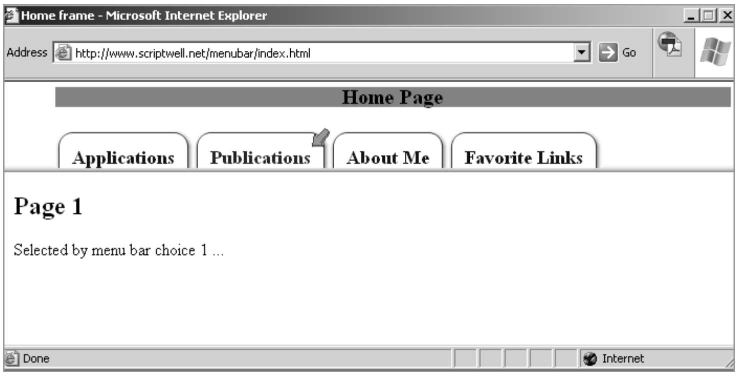
Code 1. The original index.html.

Clicking the tabs alters the content but leaves the URL unchanged:


Home frame - Microsoft Internet Explorer
Address
http://www.scriptwell.net/menubar/index.html
Go
Home Page
Applications
Publications
About Me
Favorite Links
Done
Internet

Page 0

The opening page
Clicking on a menu item takes you there ...


Home frame - Microsoft Internet Explorer
Address
http://www.scriptwell.net/menubar/index.html
Go
Home Page
Applications
Publications
About Me
Favorite Links
Done
Internet

Page 1

Selected by menu bar choice 1 ...

Various contents under the same address

To recognize the search string, a new frameset is created dynamically, according to the search string:

```
 <html>
  <head>
  <title>Home frame</title>
   </head>
   <script type="text/Javascript">
     var choice=document.location.search;//always
starts with a '?'
     frmsrc='page0.html'; //default
     if (choice.length>1) {//not just the '?'

 choice=choice.substring(choice.lastIndexOf('=')+1,
choice.length);
      frmsrc='page'+choice+'.html?choice='+choice
     }
     with (document) {
     writeln('<frameset rows="90,*" border="0"
           framespacing="0" frameborder="no">');
     writeln('frame src="top_framesimple.html"
name="menu"
       scrolling="NO" noresize>');
     writeln('frame src="' + frmsrc + '"
name="content" noresize>');
     writeln('</frameset>');
     }
   </script>
</html>
```

Code 2. The modified index.html.

The URL box accepts the search string `?choice=2` that opens `page2.html` in the frame. Entering different choices in the URL opens different pages and vice versa. If you open a page by clicking on the corresponding link, the choice of page will be reflected in the URL.

The above changes do not make the URL follow the clicks on the tabs. To make the URL box reflect the displayed page, we should make two more simple changes.

Change the base target in the `top_frame.html`'s `head` from `content` to `_top`:

```
      <base target="_top">
```

Change the `href` elements in function `DrawMenu()` to refer to the frameset with the proper search string:

```
document.writeln('
href="indexnew.html?choice='+i+'">'+
tabText[i]+'</a></td>');
```

A third change is necessary to complete the project. The original Javascript code taken from the previous article distinguishes between selected and unselected tabs. The menu drawing function in the sample of this article has been simplified. Switching the images by tab selection has been removed, not to create confusion. One can put it back and make the image switch controlled by the search string, similarly to the content switch of the lower frame. But let this be your homework, folks. Have a good navigation. Ahoy!

Closing thought

Once I brought up the topic, I want to discuss another application of search strings. I will show you how those smart, self-filled emails are created. But this would be the content of another article ...

Summary

The content and the URL of a page do not necessarily correspond when the page is a frameset. The suggested technique creates a mutual and unambiguous correspondence between the URL and the content. This way, a copied URL provides the same content as the original page.

Practice question

The sample code 2 has been called with the search string `?choice=2`. Would it work if we used the word `selection` instead of `choice` here? Why or why not? (Look at code 2.)

To get a new browser is slightly cheaper
than my time - because it is free.

A Five-Minute XHTML Class

Intro

Are you planning to take XHTML classes or read an XHTML book? If you know HTML well, don't do it, unless advised below. XHTML is not more than a cleaner and stricter HTML. OK, it is extensible. That is what the X stands for. But who cares about extensibility when one can do almost anything with HTML, CSS and Javascript?

HTML and XHTML have about the same tags, attributes and events. Some HTML attributes that became deprecated in version 4.01 finally are out in XHTML version 1. Here is an example. Because of backward compatibility, the `<td>` HTML tag still may have the obsolete `bgcolor`, `width`, and `nowrap` attributes, although other up-to-date solutions can deliver the same features. In XHTML the formatting attributes are not just outdated but unsupported, too. So, the same

`<td bgcolor="red">` line in the same browser can or cannot paint the table cell red, depending on the first, DTD line of the page. If it says

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

the red background does not show. If it says

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
```

the red background shows in the above table cell. I would not encourage you to use the latter DTD line for an alleged XHTML page just to keep the obsolete elements, because deprecated elements and HTML as a whole sooner or later become unsupported. Well, probably later than sooner. Still, you better replace the obsolete elements with the newer ones. I suggest to paint the background yellow as seen above with an inline style:

```
<span style="background:yellow">XHTML 1.0
Strict</span>.
```

Presentation with style elements has already been introduced in HTML. By applying them, one can move smoothly in the XHTML world. Until you replace all HTML presentational elements with style elements, you may stick to the `Transitional DOCTYPE`.

The basic rules that differentiate XHTML from HTML are as follow:

- XHTML elements must be properly nested.
- XHTML documents must have the <html> root element.
- Tag names and attribute names must be in lowercase.
- All XHTML elements must be closed.
- All attributes must have a value.
- Attribute values must be (single or double) quoted.
- The `id` attribute replaces the `name` attribute.

If you were a conscious HTML developer you probably obeyed about two-third of the above rules anyway. Here is an example that may work in HTML but doesn't work in XHTML:

```
<html>
  <head><title>Ill formed</title>
  <body> <b><i>Hey you!</b></i><br>
    Check the box if you like to rock and roll:
    <input type=checkbox checked>
  </body>
</html>
```

Can you see the problems? Come on, name at least one!

1. `<b><i>text</b></i>` is not properly nested.
2. `<head>` is not closed. The `</head>` end tag is missing.
3. The checkbox value is not quoted and the checked attribute of the input box is empty. The correct form is

   <input type="checkbox" checked="checked" />
4. `<br>` is not closed.

OK, as an HTML developer, you were not supposed to know the last one. Empty elements, i.e., `meta, link, br` and `img` that don't have end tags should be closed like this: `<br />`. A space before the forward slash "/" character is not part of the rule but necessary to avoid a glitch in Netscape.)

A kind note to the reader: If you need further explanation to the above example, you need to go to that XHTML class or read that XHTML book. If you did not understand the first two problems, you need to take an HTML course before dealing with XHTML.

Once you follow the above rules, you may try to change your `DOCTYPE` from `HTML 4.01 Transitional` to `XHTML 1.0 Transitional`. I would not advise you to try the `XHTML Strict` standard at first. The HTML version of this page, for example, is valid `XHTML 1.0 Transitional` but it would not pass the strict criteria, because I used the `target` attribute for an anchor, which is unacceptable by `XHTML 1.0 Strict`. If I set the target by adding a

```
<base target="_blank" />
```

tag in the head, I could have eliminated the target attribute from the restricted `<a>` tag but then, my navigation buttons worked incorrectly and opened the previous or next page in a new window. On the other hand, putting the `target="_blank"` attribute of the anchors in the Javascript code that creates the navigation buttons, would have made me strict XHTML compliant. In this case the page would have passed the strict criteria because the non-compliant part was hidden in a script, in which the validator does not check the generated HTML or XHTML code. Then I would have displayed the W3C's `XHTML 1.0 Strict` icon legally but I would not be proud of it. I am proud of the cross-browser logo though, that I designed, granted to myself and displayed at the bottom of all pages on my site.

The following anchor is `XHTML 1.0 Strict` compliant. It opens the page.html page correctly in a new window if Javascript is enabled but in the same window if Javascript is disabled:

```
<a href="page.html"
onclick="window.open('page.html','myWindow',
width=400, height=400');
return false">Open page.html </a>
```

You may test the validity of pages at http://validator.w3.org. Are your pages valid `XHTML Transitional`? Good! Actually, transitional does not mean exactly what it says. One should also use transitional `DOCTYPE` if one assumes that the used browser does not understand CSS and the developer has to double her presentational efforts by including old fashion formatting with HTML tags, not only with CSS. If you are heading in that direction, you really need to go to a class or read an XHTML book. Recently I gave up dealing with browsers without CSS capability. I just tell the users: Go get a new browser that supports Cascading Style Sheets. It is slightly cheaper than my time because it is free.

Again, if you understood the above seven rules and your page passed the XHTML Transitional validation, you are an XHTML developer already. Class dismissed.

Summary

Following the standards helps you to create fast, maintainable cross-browser pages. Big Web sites seem to be ignorant and they don't obey the rules. You can do better than them without taking XHTML classes.

XHTML is not more than a cleaner and stricter HTML. Strict XHTML pages are rendered faster than loosely written HTML pages.

Problem

In a previous article, ***Let's Make the World Round Again***, I showed a solution for creating rounded frames. The solution uses table elements with non-XHTML background elements, discussed in the introduction of this article. Rewrite the `roundborder.js` script to make it XHTML-compliant.

No matter how fast the page renders,
twice as slow is always twice as slow.

The Ultimate XHTML Template

Intro

You may have read some articles about migrating from HTML to XHTML that suggest that all you need is to rewrite the page by following a few rules; stick a doctype in front of it and your page will become a fine and valid XHTML. Don't believe them. The page may become valid XHTML but not fine. You remove the doctype and the page looks fine again. Now you are close to forget the whole standardization issue. I hope that I didn't make the same false impression of easiness with my ***Five-Minute XHTML Class****. If I did, I have to refine my statements. Nevertheless, don't give up! Creating good-looking standard pages is still not too difficult.*

When I said that HTML and XHTML have about the same tags, attributes and events I wanted to put the emphasis on the word 'about'. When you decided to create XHTML pages you agreed to obey stricter rules than as if you write HTML. However, if you only think that you developed an XHTML page and, in fact, it is not XHTML, the browser has to reinterpret it according to a lower standard that the page meets. This mode of operation, when browsers try To Whom It May Concern: find an appropriate older mode to display an (X)HTML page is called quirks mode. Quirks mode also kicks in when the page does not contain a doctype declaration or when it contains a doctype that calls for quirks mode.

These rollbacks to lower standards result in an unpredictable and non-uniform look of the page. One browser may stop rolling back at an earlier version as the other. Ad hoc modifications, trial-and-error efforts may fix outstanding discrepancies but the general problems will come back. A painstakingly designed page may look the same in FireFox 1.0 and in Internet Explorer 6.0 but newer browser versions like IE 7 will display them differently, and the tinkering must start again. For example, IE 6 implemented a confusing rule. If a doctype that triggers strict mode is preceded by an xml prolog, like

```
<?xml version="1.0" encoding="iso-8859-1"?>,
```

the browser falls back in quirks mode. Fortunately Microsoft removed this "feature" from IE 7, and unfortunately those developers, who used it, have to tinker their pages again to satisfy the IE 7 users. Henry Sivonen, a Finnish expert, collected an excellent list of relationships between doctypes, browser versions and modes.

The transition from HTML to XHTML emphasized the need to eliminate tables as basic design elements. Not only because rendering a table is a difficult task for a browser, and using tables for arranging non-tabular page elements is not a good idea. The browser must read and parse the table twice before it finally arranges all the elements, because the content and form of the last cell may redefine the arrangement of all the previous cells. So, if you design your page structure in a table, as most designers do, the browser will go through the whole page twice. No matter how fast the page renders, twice as slow is always twice as slow. But forget about speed concerns now.

The main reason to eliminate old aligning table structures is that XHTML introduced some drastic changes in table elements. For example, the height attribute and style have been expelled from the table, row and td tags. These changes have been logical and served the goal of separating presentation from content, but they have ruined many designs that were based on an outside table at top level. Most designs included multiple tables inside tables inside a table right below the body level. In strict mode these older table-driven layouts easily disintegrate.

The question is whether we should force the newer browsers back into quirks mode to reserve the old layouts, or, we should revisit our codes and redesign our pages without tables in XHTML. The latter needs some extra efforts, than dragging and dropping, but again, if you are willing to move in the solid field of XHTML, your pages will become faster, more predictable, cross-browser and platform independent. In populist language I may ask: Do you want to make pages for the past or for the future? In fact, keeping your pages running in quirks mode is also for the future in a sense, because it provides job security: Your pages will always need your modifications as browsers evolve. XHTML is the progressive way of approach and it may solve returning problems for good. Your page will be displayed correctly for eternity, or at least five more years, and through one major browser generation change without your touchups, even if you lose your job tomorrow. (What a great perspective!)

The desired look

I show some basic designs that many developers cannot accomplish without tables. My designs don't use tables; work correctly both in HTML and XHTML; from quirks to strict modes; in any modern browsers. They do not use scripts for the alignment or for the resizing. They look the same on various monitors with different resolutions, in various browsers and browser versions, in various platforms.

The sample page represents a commonly desired page layout. It has a header, a main block, and a footer. The page of fixed width is horizontally centered, so, it does not tip over on a large, high resolution screen, as some left aligned pages do. Vertically it spans across the whole height of the window. Regardless of the window height, if the filled area were smaller than the height of the page, the height spans vertically and the footer sticks to the bottom of the browser window. Naturally, if the filled area does not fit in the window, the page does not contract vertically or horizontally but scroll bars appear and help to show the outstanding parts of the page.

The sample page is valid XHTML with XHTML doctype. So, it will not fall back to quirks mode in any modern browsers. However, I can remove the doctype, making the page fall back in quirks mode, and the layout remains the same. This can be handy if I do something quirky, like use the non-standard innerHTML inside the basic structure. I checked it with Mozilla/FireFox 1.5, Internet Explorer 6 and 7 on Windows, Opera, Safari on Mac and FireFox on Linux, and it looks the same.

The new template

Let's start to play with a simple page that has the usual header - main block - footer arrangement, and some intentionally chosen ugly colors to draw attention to the structure.

Template #1, the first modified version centers the page horizontally. Unlike many other solutions, this one works both in quirks and standard modes, and looks the same in Mozilla/FireFox, in IE 6 or 7 and in Safari. You can test the former mode by removing the strict XHTML doctype declaration. You may replace it with transitional XHTML doctype, too. Sometimes I'm intentionally rolling back to transitional XHTML because I like `iframes` and `innerHTML`, and they are not allowed in strict documents. The following style horizontally centers the fixed width div by shifting its left edge to center, then back half width:

```
#mid {
    width:760px;
    position:relative; left:50%; /*Shift left edge
to center*/
    margin-left:-380px;          /*Shift back half
width*/
}
```

Now that we centered the page horizontally, we can work on aligning it vertically. After all, a short page in a long window can give a feeling of "tipping-over", too, as left aligned pages in a wide window. There are two ways to improve short pages: We either fill the full height of the window, by pushing down the footer to the bottom, or, centering the page vertically as well.

A couple of simple CSS additions span the page to the full height of the window:

```
html, body {height: 100%;}
```

is necessary for FireFox in strict mode, but it is required in IE in quirks mode, too.

The actual full-height layout is due to the CSS

```
#mid {
    min-height: 100%;
    height: 100%;
}
```

The first attribute makes FireFox work correctly in strict mode but is unknown for IE; the latter is for IE and for some doctype constellations in FF.

See the horizontally centered and vertically spanned page, Template #2.

Again, the template displays uniformly in all modern browsers, in both quirks and standard modes, with or without a strict or transitional doctype.

Now that the page occupies the whole window height, we can push the footer to its bottom.

Let's add the attributes

```
     position: absolute;
     height: auto !important;
```

before `height: 100%;`

to `#mid`, and

```
     position: absolute;
     bottom: -1px !important;
     height: 20px; margin-top:-20px;
```

to the `#footer` CSS, and voila! In Template #3 the footer is sticking to the bottom of the window when the page height does not exceed the window height.

The following improvement makes the viewable area shorter and centers Template #4 vertically:

```
body { /* in FF strict requires for spanning the
height*/
    height: 80%;
/* In quirks mode 100% pushes the footer to the
bottom */
}

#mid {
    min-height: 80%;/*Works for FF, unknown by
IE.*/
    height: 80%; /* Works for IE 6 and 7. */
    top:8%; /* = (100% - height)*height/200 */
}
```

Final Note

Printing horizontally centered pages can be a pain in some browsers. An additional simple printer-only CSS will fix the problem. In our case place a link in the head:

```
<link href="printstyle.css" rel="stylesheet"
type="text/css"
media="print" />
```

and the following styles in the printstyle.css file realign the page for print:

```
body {
    margin:1em; padding:1em; background-
color:white;
    font-size: 12pt; font-family:Georgia serif
}
.mid, #mid {
    margin:1em; padding:1em; border:0px;
width:760px;
    position:relative; left:0; width: auto;
    color:black; background-color:white;
}
```

Summary

This article suggests a tableless XHTML template that provides the page layout that many developers prefer, and a few is able to correctly create without using tables: The content is placed in a fixed width column and centered in the window, no matter what the window size is. This design is obviously superior to the so-called liquid, relative-width design, which automatically narrows the page elements when the user narrows the window width. The suggested design also centers a short page vertically, or makes it fill the full height of the window, if its content-filled height would be shorter than the window height. This feature comes handy when the page contains a footer that should stick to the bottom of the window.

Problems

- Can you add a shadow to the left and bottom edges to the above designs that give the page a 3D effect as if it was elevated a little? I accept graphical solutions but pure CSS is preferred. However, no tables, please!

- Have you already used templates that provide similar centered or vertically spanned layouts? If yes, please check the following:
    - Do they look the same in FF, IE6, IE7, Opera, and Safari?
    - How about removing the doctype; setting the doctype to XHTML strict or XHTML transitional? Does the page look still the same in all cases? Again, check the three standards with at least three of the above browsers.
    - If your sample page passes the above nine tests, try to disable Javascript, as many users do and rerun the tests. Does the page still look the same?

If you think that it's not you who pays for the free software or for the free TV channel, get out of here. This is for adults only.

Pay for the Fly in the Soup

Intro

Older sailors of the sea called Internet remember the resizing bug of Netscape 4. It was more like an elephant, rather than a bug. Whenever one wanted to resize the browser's window, its content got messed up. Since then browsers became better but still not bug-free. This article is about a bug of the latest browsers. But first a short history lesson...

People over 50 may remember the pioneer age of computer science. Most software was free. Unless it was against national security or strict company policy, programmers exchanged their ideas, subroutines and program snippets free of charge. The system worked well and technology evolved fast. Naturally, all software bore bugs but free access to the source code helped the community to get rid of many bugs. The open-source notion was a live practice before the Open-Source Initiative. I remember the time when a dozen of free editors, most of them better then today's Notepad, were circulating in the programmers' community.

Then there came Microsoft and other software companies and software for fee replaced software for free. Also, loose chats among developers were replaced with strict communications between users and lawyers of companies. You cannot even install or start an application without clicking on the "I agree" button that legally binds you to obligations that you are not willing to read fully. The so-called free software is not free anymore, either. Some companies talk straight like the owners of Opera: "Our supporters pay for your browser. Want a free browser? Then put up with their advertisements. Don't like ads? Then pay for the ad-free version."

Others are shrewder. They say their browsers are free. Period. But you pay for Internet Explorer indirectly when you buy Windows or a computer that has IE installed. The development costs and the profit are hidden in other product's price. You, as an individual, may avoid paying directly for software but, as a member of the users' community or as a buyer of an advertised product, pay enormous price. Similarly, when you watch a free TV channel with the same brand of beer in your hand as in the guy's on the TV screen, you already paid for the TV show. If you own a car, remember: you are paying for the "free" TV show that has been interrupted by a shouting car salesman. The car dealership pays the TV station and you pay the car dealership. The price of the TV show is part of your monthly car loan installment. The shouting salesman is free, although you'd prefer to see him behind bars instead of letting him interrupt the TV show that you paid for through him. If you still think that you don't pay for the free stuff, then get out of here. This is for adults only.

Iframe, you frame

In a previous article, *Info On the Fly,* I explained the use of the `iframe` tag. Let's return to the fly, more exactly to the fly in the soup. Netscape 4 introduced a revolutionary feature, the inline layer or ilayer. It is like picture-in-picture on TV. Microsoft did not accept the introduced tag but used its idea and "invented" the inline frame or `iframe`, with a similar functionality. The latest versions of Netscape and Opera incorporated the `iframe` tag as well. However, Internet Explorer 6 has an `iframe` related bug. During a full year from March 1st, 2004 I could not find any reference or literature on this bug. It's unlikely that the fly fell only in my soup, but hey! Some people can live with flies in their soups. I can't. Allow me to speak up.

The problem is a little complex. When you place an `iframe` in your HTML page and scroll its content dynamically, by moving the `iframe`'s content to an anchor, then, beside the wanted scroll inside the `iframe`, the whole document scrolls down until the top edge of the `iframe` reaches the top edge of the including page or until the bottom of the page comes up. (When I say scrolling down, I mean the upward motion of the page, approaching its bottom.) The phenomenon does not occur if the size of the outside page does not exceed the size of the window or if the outside page is already scrolled all the way down. Of course, in these cases there is no space for further down-scrolling.

The http://www.scriptwell.net/fastweb/TOC.html contents page is a good example. Just narrow the window's height, drag the scrollbar a little upward, then move your mouse over the links and you can see the whole page jumping up. There, the `iframe` and the links almost reach the bottom of the page, so, chances are that when you move the mouse over one of the links, which event causes the dynamic scroll of the inline frame's content, the whole page cannot scroll much. But when you trigger the mouse-over event with having the scrollbar higher than the bottom, you can see the page jumping. Again, it does not happen with Netscape 6.2 or with Opera 7. I almost wrote Netscape 6.2 and above because the first Netscape 7 worked correctly, then, in version 7.1 the `iframe` bug propagated in the world of Mozilla, too. FireFox, which is based on the same Mozilla 5.0 version, has the jumpy attitude, just like its sibling Netscape 7.1. Also, don't try my example with

Netscape 6.0, either, because of another minor problem. NS 6.0 does not support the used `onclick` event in a `span` tag.

I tried to get around the problem but I could not find a simple solution. It is like Netscape 4's resizing bug. The latter could be eradicated with a one-line Javascript that automatically refreshes the window after every resizing but this solution created other well-documented problems that made Netscape come up with a new version as soon as it could. The bug I found seems more stubborn than that. The only non-blinking work-around I found so far is so ugly that I do not dare to publish. I hope you find a better solution. Until then, let's accept the official standpoint: It is not a bug. It is a feature.

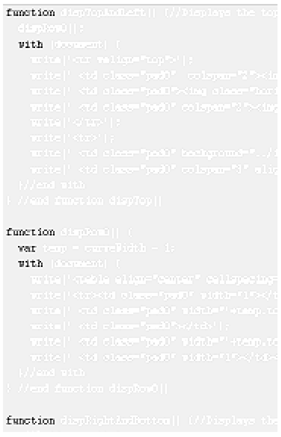
Summary

The shift from free software to paid software products did not change the fact that the products contain bugs. Some of them can be avoided by cautious programming techniques, some others we have to live with.
I show an undocumented one in IE 6 that bugs me. Unfortunately, Mozilla has it in its latest version, too. Opera fans, clap your hands...

Questions and Problems

Do advertisements bother you? How about their hidden price you pay? How much of your cell phone bill do you think is spent on marketing? I accept any unacceptable answers around 50%. Do you?

Find a solution of the outlined problem. It must be of reasonable size. It cannot suggest to load various documents in the inline frame instead of scrolling its content. The idea in the previous article was to have the whole content of the `iframe` loaded and available immediately, without a roundtrip to the server.



Trust me.
It won't take long and it won't hurt.

One Code Fits All

Intro

Which browser should we consider when writing a Web application? Technically none of them and practically all of them. We should write our code the way that the resulting pages look the same, or at least similar in all modern browsers. Theoretically you just need to follow general, browser-independent standards. "What standards?" you may ask. Unfortunately that's what browser makers asked for decades, too.

The standards

The World Wide Web Consortium, W3C in short, is an independent expert body that sets the rules. They don't issue laws, only specifications and recommendations for meta-languages like HTML and XML, and for styling properties like the ones in CSS. As an organization recommending standards for browsers, they don't create browsers, either. They have an open-source initiative though, on developing an Integrated Development Environment with browser functionality, too. It is called Amaya. Amaya is a Web editor, with the intention to be a comprehensive client environment for testing and evaluating new proposals for Web standards and formats. As of January 2005, it has its version 9.0 with many good editing features but still far from working as a fully functioning browser. Browser makers try to follow the rules that actually don't confront with their schedules and financial interests. After decades of a nasty jungle war browsers tend to agree in the most important issues. The leading browsers, Internet Explorer, Netscape and Opera can display well-written, HTML compliant pages with about the same look and functionality now. The rest is on us, developers. Are we complying? I must say, no. Would you believe that most Web sites don't follow the latest HTML standards? I randomly checked the opening pages of some high traffic sites. Here is the result:

Non-compliant	**HTML 4 compliant**
Yahoo.com	
AOL.com	
Google.com	
CNN.com	
Netscape.com	
Amazon.com	
eBay.com	
Expedia.com	
MSN.com	
Microsoft.com	
ESPN.com	

Can you find any compliant sites? Neither can I. Last I tried the official Web site of the President of the United States: http://www.whitehouse.gov/. Guess what, he failed, too. But we can't expect George W to be HTML compliant before Bill Gates.

AOL is the only one with a doctype claiming that its opening page is XHTML compliant but it is not. Bigger mouth, same sloppiness. It does not even follow the less strict, HTML 4 standards. I believe that these companies are just too big to be good. OK, their revenues are breathtaking but can they produce a standard HTML page? After all, they **are** the Web; they transact at least half of all the traffic on the Internet. What about you? Do you think that the only way to get rich is to neglect the rules? Be a man (or a person, politically correcting myself, although it does not make sense) with self-respect and show the big shot organizations that you are better than them. You can develop W3C compliant applications. For one, this is the way to create cross-browser pages. Even if your boss says that the company uses only IE 6, and you don't have to care about other browsers, you should follow the W3C standards. And those rules that I collected in *The Web Developer's Ten Commandments.* I believe that you will make your life easier. And that of your customers. And that of your colleagues. And the day will come when your employer will be thankful to you. Neah! Too much of dreaming.

The art of cross-browser scripting

In the beginning there was chaos. We developed a nice solution for one browser, then another browser displayed it wrong. We created another solution for the other browser, programmatically checked the type of the actual browser and depending on the type, branched the execution to one or to the other solution. Then the first browser came up with a new version that worked with the solution written for the second browser. The branching must have been replaced with one that contained version numbers, too. The Internet was crowded with browser sniffers; codes that detect which browser version is viewing the page.

Microsoft has been pushy in introducing proprietary objects and properties in IE, such as `document.all` and `innerHTML`. You can avoid using them but you shouldn't necessarily oppose everything that is not part of the W3C standards. The first one has the practical drawback that no other browsers accepted it. The second one carries an excellent idea. The only downside of `innerHTML` is that someone else invented it, not me. Although it is not part of the W3C DOM, Mozilla- and Gecko-based browsers (such as Firefox and Netscape 6) decided to support it. Most browsers newer than May 2000 understand `innerHTML` and you can use it safely.

The situation is clearer nowadays. We don't have to check which one of the numerous browser versions we actually serve. We rather check for functionality. For example, if we want to get or set the value of an object, we use the `getElementById()` method in most browsers. Apparently, I don't deal with those older browsers anymore that don't understand the `getElementById()` method. It works since the introduction of IE5, NS6, and Opera6. Once a feature survives two version changes of three browsers each, chances are that it will stay for a while. Instead of keeping track of browsers and their versions, we may simply check if the used function works in the running browser. Good developers don't sniff for type and version; they test the browser with a code similar to this:

```
if ( isNull(document.getElementById(id)) ) {
  alert('Go get a newer browser that understands
getElementById()')}
else {// do the getting and the setting of values
here}
```

Of course, `id` has to be the identifier of an existent object, for example, the `body` of the page. I prefer to run this test at the opening page of an application once, because I can give the warning at the beginning; so, I don't have to run the test over and over again. In some applications, like in an eBook, the author should be prepared that the visitor does not start the reading from the front page. Some like it from the back; others jump in the middle accidentally or intentionally, hoping that there is a centerfold picture there. In these cases, one needs to run the test at every occurrence of the questionable element.

With my present awareness I am supporting IE version 5.5+, Netscape version 6.0+ and Opera version 7+. When I say Netscape, I mean Mozilla throughout these articles. Recent Netscape versions are built on the Mozilla open-source code, as well as many other new browsers, like FireFox. I hope that their common core can assure us that if a solution works in the latest Netscape, it works in other Mozilla-based browsers.

There are still interpretation differences among Internet Explorer, Netscape and Opera. Let's take the treatment of spacing. Netscape and Internet Explorer give the `body` tag a default `margin` of 8 pixels but Opera does not. Instead, Opera applies a default `padding` of 8 pixels, so if one wants to adjust the margin for an entire page and have it display correctly in Opera, the `body` padding must be set as well. In general, never let the default values kick in. Always set them to your taste. Well, this is easy to say but difficult to do. Not only their default values, but the interpretation of spacing between and around objects also distinguishes browsers. The spaces occupied by padding, border and margin are different in different browsers. Their sizes are usually added to the surrounded object's width and height but sometimes deducted. If we also take the scroll bars into account, which have non-adjustable various sizes in various browsers and the sizes of which sometimes are added, sometimes deducted to/from the scrollable objects' sizes, the mess can be big. Internet Explorer 5.x is famous for its broken box model. Padding and border are supposed to be applied outside the box but in IE 5.x they are inside. I am trying to sidestep rather than fight the problem. I consider myself a perfectionist but I would not waste my time for a pixel-perfect cross-browser layout. When the different arrangements all look good in three major browsers, I stop tinkering the layout and I don't try to fix what is not broke. My advice is that try to set all the differently handled spacing to zero or small if possible and avoid designs where displacements by a few pixels add up and become critical. A table row with 20 cells will be off 160 pixels at the end of the row, if a two-pixel padding of the individual cell is not added but deducted.

As a last resort, you need to write various style sheets for various browsers and to go back to the world of browser sniffing. You will find many code samples on the Net. I am afraid that too many, compared to the importance of the issue. If you find your page similar in three major browsers, you may grant yourself the cross-browser compliant award and display its icon as I did below.

When Sniffing Is a Must

Browser sniffing is necessary when you cannot create a general solution that works in every browser you consider.

So far, I could overcome the differences among the layouts rendered by different browsers, without handling them differently. I met one problem where I could not avoid the various behaviors of the various browsers, though. I have created a few Web applications that open XML data files. XML is a nice standard for carrying data but loading an XML data file into HTML is still a challenge. Internet Explorer has a relatively simple method, using the Microsoft-only ActiveX object. Unfortunately, it does not work in its own IE for Mac. OK, we can forget those few twisted who deny the word of PC's and Windows but stick to Internet Explorer on a Mac. I have a good solution that works in Mozilla browsers, like in Netscape 7 and in FireFox. None of the above works in Opera. Should we neglect the Opera fans? No way! There is a solution for Opera, which actually works in the other two major browser groups, too, even in IE for Mac. It also has its drawbacks but it helped me to complete the jigsaw puzzle. There is cross-browser and cross-platform solution to loading XML data. The discussion of this solution needs a separate book, or at least a separate article. And I am working on it...

Most visitors don't like
Web sites that attack them
with multimedia
without notice...

How To Play Sound

Intro

So, you want the visitor to listen to your music, no matter what kind of operating system, browser, media player he/she has. You've already tried various solutions found on the Internet from Mickey Mouse to Overkill Guru, but not one worked exactly the way a sound playing Web page is supposed to work.

This article shows you how to play sound on all major browsers and platforms, without having any extra Web development environment other than a text editor.

What can a visitor expect?

First of all, the solution should work, regardless of the computer's operating system (Unix/ Linux/ Mac OS/ Windows 2000/XP, etc.), of the browser used (Internet Explorer, Netscape, FireFox, Opera, Safari), and of the media player installed (Windows Media Player, QuickTime, RealOne Player, WinAmp).

Most visitors don't like Web sites that attack them with multimedia without notice. They want to control audio and video. The developer must give them the opportunity not to experience what they think they wouldn't enjoy. They expect a textual suggestion and a button to start the play, if and when they want it.

On the other hand, when they decide to try it, they want it immediately. It is not acceptable that the visitor has to wait 2 minutes after clicking and before listening to a three-minute song.

Most visitors won't reset or change browsers, download and configure plug-ins, and then revisit your Web site. This kind of advice is the last one your visitor reads from you. It may be a correct advice, still, not accepted. When someone sits in front of a new computer, and wants to listen to the music immediately, would he accept the otherwise reasonable advice that he should stand up and turn on the radio? Then and there he may want more freedom and choice than what a radio can offer. Some are willing to lose comfort and waste time for freedom, some don't.

Most computer users have relatively new, (not older then two years) versions of an operating system, of a multimedia player, and of a browser. Don't expect them to have a particular one or the very latest one. Don't expect them to have a Flash player. I happen to have one, but I disabled it because of the frequent abuses of Flash. One must watch too many senseless, eye-infecting videos to find some good ones.

Historical Solutions

Basically, four HTML tags can make your page play sound; the BGSOUND background sound tag, the A hyperlink, the EMBED, and the OBJECT tags. Other solutions involve either Javascript programming or even more sophisticated approaches. Let's deal with these HTML solutions first, then with a simple script.

1. Background sound BGSOUND

Form:

```
<bgsound src="sample.wav" loop="-1">
```

BGSOUND is an Internet Explorer-only, non-standard tag. So, if you say "Forget about it" now, you are right. But there are more arguments against it. It loads the sound file when the page is loaded and after the full download, it starts to play without user intervention. The loop="-1" attribute makes the sound replay over and over. This is one of the most annoying solutions, so, I don't create an example here, as I do in the following solutions. It doesn't give the visitor any control. It's only "advantage" is that, unlike in some other sophisticated solutions, the sound stops immediately when the horrified visitor clicks away from the site.

2. Hyperlink A

Form:

```
<a href="sample.wav">Play Sample</a>
```

This is the simplest way and it almost always works with the three major browsers, Internet Explorer, Mozilla (Netscape/FireFox), and Opera. With most browser settings and file formats, it launches a media player. If no player application has been installed, the visitor probably has not played sound on this computer before. A reasonable visitor will not blame you that your site is the one that fails. The real drawback of this solution is that the sound usually doesn't start playing until it has been completely downloaded, and the downloading doesn't start until one clicks on the link. Some browsers combined with some media player applications try to stream the sound; i.e., start to play the sound before it is fully downloaded but it may stall later.

3. EMBED

Form:

```
<embed src="sample.wav" autostart="false"
loop="false">
</embed>
```

The EMBED tag causes the sound file to be downloaded when the page is loaded, just like an IMG tag makes an image load. The browser then looks for a plug-in or for a built-in player. Internet Explorer has the least problem because its Windows Media Player is integrated in its newer versions. Mozilla prefers Quick Time as media player but it is an external application. RealOne (Real Audio Player) or WinAmp are alternative media players to plug in. EMBED is not a standard HTML or XHTML element, either, but all major browsers handle it very well.

This solution may not work on a networked computer in certain network settings and may cause an "Unable to establish connection to the server" message popping up. You may also need to click the control more than once to make it start.

4. OBJECT

Form:

```
<object height="50%" width="50%"
classid="clsid:22D6F312-B0F6-11D0-94AB-
0080C74C7E95">
<param name="AutoStart" value="1" />
<param name="FileName" value="sample.wav" />
</object>
```

The World Wide Web Consortium (W3C) recommends using the OBJECT element instead of EMBED. Yet I discuss the solution with EMBED here, because I couldn't create an OBJECT that was as user-friendly, cross-browser, cross-platform, less plug-in dependent, as the solution with EMBED.

And The Winner Is...

So far the streaming effect has not been discussed. All of the above mentioned media players are capable to start playing before the whole sound file is downloaded. However, the commonly used file formats, .wav, .mid, .mp3 don't tell the player to stream. Fortunately, the format m3u does. Unfortunately, browsers don't necessarily know what to do with an m3u file, unlike, for example, with an .mp3 sound file or with a .gif image file. You may be as hesitant as the browsers, so, I let you know what m3u file is. M3U is a media queue format, also known as a playlist. In its simplest format it is a list of sound file names, separated with new lines. Here is an example:

```
Sample.mp3
music/Song.mp3
http://www.juancarlosproductions.com/music/mystaSam
ple.mp3
```

The first line refers to a sound file being in the same folder as the m3u file, the second to one being in a music subfolder, and the third to a sound file on the Net. The form of first line is the shortest, but of the last line is the safest. Let's save a text file that contains the third line only, or a similar reference to a sound file, as sample.m3u, and invoke it from an EMBED tag:

```
<embed src="sample.m3u" type="application/x-
mplayer2"></embed>
```

Interestingly, m3u has its dedicated MIME type, audio/x-mpegurl, but it doesn't always work. In default setting, FireFox for Windows doesn't recognize audio/mpegurl (or audio/x-mpegurl). An in-depth recent article on boutell.com suggests to refer to the m3u as "audio/mpeg" type. According to Thomas Boutell, this type calls the correct media player assigned to the m3u type files. In my experience, it doesn't work with the default settings of Opera in Win 2000 or in XP, but the type="application/x-mplayer2" does. In Windows XP the latter type correctly invokes the default media player of Windows from FireFox, too. I also experienced that the attribute autostart="0" of the EMBED tag works in more cases than the substantially equivalent autostart="false". My suggestion is to call the m3u play list in the following EMBED tag:

```
<embed src="sample.m3u" height="45" width="170"
    type="application/x-mplayer2" autostart="0"
    loop="0" volume="-300"></embed>
```

The above one-line HTML solution works with the default setup of IE6, Opera 7.21, FireFox 1.06 (Mozilla 5), Netscape 7.1 on Windows 2000 and XP; and sometimes with Safari on Mac OS. To make it always work with Mac/Safari and with FireFox on Linux and with other Mozilla versions in non-Windows platform, I created a Javascript. It checks and resets the MIME type of the embed tag to "audio/mpeg", if necessary.

So, I am sharing the general solution with you. It is not too difficult, although requires some understanding of Javascript.

General solution

To make pages context-sensitive, we need to add scripts to the HTML. The following solution doesn't require any special configuration of the Web server or of the client's computer. It doesn't contain long sniffing code for finding the platform and the browser version of the visitor's computer. With this Javascript we can check the available plugin and create an EMBED tag with proper MIME type.

Let's first put a SPAN in the HTML page, where we want the EMBED tag later:

```
<span id="sample">Finding plugin...</span>
```

Note that I'm using the m3u file's name as the span's id. The following function finds the SPAN with a given id and inserts the EMBED tag into the SPAN, by replacing the "Finding plugin..." text:

```
function setEmbed(ID, dir) {
    var element = document.getElementById(ID);
    //Write the following three lines in one:
    element.innerHTML = '<embed
src="'+dir+ID+'.m3u"
      autostart="0" loop="0" height="45"
width="170"
      type="'+getMimeType()+'"></embed>';
}// end function setEmbed
```

The getElementById function works in all modern browsers. If you want to take care of the less up-to-date visitors, read the ***One Code Fits All*** article. The setEmbed function calls the getMimeType function, which returns the proper MIME type:

```
function getMimeType(){
var mimeType = "application/x-mplayer2"; //default
var agt=navigator.userAgent.toLowerCase();
if (navigator.mimeTypes &&
agt.indexOf("windows")==-1) {
//non-IE, no-Windows
  var
plugin=navigator.mimeTypes["audio/mpeg"].enabledPlu
gin;
  if (plugin) mimeType="audio/mpeg" //Mac/Safari &
Linux/FFox
}//end no-Windows
return mimeType
}//end function getMimeType
```

The navigator.mimeTypes object returns false in Internet Explorer, and the second condition makes sure that the platform is not Windows. The setting to default MIME type doesn't need changes in any browsers running on Windows 2000/XP or in IE on any platform. Obviously, this function can be refined and extended, if one wants to consider more situations, more MIME types. I, for one, haven't checked any Unix setup yet but some of my readers have. For example, a Unix machine with FireFox and installed mplayer plug-in played the embedded play list flawlessly.

The above two functions should be placed in the HEAD section between a

`<script language="JavaScript" type="text/javascript">`

and a

`</script>`.

The best place and time to invoke the setEmbed function is in the BODY tag after page load:

```
<body onload=
  "setEmbed('sample',
'http://www.yourdomain.com/music/')">
```

The above code replaces the "Finding plugin..." text with the media player's control buttons after the page has been loaded and the proper player has been found. Then a little extra time is needed to load and buffer the beginning of the sound. During these few seconds the Play button looks grey, i.e., disabled. Clicking on the later enabled Play button, it will immediately start playing the sound file listed in the sample.m3u play list, which resides in the music subdirectory of yourdomain.com.

Summary

A simple HTML solution has been suggested for playing audio. This solution covers at least 90% of the users; those who use almost any browsers in Windows 2000 or XP. By adding a dozen lines of Javascript we can also reach the growing camp of users, who utilize other configurations, like FireFox in Unix/Linux or Safari in Mac OS.

PART 3. AJAX IN ACTION

What about repetitive HTML sections?
AJAX verbalized that the Emperor was naked.

Making The Fastest Photo Album

Intro

AJAX is not really a new technology. It revisits accepted principles and approaches, combines others, and puts good old technical tricks in one paradigm. When we review the web sites created by regular development tools, we must admit that their architecture is far from optimum regarding response speed. AJAX helps us create faster sites by eliminating unnecessary round trips to the server.

Since the industry got obsessed with drag-and-drop, point-and-click development, the quality of the resulting code has became marginal. Microsoft's FrontPage or Visual Studio, Macromedia's Dreamweaver or ColdFusion are good products in a sense that non-technical people can develop good-looking web sites with them. If someone can edit a document in Word, she can make an HTML by clicking on the "Save as a web page". What is common in the web pages created with the above WYSIWYG tools, or with many other, so-called RAD tools? They all are lengthy and redundant. They don't consider limited band width, readability or optimization for the speed of displaying pages. People don't have fast enough connections for these long pages? Have the visitors buy a faster modem, subscribe a faster Internet provider and have the web site owner move to a faster web server. Since the existence of CSS it is obvious: there is no place for repetitive styling in the HTML lines. But what about repetitive HTML sections?

BTDT, The Photo Album Application

Server-side scripting has a serious problem. It creates and sends repetitive HTML. The script itself is not necessarily redundant but the resulting page is. A script that creates three rows and four columns of complex cells generates a table of twelve cells. The internal structure of the twelve cells is spelled out twelve times in the transferred HTML, although only one relatively small difference may occur among the individual cells. One may argue that repetitive HTML is necessary because one cannot create a loop in HTML. It is true but it does not keep us from running a loop on the client-side that creates an HTML of repetitive structure. Instead of a multi-second download, we can run a client-side script in milliseconds.

AJAX verbalized that the Emperor was naked. Or better to say, he was overdressed. Why should we retransfer the same page with minor differences and the same structure inside a page multiple times? Take a typical application, an online photo album as an example. When the user wants to see a thumbnail view, we may not need to send any HTML from the server at all because we can create the structure and most of the content on the client's machine.

There are dozens of applications that create photo albums. They all compete in appearance but don’t care much about the optimization of network traffic. A typical developer tool gives a set of choices in colors and shapes. The better ones collect the user’s choices and produce a CSS file according to the chosen “skin”. Then they create as many alike static HTML pages as many photos are included. Clicking the Next or Previous button results not just in the loading of a new picture but the loading of a similar HTML page, with two differences. The obvious one is that the source attribute of the IMG tag has been changed to the name or URL of the new picture file. The second difference is that the new state of the site has also been stored somewhere. Either on the server or on the client-side, for example in a cookie, in global variables, or in a search string of the URL.

What is the cost of sending pages and structures with minor differences? One of the best photo album creator software is Jalbum. It offers beautiful skins, fast and automatic creation of slides and thumbnail pictures. However, the resulting site contains as many similar, 10k HTML pages as many pictures the album has. An album of only 10 pictures contains 100k repetitive HTML.

In the following I will show a solution that uses less HTML, a bit more client-side scripts but no server-side scripting. It has one single HTML file of about 10k. When we want to show two times as many pictures, Jalbum almost doubles the size of its index.html and it doubles the number of its 10k HTML pages. Even worse, at every mouse click one of these 10k files will be downloaded. The size of my solution is independent of the number of pictures and it does not request a new HTML page at each click.

AJAX suggests that the site should store its state on the client-side in a hidden frame or in a hidden iframe. It also suggests that reusable information, like the file name list of pictures should also be stored there. In this case, download of a new page is not necessary when changing an image or text content. In the photo album example, clicking the Next or Previous button would invoke the dynamic change of the image source of the same page. This way the client requests a download of another image only, if the image is not in the cache already. Not even a click requesting the thumbnail view invokes a new HTML load. You have five or fifty albums? The same small HTML page can display all of them.

Here are the snippets completing the above tasks:

1. The IMG tag in the HTML file:

```
<img id="Img0" class="image" src="url or file name of
opening picture" alt="picture" />
```

2. The Javascript codes that handle the dynamic change of the picture:

a) The declaration of an onclick event function of the buttons:

```
objButton.onclick=function() {setImage(albumNo,
picNo)};
```

b) The setImage function:

```
function setImage(albumNo, picNo, imgId) {
   if (isNull(imgId))   imgId = "Img0";
   if (isNull(picNo))   picNo = defaultPicNo;//global
   if (isNull(albumNo)) albumNo = defaultAlbumNo;
   var oImg0 = getObj(imgId);
   oImg0.src="images/"+arrNames[albumNo][0]+"/" +
           arrNames[albumNo][picNo];
//… Maintain state here …
   return true
}
```

The first three lines implement a simple polymorphism. They ensure that setImage can be called with zero to three arguments. For example, `setImage( )` without an argument can display the default picture, say, when the visitor clicks the Home link. In this case no page reload should occur as opposed to 99.9% of web sites.

`setImage` calls two simple, general-purpose, cross-browser functions that speak for themselves:

```
function isNull(obj) {//Check undefined explicitly
(NS6.x)
  return(obj==null)||(obj==undefined)
}
```

```
function getObj(id, doc) {//doc can be an XML
document object
 if (isNull(doc)) doc = document;
 with (doc) {
  if(!isNull(getElementById(id))) return
getElementById(id);
  if(!isNull(getElementsByName(id)))
                             return
getElementsByName(id);
  if(!isNull(getElementsByTagName(id)))
                             return
getElementsByTagName(id)
  else return false
 }
}
```

The second function is also polymorph in the above sense. With one argument, it returns an object of the window document. Giving an XML document as the second argument can be handy when one retrieves the directory names and file names from an XML file. My photo album site handles multiple albums. After I read the XML file of all album names and picture names, I store them in a two-dimensional array arrNames[ ][ ]. If you are not familiar with XML, you may hard-code this array the following way. The 0-th element of the array representing the i-th album is the name of subdirectory in which the pictures of the particular album reside:

```
arrNames[i][0] = "name of subdirectory of album i"
arrNames[i][j] = "name of picture j in album i" (j>0)
```

With the above design a single page load will handle the functionality of browsing through several albums in single view and in thumbnail view. This architecture also allows us to play a slide show without loading new pages, by timing and imitating clicks on the Next button periodically.

You can check out my photo albums at scriptwell.net/myPhotos. They are not as nice (yet) as those generated by Jalbum but definitely faster.

The next article, AJAX In Action (2): Fixing The Broken Bookmark solves a common problem of partial change of page contents. Then I'll show you how the slide show is implemented and how the thumbnail view can be handled without loading a new page. Another article is about speeding up the Photo Album Application by proper preload technique.

You can find the complete HTML, Javascript and CSS lists of the Photo Album Application in the last article of this series.

Abstract

This article shows how to start creating a complex photo album site that requests minimal downloads from the server. Instead of loading several pages, requested updates take place on the client-side via dynamic changes of a single page. At the end of the series the reader will have a fully functional, cross-browser, fast and compact photo album application that does not even need a web server.

? Questions

1. The usual way to visit a site is to go to the opening page or home page first. So, before the visitor clicks the Home link somewhere in your site, the opening page has already been visited. Does your web site reuses the previously downloaded opening page in this case or does it request a reload from the server?

2. Have you created or have you used a photo album creator? Does it change the displayed photo by dynamic change of the source of the IMG tag or by loading another HTML page?

It's the hash, baby!

Fixing the Broken Bookmark

Intro

AJAX is a technique that enables us to create web pages with changing segments. The need for such pages brought up the concept of framesets in the early years of Internet. One can easily change a segment of a page by changing the content of one frame and leaving the others intact. A major drawback of this technique is that the URL usually refers to the default opening state of the frameset. When you are not aware of browsing in a frameset, find something interesting, bookmark it or send the URL to your friend, you or your friend will not get back the same content what you saw at the time of copying the URL. In an article, Got Lost on the Web I gave a solution to that problem. The main idea has been that a manipulated search string of the URL can reflect the state of a page. (For sake of clarity, search string is also called query string in the literature.)

The same exact technique does not work in our case. Remember, we want to eliminate unnecessary round trips to the server. One of the reasons of changing partial content with AJAX is that we don't want to request a new page or to reload the present page. However, changing the search string invokes a page reload. This has not been a problem with framesets because there we always wanted to change the content by requesting and loading a new HTML in a frame. So, what is the solution in case of single-document pages? What part of the URL can we change without the consequences of requesting a load? It's the hash!

What Is Hash and How To Use It?

No, it's not what you think. This hash you cannot smoke. Hash is the part of URL after a # hash mark. It's original use is to navigate inside a page. The

```
 href="www.mypage.html#middle"
```

attribute links to the said page and scrolls to the section starting with the anchor <a name="middle"> or with <a id="middle">. (The latter is better: the name attribute has already been deprecated in HTML 4 and future XHTML versions will completely eliminate it.) The good news is that one can set the hash property without invoking a load or navigating inside the page. If there is no anchor named "middle", the browser shows the top of the page.

With the above features of hash we can keep track of the page's status. Every time we change the page content dynamically, we can change the hash as well. In the Photo Album application the actually displayed photo corresponds to an album number and a picture number. The snippet that writes these numbers in the hash is as follows:

```
 function SetPresentState(albumNo, picNo) {
  if (isNull(picNo)) picNo = 0;
  location.hash = albumNo + ";" + picNo;
}
```

The picNo = 0 state refers to the thumbnail view of an album. SetPresentState should be called every time the content changes, i.e., when we set the image source and when we switch to a thumbnail view. Then the URL corresponds with the page state. When the page shows the 3d album's pictures in thumbnail view, the URL ends with index.html#3;0 and when it shows the 4th picture of the 1st album, it ends with index.html#1;4. We can bookmark or send these URL's to a friend.

How To Retrieve The Page?

The remaining problem is that how we can reconstruct the page from the above URL. By default it makes index.html load and tries to find an anchor inside with the name or id of "3;0" or "1;4". These anchors don't exist, so, the page stays on its top. However, the following code in the HEAD enables the page to appropriately set the content both in thumbnail view and in single view:

```
 //Constants:
var ALBUMNO = 1;//default album number
var PICNO = 1;//default picture number
//Global vars:
var g_albumNo = GetPresentState(1);
var g_picNo = GetPresentState(2);
if (g_albumNo + g_picNo < 1) {//hash is empty
  g_albumNo = ALBUMNO;
  g_picNo = PICNO;
}
```

```
function GetPresentState(paramNo) {
  return Number(parseString(paramNo));
}
```

```
function parseString(n){//N>=n>0;
//Form of str: param1;param2;...;paramN
// Returns nth param of the string.
  var delim = ';'
  var str = getHash();//hash string w/o the '#'
  if (str.charAt(str.length-1)!=delim) str += delim;
  var i = n;
  while(i>0) {
    retVal = str.substring(0,str.indexOf(delim));

str=str.substring(retVal.length+1,str.length);//cut
1st
    i--;
  }
  return retVal;
}//end function parseString
```

```
function getHash() {
  var str = document.location.hash;
  return str.substring(1,str.length);//cut the '#'
}//end function getHash()
```

parseString is a simplified version of a more general function. The original one can handle various types of strings with various delimiter characters and of various origins. You can find the complete HTML, Javascript and CSS lists of the Photo Album Application in the last article of this part.

Summary

This article gives a solution to a common problem of AJAX: when the content of a page changes without page load, the URL does not necessarily reflect the state of the page.

When web developers become envy of desktop developers, wonderful things happen!

Slide Show With Asynchronous Calls

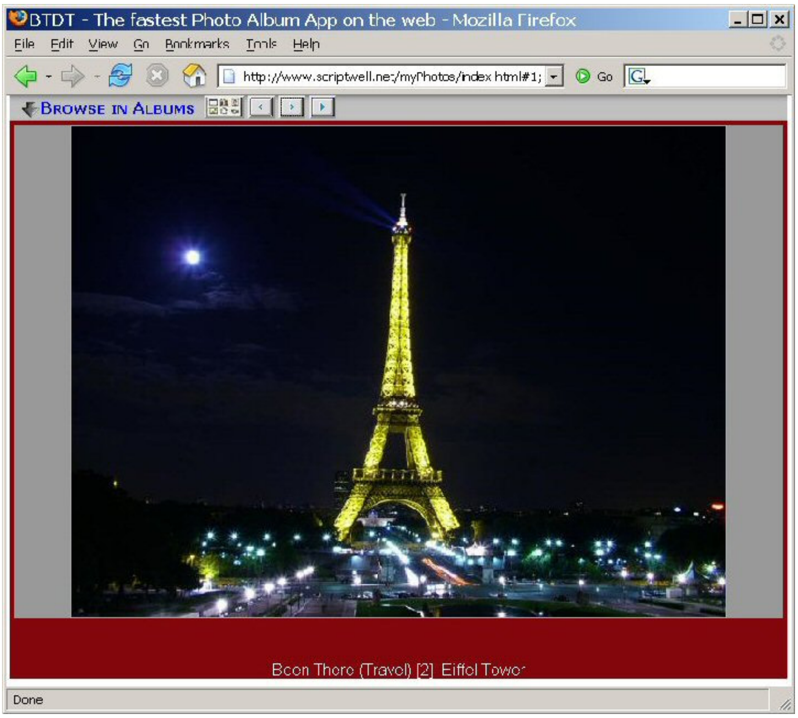
Intro

In the early computer age programs executed procedures one after the other. The asynchronous approach enables us to trigger new procedures while old ones are still running. In many cases we don't have to care about parallel runs of procedures because the individual actions finish so quickly that the sequential processing is not noticeable. However, we must arrange parallel processing when a process takes long or when we want to time and delay processes. In desktop applications it is obvious that calculations continue, data streams flow, and clocks are ticking while we do something else on the computer. Web developers became envy of desktop developers, and when we get envy, wonderful things happen!

Many web applications don't work with sequential processing. In some cases the user experiences only annoying effects, like temporary space holders instead of pictures. In other cases the web application crashes because it expects downloaded elements which are not there yet. For the latter I will show an example in one of the next articles, where I discuss the handling of XML data that carry the names and descriptions of the pictures.

Buttons Of The Photo Album Application

Here is a screen shot of BTDT:


BTDT - The fastest Photo Album App on the web - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.scriptwell.net/myPhotos/index.html#1;
Go
Browse in Albums
Been There (Travel) [2] Eiffel Tower
Done

Four buttons are visible: the thumbnail view, the previous, the next, and the slide show buttons. The buttons are INPUT tags:

```
<input type="button" title="Thumbnail View" id="btThumb"
  onclick="thumbNails(GetPresentState(1))" />
<input type="button" title="Previous" id="btPrev"
  onclick="Prev()" />
<input type="button" title="Next" id="btNext"
  onclick="Next();" />
<input type="button" title="Start Slide Show" id="btPlaySlide"
  onclick="SlideShow()" />
```

The title attribute provides a nice effect. Its value pops up when the mouse goes over the button. In this article we discuss two functions that are called by clicking on the previous and on the next buttons:

```
function Prev() {
  if (g_Play) SlideShow(STOP);
  var albumNo = GetPresentState(1);
  var picNo = GetPresentState(2);
  if (picNo==0){  //thumbnail view of next album
    albumNo--;
    thumbNails(albumNo)
  }
  else {
    picNo--;
    setImage(albumNo,picNo);
  }
}//end function Prev

function Next(timeLeft) {//Called with arg
programmatically
  if (isNull(timeLeft)) SlideShow(STOP)
  else timeLeft--;
  if (timeLeft < 0) {  //set next image immediately
    var albumNo = GetPresentState(1);
    var picNo = GetPresentState(2);
    if (picNo==0){  //thumbnail view of next album
      albumNo++;
      thumbNails(albumNo)
    }
    else {//next single slide
      picNo++;
      if (picNo>=arrMenus[albumNo].length) picNo=1;
      setImage(albumNo,picNo);
      timeLeft = g_delayTime;
      if (g_Play) Next(timeLeft);
    }
  }
  else {//after 1 sec(=10000 msec) call Next again
    tout = setTimeout('Next(timeLeft--)', 1000);
  }
}//end function Next
```

The first line of `Prev()` (and of `Next()`) is obvious. It stops the slide show. The second and third lines read the state variables, discussed in the previous article. The following branching shows the dual task of the previous button: When we are in thumbnail view, the click brings up the thumbnail view of the previous album. When we are in single picture view, the click calls the previous picture of the same album.

Function `Next` could have been similar to function `Prev` with the minor difference that `albumNo++` and `picNo++` increment operations would replace the `albumNo--` and `picNo--` decrement operations. `Next()` without a parameter basically does the same as `Prev()`, in opposit direction. However, the slide show is implemented the way that function `Next` is called with a `timeLeft` parameter and with a global Boolean variable `g_Play` that is set to true. It recursively calls itself until the user hits a button.

How The Slide Show Works

The key statement of the slide show is the last line of function `Next`. The function `setTimeout()` is one of the most important functions in asynchronous Javascript programming. Its first parameter tells which function should be called later, and the second parameter gives the definition of 'later' in milliseconds. In our example `Next` calls itself recursively, with decrementing parameter. When the parameter becomes negative, `Next` behaves as if it was called by a regular click. If nothing happens in the meantime, `g_Play` stays true. Right after setting the new image, it also resets the waiting time, and the process starts again.

Summary

This article describes the evolution of the Photo Album Application's Next button. The slide show has been implemented as an imitation of periodic clicks on the Next button.

Preloading Images

Intro

The idea of preloading images is almost as old as the Internet. Yet, many misconceptions are circulating in the web developers' community. The basic problem is that downloading large files, mostly pictures and other multimedia files, may last longer than your visitor's patience. For movie or music files continuous streaming is the solution, when, instead of downloading the complete file before playing it, loading and playing occur simultaneously. In a previous article I discussed some possible solutions to play sound on the web. Here we only concentrate on how to display image files fast.

The concept of preloading is based on the idea that large or repeatedly used files should be downloaded in advance, held in memory or at least on the client's hard drive. This way the delay of displaying them would not be noticeable. A typical application of this idea is the switching of images upon the `onmouseover` and `onmouseout` events. The switch must be prompt. If the images are relatively small, default browser settings may automatically handle the local storage of the pictures properly. In the BTDT application the picture of a closed folder is supposed to change, i.e., open when the user moves the cursor over it and close again when the cursor leaves the picture.

Possible Solutions

The changing image is accomplished with the following snippet:

```
<img alt="folder" src="images/folder_closed.gif"
  onmouseover='this.src="images/folder_open.gif"'
  onmouseout ='this.src="images/folder_closed.gif"'
/>
```

It should work fine without preload, with the default settings of Internet Explorer or Firefox. If the pictures don't alternate fast for you, you or somebody else may have changed the browser's setting. You, as a developer, can hope for the best but you should prepare your web application for the worst. You cannot underestimate the user's creativity in messing up default settings. If, for example, the user sets IE's cache size to minimum and the "Check for newer versions of stored pages" to "Every visit to the page", not much playground is left for the web developer. Lately I gave up handling extreme users. I usually issue a warning message to those, who disable Javascript or use ancient browser versions but I'm not willing to make extra efforts to serve them the same way as the rest of the word, which luckily makes up at least 90% of the users. However, you should prepare your app for handling normal situations, like preloading larger pictures that would not be kept automatically in the client's hard drive or memory.

Here is the preloading version of the previous example. The following snippet should be placed in the HEAD section:

```
<script type="text/Javascript">
  imgArray = new Array();
  imgArray[0] = new Image();
  imgArray[1] = new Image();
  imgArray[0].src = "images/folder_closed.gif";
  imgArray[1].src = "images/folder_open.gif";
</script>
```

Misconceptions

Because the code is invoked in the HEAD, the preloading snippet is executed before the page load, right? Well, almost. The execution starts but it may not finish before the page load. It depends on the size of the preloaded files, on the connection speed and on the type and configuration of the browser. In case of two small pictures, like in the above example, the preload occurs on time. This way the "onmousover" and "onmousout" events will work smoothly upon the image in the BODY:

```
<img alt="folder" src="images/folder_closed.gif"
 onmouseover='this.src=imgArray[1].src'
 onmouseout='this.src=imgArray[0].src' />
```

Is this solution always appropriate? Let's say we have 20 pictures, 25k each, in a photo album. That's half a megabyte to load. Also, assume that preload works as expected, i.e., it finishes before the page load starts. The creator of the page can assume that the pictures are there when needed. However, the visitor does not see anything until the preload is complete. If we used the above technique, the preload would take at least 10 seconds even with a fast Internet connection. 10 seconds is a critical waiting time for the visitors. Many of them will not watch an empty window that long and they leave the page before the preload is complete.

Most tutorials advise to preload all pictures before loading the page. A more intelligent approach is to preload only those pictures that should be seen first, and load the rest later. This can be an asynchronous process. While the visitor watches the present page, a timed process can preload the files that are necessary for the next state of the page. The next state can be the result of a scroll-down or of a click on a button, and we should preload the forthcoming picture(s) before these events but after the load of the page. Our case with the photo album is simple, and we don't have to take care of an additional timing of the preloading process. The preload will be triggered by a user intervention or by another, automated asynchronous process. The Next function that sets up the asynchronous process is discussed in the previous article of this series.

The Been There Done That Photo Album preloads only one picture at a time, namely, the one that would be displayed after the most likely actions, the click on the Next button. Since the slide show uses the same function to automatically change the picture, this preloading technique works during the slide show, as well. The preload function, of course, is located in the HEAD, but its execution happens whenever a new image is set:

```
 function preload(albumNo, picNo) {
  if (isNull(imageArray[albumNo][picNo])) {//not
preloaded yet
    imageArray[albumNo][picNo] = new Image();
    imageArray[albumNo][picNo].src = 'image file name
...';
  }
}

function setImage(albumNo,picNo){
//Displays the currently asked image
...
//Preloads the next image
  picNo++;
  if (picNo<numberOfPictures) preload(albumNo,
picNo);
}//end function setImage
```

The complete code is available from the BTDT web site.

Summary

In case of large downloadable files, the developer cannot assume that the files will be at the client's computer when needed. Preloading may make an application faster or slower. Both premature preload and negligation of preload can cause problems. The problem may require an asynchronous approach by its nature. One should not rely on default settings and features and must take care of the sequence of processes. The demonstrated solution is a good example of situations where we can trigger a process synchronously, i.e., sequentially after an asynchronous process.

Web marketers and drug dealers have similar methods, only that of the latter are always illegal.

Of Web Marketing and Other Cyber Terrorism

Intro

It is all about security holes, businesses and crooks that hide and exploit these holes; malicious and benign scripts; the thin line between the criminal and the legal but immoral activities.

Fine print

```
Some findings of this study have not been published,
due to their sensitive nature and their possible
volatile use. If you are not a terrorist or a
marketer, you may contact the author via email below
for technical details.
If I hurt your political sensitivity, you may also
write me and initiate an open discussion in a polite
manner, or keep your opinion, suck it up and deal
with it. If you are a Web marketer, and don't like
what I wrote about the nature of your business, I'm
still willing to discuss issues with you, assuming
benevolence. However, don't try to abuse my Web site
or my email address, don't send solicitations, junk
mail, spam, hate mail, etc. The consequences of your
malice will be serious and stronger than what you can
expect:
I'll use my rights of free speech against yours. You
may legally take away my resources, my storage space,
my bandwidth, my time. I will morally make your
business impossible with the power of bad publicity.
Don't pick a fight with me. You better keep your junk
to yourself.
```

Zero Degree Crime: Marketing

The goal of marketing is to emphasize the attractive features and hide the repelling ones of a business. White lies are common in marketing, frequently outside the ethical boundaries but mostly in the frame of legality. Most people know that marketing is about illusions as opposed to truth, misinterpretation as opposed to honest knowledge sharing, and about sliding facts. Yet few say when they talk about business as usual: "Show me a liar and I'll show you a thief". Why? Because lying in marketing is legally and publicly accepted; it creates jobs and it improves the economy. Stealing, on the other hand, is a criminal activity. If the first saying was true, then another one was an obvious consequence: "Show me a successful business and I'll show you a liar and a thief." Morally, a marketer is a thief when he takes away business from a better product or service, with the means and power of advertisement.

When Microsoft sells its Student version of Windows XP and of Office, it uses the same strategy as drug dealers. The price is extremely low, as well as the version's capability, but the marketing buzzwords fly high: The name of the campaign is "Unlimited Potential". By the time the students realize their very limited potentials, they already became Windows-dependant. They will not begin studying Linux, a free and highly capable operating system. When the addicted ex-students graduate, they will buy and have their employers buy the pricier "hard stuff", the full-blown operating system and Office suite, naturally from Microsoft.

No Crime: Using Cookies and Scripts

Cookies and Javascript are important tools for keeping track of online sessions. Let's say, you fill out an order that spans through two pages. Once you entered your name in the first page, it's very convenient that the second page automatically fills out the field of credit card holder's name with your name, as a default. A Web site is capable to do so, without taking the information, sending it from your computer to the server, storing it into a database that keeps track of your session, and sending back your name along with the second page. Another application of client-side scripting is to formally check the data entry. You can enter your phone number with or without dashes, your name capitalized or not. The Javascript will take care of the right format. You may enter a six-digit phone number, a Javascript can show your mistake immediately, doing the format check on your computer, without sending the incorrect phone number to a roundtrip. Using cookies and Javascript helps the programmer keep the info and the process in your computer, rather than exposing it. Of course, there may be a point when the info must travel, but it has nothing to do with the described use of cookies and scripts.

Still No Crime: Cookies, Scripts and Spies

The content of this section has been removed due to its sensitive nature and its possible volatile use.

First Degree Crime: Reading Others' Cookies

The content of this section has been removed due to its sensitive nature and its possible volatile use.

Suggested Counter-Technology

The content of this section has been removed due to its sensitive nature and its possible volatile use.

Suggested Legislature

We have rights to free speech, which, unfortunately protect the liars and those who spread the bullshit, too... But what about thieves in the above sense? That a liar steals business from the competition? Shouldn't we bring our legal values closer to our moral and ethical values? If immoral lies would be considered a crime as theft is, false advertisements wouldn't gain so much ground in business. This step would require a courageous legislature, turning the questionable business behavior from free speech issue to "crime against property" issue, facing possible slow-down of the economy, etc. On the other hand, we could live in an open, honest, more humane society.

This page has been left intentionally blank.

PART 4. APPENDICES

The Web Developer's Ten Commandments

Thou shalt make thy programme's structure clear to thy fellow man

Thou shalt not mix content with presentation

Thou shalt not trust in default value settings

Thou shalt not annoy thy fellow man with blinking and jerking images or texts

Thou shalt validate thy HTML

Thou shalt not leave loose ends in thy code

Thou shalt respect thy user with proper and necessary messages

Thou shalt not use images excessively

Thou shalt not mix zero with null, lest grievous harm befall thy code

Thou shalt always keep it simple

Useful Links

That's us, Scriptwell.net, the home of good scripts. No hassle, no ads, no pop-ups, no dancing logos. Almost like a visit in a public library. Have your own armchair, and we supply the free reading material.

World Wide Web Consortium Markup Validation Service (validator.w3.org), where you can check the validity of your pages before or after you publish them.

Dynamic Drive DHTML code library (http://www.dynamicdrive.com/). Free, original DHTML scripts and components, all of which utilize the latest in DHTML and Javascript technology.

Jeffrey Zeldman's Web site (http://www.zeldman.com/). Free downloads of chapters from a must-have book. The writer is a real Web guru, not one of the many self-proclaimed ones; a funny but worrying warrior of browser compliance to standards.

WebReference.com (http://www.webreference.com/)

References

Brian W. Kernighan, Rob Pike: The Practice of Programming. ADDISON-WESLEY 1999, ISBN 0-201-61586-X

Jeffrey Zeldman: Designing with Web Standards. New Riders 2003, ISBN: 0-7357-1201-8

Dan Shafer: HTML Utopia: Designing Without Tables Using CSS. SitePoint 2004, ISBN: 0-9579218-2-9

Andy B. King: Speed Up Your Site: Web Site Optimization New Riders Publishing, 2003, ISBN: 0735713243.

Acronyms and Abbreviations

AJAX – Asynchronous Javascript and XML

CSS – Cascading Style Sheet(s)

DHTML – Dynamic HTML

DRA – Development of Rapid Applications

FF – FireFox

HTML – Hypertext Meta Language

IDE – Integrated Development Environment

IE – Internet Explorer

IT – Information Technology

NS - Netscape

RAD – Rapid Application Development

WYSIWYG – What You See Is What You Get

XHTML – Extensible Hypertext Meta Language

XML - Extensible Meta Language