Some PostScript Utilities for HTML and XHTML Revalidation

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2009 pub 11/09 as GuruGram #102
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

One of the ruder surprises of the web is that **different browsers tend to display in different manners**. Some may allow unique custom features, while others choke on them. As the web has aged and newer and better standards have emerged, the rules have gotten more and more strict.

In particular, HTML 4.0 and XTML now demand that...

Most commands are now case sensitive.
Most data must be quote bracketed.
Most commands must be lower case.
"LOWSRC" is no longer permitted.
"alt=" on images is now mandatory.
Text ampersands must be in "&amp;" format
<blockquote> has largely supplanted <ul>.
Some commands (such as <br />) must now self-delimit.
Id's have largely replaced names.
Id's and names have to start with a letter.
Delimiting spaces are now often mandatory.
JavaScript interpretation is now stricter.

A very useful validator can be found **here**. I was rudely surprised to find my older **web pages** generating thousands and even tens of thousands of errors per page. After manually correcting a few pages, I decided that most of the revalidation and verification could easily be handled by some hand written **PostScript** utilities.

As we have seen countless times in the past, **PostScript** excels as a **General Purpose Computing Language** when its unique features can be properly exploited. In particular, **PostScript is especially adept at modifying most any uncompressed text based disk file written in virtually any other computer language**.

In this **GuruGram**, we will explore a few ways that **PostScript** can dramatically speed up and simplify reverification of older website content to newer html and xhtml standards. Much more on our **PostScript** utilities appears **here**.

Repairing Ampersands

An ampersand is used as an "escape" character in both HTML and XHTML. Ferinstance **&nbsp;** creates a nonbreaking space, while **&gt;** gives you a "greater than" closing carat text character not to be used as a command delimiter. When a lone ampersand was found in an earlier browser, it was guessed to be a printing character. But such guesses are **not** permitted in current HTML or XHTML.

All printing ampersands must now be shown in their &amp; format.

Repairing ampersands gets ugly in a hurry. Lone ampersands are quite common in URL's such as **eBay Listings** or **Acme Mapper** locations, among many others. But **only those ampersands that are not followed by a semicolon within a few characters should get corrected**.

To make matters worse, **there is an insidious bug in DreamWeaver that may change all of your ampersands back the way they were hours after you fixed them!** If you must use **DreamWeaver** to change ampersands, **ALWAYS close your file immediately afterward** and **NEVER click on the refresh button**.

Instead, a simple and versatile utility can be created in **PostScript** that opens any file, inspects each ampersand to make sure there in no semicolon following in the next few characters, and then alters only those that need changed. One example program is **FIXAMPS1.PSL**. It simply reads one character at a time of an HTML or XHTML file and tests to see if a correction is needed.

The high level code looks something like this…

```
/correctampersands {
    /readfilename fileheader infilename mergestr store
    /readfile readfilename (r) file store
    /writefilename fileheader outfilename mergestr store
    /writefile writefilename (w+) file store

    0 1 10000000 {
       readfile (x) readstring not {exit} if
       /curchar exch store
       writefile curchar writestring
       testforampersand} repeat

    readfile closefile
    writefile closefile} store
```

… while the substitution utility is…

```
/testforampersand {curchar (&) eq {

    readfile bytesavailable 8 gt {

      /curposn readfile fileposition store
      readfile (x) readstring pop (;) eq
      readfile (x) readstring pop (;) eq or
      readfile (x) readstring pop (;) eq or
      readfile (x) readstring pop (;) eq or
      readfile (x) readstring pop (;) eq or
      readfile (x) readstring pop (;) eq or
      readfile (x) readstring pop (;) eq or
      not {writefile (amp;) writestring } if

      readfile curposn setfileposition} if
                        } if } store
```

If the present character is an ampersand and if none of the next few characters are a semicolon, then a new **amp;** is written between the existing ampersand and the continuing text.

One known bug is that ampersands inside a Visual Basic script internal to an **.asp** file will need separate attention as they must not be changed.

Our **Banner Rotator** uses a VB script line of **pattern = a(0) & a(1) … & a(8)** which must remain intact. Such exceptions are very rare and easily dealt with.

"Search and Destroy" Phrase Substitution

A different approach can be used to try and repair the majority of earlier HTML errors. In which big chunks of the code are bulk scanned for problem phrases. Should a problem phrase be found, it can be replaced with corrected code. We might call any earlier phrase **Wuz** and the replacement phrase **Wilby**.

Now, some Wuz phrases will be generic and common to most all early HTML. Such as converting a **<br>** to a **<br />**. Others will be specific to your web page style. Such as changing a **color=$FFCC99** to a **color="FFCC99"** . Such substitutions will be useful **only** if this particular color is of importance in your page layouts.

It is important to decide how much generic and how much specific code you wish to correct. In general, taking out **most** of the errors with an automated routine will greatly simplify and speed up your **revalidation**. But, try for a perfect repair and you will end up spending much more time coding and testing than you would fixing the problems in the first place.

AUTOVAL1.PSL is an example of a **PostScript** phrase substituter. It can be used on most any uncompressed text file in most any language, but clearly excels at html and xhtml repair.

At present, the repairs take place on sequential 40K strings. These are all long enough for surprisingly fast operation but still stay within a 64K limit should your wilby strings add to your file size.

A **srpairs** scripting data file is first created listing your possible wuz and wilby substitutions. This is an array of form **[[(wuz1)(wilby1)] ... [(wuzn)(wilbyn)]]** The wuz and wilbys can be a mix of your generic and specific code. Such as this partial example...

```
/srpairs [
  [ (<br>)(<br />) ]
  [ (<BR>)(<br />) ]
  [ (<ul>)(<blockquote>)]
  [ (</ul>)(</blockquote>)]
  [ (<UL>)(<blockquote>) ]
  [ (</UL>)(</blockquote>) ]
  [ (<name=)(<name=")]
  [ (<href=h)(<href="h)]
  [ (<a href=h)(<a href="h)]
  [ (<A href=h)(<a href="h)]
  [ ( href=m)( href="m)]
  [ (HREF=) (href=)]
  [ ("")(")]

  [(color=#FF9999) (color="#FF9999") ]
  [(width=87 ) (width="87" ) ]
  [(height=25 ) (height="25" ) ]
       ] def
```

The entries before the break are generic examples, while those after the break are specific to a particular website style. Wuz and Wilby entries can be much longer, especially when changing from older to newer html versions. A **srpairs** file will typically be dozens to hundreds of entries long. You could directly study our **AUTOVAL1.PSL** for current examples. These are easily customized for your particular needs.

It is important to chose your Wuz phrases with care. It is usually best to have an ending space or closing carat. This prevents, say, a **(width=65)** conversion from doing both a **(width="65")** and an unwanted **(width="65"0)**. Or picking up only part of a URL because of trailing slashes.

Your high level code might look something like this…

```
/autovalidate {
  /readfilename fileheader infilename mergestr store
  /readfile readfilename (r) file store
  /writefilename fileheader outfilename mergestr store
  /writefile writefilename (w+) file store
  /curstring 40000 string store

  {readfile curstring readstring      % early strings
    not{exit}if searchandreplace
    writefile exch writestring} loop

   searchandreplace                     % final string
   writefile exch writestring

   readfile closefile
   writefile closefile} def
```

The high level code breaks up the input file into 40K chunks and then will do a **searchandreplace** on each chunk. The Wilby for Wuz substitution code can be…

```
/searchandreplace { /curstring exch store

    srpairs { dup 1 get /wilby exch store
            0 get /wuz exch store
            {curstring wuz search not {exit} if
            exch pop wilby mergestr
            exch mergestr
            /curstring exch store } loop

               } forall

            curstring} store
```

The **PostScript search** operator normally returns three strings: Everything up to Wuz which simply gets written to your new file. Wuz, which gets replaced by Wilby in your new file; and all post characters which become the next **curstring** for the continuing search. The **forall** operator goes through the Wuz and Wilby pairs in order.

Speed is typically faster than one second, but varies with the complexity of your **srpairs** array and the length of the html file being corrected.

Yes, there is a remote possibility that a repair might be needed spanning two repaired chunks. But the odds of this happening are very low and your validation routines will quickly spot them. Again, your usual goal should be to have the automated code dramatically reduce and ease your total revalidation time. But perhaps not eliminate it entirely.

Fancier code can easily be written that makes a second pass with an offset to deal with chunk spanning. You could also **isolate html carat strings** to make sure what you are changing is in fact html and not coincidental content text.

Working with carat strings can also greatly eases any changes you want to make to the **end** of an html command. Such as changing the ending of **<img ... >** to an **<img ... alt=" " />**. Or adding an **id=** after each **name=**.

Or putting an ending quote on some **href="..."** without worrying about every possible filename extension or slash.

Custom code services are **available**.

Again, I was overjoyed that the automated **PostScript** code reduced the per page errors from thousands of difficult and obscure ones to a dozen or so of simpler ones. And did so astonishingly fast with remarkably simple and easy code.

Scripting

The **AUTOVAL1.PSL** utility is easily modified so it can work with an entire list of web pages to be **revalidated**. You first create a scripted list of files to be corrected...

```
/filestofix [

(bkpz.asp)
(bkradast.asp)
(bkrecomm.asp)
(bkreview.asp)
(bkrfid.asp)
(bkrobot.asp)
(bksanta.asp)
(bkseismo.asp)
(bkseti.asp)
(bkspect.asp)
(bksvg.asp)
    ...
                ] store
```

Your new **SCRIPVAL.PSL** uses a high level loop similar to...

```
/scriptvalidate {
    filestofix {/curfile exch store
    /readfilename infileheader curfile mergestr store
    /readfile readfilename (r) file store
    /writefilename outfileheader curfile mergestr store
    /writefile writefilename (w+) file store

    /curstring 40000 string store
   {readfile curstring readstring        % earlier strings
     not{exit}if searchandreplace
     writefile exch writestring
    } loop

    searchandreplace                      % final string
    writefile exch writestring
    readfile closefile
    writefile closefile
           } forall} def
```

A second **forall** loop get used to step through the file page names in order. Amazingly, many hours of hand labor is replaced with a very few seconds of instant results.

One detail: Apparently there is no direct way in Windows XP to capture directory listings. As would be handy to make your list of files to be corrected. Instead, a free utility called **CopyFilenames** can be installed to provide directory-to-clipboard transfers.

Some JavaScript Considerations

Older client side JavaScript programs also can also present problems with newer versions of HTML or XHTML. For instance, **this example** originally would display but not calculate at all.

Our first rule in dealing with JavaScript revalidation problems...

Make sure the HTML or XTML code revalidates before attacking JavaScript specific issues.

Note that most modern browsers may have an optional **JavaScript Debugging Console** that can be activated with a mouse click or two. This can be a highly useful tool to locate and repair compliance issues.

One major compatibility problem is...

While HTML ignores scripts,
XHTML parses them as data!

There are two recommended workarounds. The "best" appears to be to **import your JavaScript code from another file**. Like so...

```
<script type="text/Javascript"
    scr="externalfilename.js">
</script>
```

A second workaround is to **tell XHTML to ignore your JavaScript code**. This approach may be better for very short routines of only a few lines...

```
<script type="text/javascript">
/* <![CDATA[ */
     // JavaScript content goes here
/* ]]> */
</script>
```

Some sneakiness is involved here. The **CDATA** portion tells XHTML to ignore the enclosed data. While the **/*** and ***/** bracketing tells JavaScript that this is to be ignored by treating as a JS comment.

Another issue is that **some early browsers introduced their own features that were not part of the JavaScript standard. In particular, the problem with our** **Magic Sinewave** calculator was that IE allowed a "short form" of i.d. that was **not** part of the JavaScript standard, but got picked up by other browsers. Some newer browsers (especially **Firefox 3.5** or later) may place a more strict interpretation of what is or is not legal in JavaScript.

Specifically, these may no longer be legal or allowed...

```
        cfh05.bgColor="#CCFFFF" ;
or
        eval ( unrejHarm1 + ".bgColor='#66FFCC'") ;
```

Instead, replace with these…

```
   document.getElementById('cfh05').style.backgroundColor =
   "#CCFFFF";
or
   eval ( "document.getElementById('" + unrejHarm1 +
   "').style.backgroundColor = '#66FFCC'" ) ;
```

Our first example defines the color of a fixed location called **cfh05** , while the second one allows a calculated or changing color for a variable location. These also created messages trapped by the error console, making them relatively easy to find and fix.

Proofing

Having a "green" validation does **not** mean your web pages are error free! All green means is that your work is not so bad that a browser will choke on it. You could still have broken links or changes in format that you did not intend.

So, **it is extremely important to carefully proof each and every page after your validation!**

Such proofing is best done by a third party. While a url checker is found **here**, a manual recheck of each and every link is strongly recommended.

It is also important to **use your ISP's log files** to keep track of 404 trends and similar errors. It is usually not possible to get much below a one percent error rate. But anything above should be suspect. As should more than a very few hits on one particular error. More on log file use and manipulation **here**.

For More Help

Many more application programs and utilities appear on our **PostScript Library** page. A guide to our **PostScript Gonzo Utilities** and additional links appear **here**.

Additional consulting services are available per our **Infopack** services and on a contract or an hourly basis. Additional **GuruGrams** are found **here**. Seminars also available.

Further **GuruGrams** await your ongoing support as a **Synergetics Partner**. For details, you can email **don@tinaja.com**. Or call **(928) 428-4073**.