A PostScript-as-Language Search & Replace Utility

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2002 as GuruGram #12
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Ordinary word processors can create problems when editing eight bit data due to control characters, reserved codes, and differences in line endings. Attempting to edit an **Acrobat PDF** file with an ordinary word processor can cause problems with embedded fonts and encoded Flate Compressed strings.

Instead, a **PostScript-as-Language** utility can easily be created that lets you globally search and replace a file of any content without any rude surprises. Such a program is also a good review of PostScript disk file manipulations as well.

I've uploaded a PostScript Search & Replace **(S&R)** utility to our **PostScript** library as file **SEAREPL.PSL**. Normally, you would bring this file up into a wp or editor, modify it for your own uses, and then send it to **Acrobat** distiller. Let's look at some key code in this utility…

I strongly recommend using my **gonzo utilities** for most PostScript-as-language development. But for this file, the only utility we only immediately need is the ability to merge two stack-top PostScript strings…

```
/mergestr {2 copy length exch length add string dup dup
4 3 roll 4 index length exch putinterval 3 1 roll exch
0 exch putinterval} def
```

We start the utility by entering the access data…

```
/diskfileheader (C:\\WINDOWS\\Desktop\\inherit) store
/diskfilesourcename (g9demoyx.pdf) store
/diskfiletargetname (g9demozz.pdf) store
/wuz   ( 4 0 R) store                 % string to search
/wilby ( 3 0 R) store                 % string to replace
```

Several minor gotchas here: **All PostScript strings MUST include double slashes when a single slash is wanted!**

Also, when appropriate be sure to provide a leading space in previous content **wuz**. And, when using this utility for PDF post processing, it is a very good idea to have the same character counts in **wuz** and **wilby**. This can eliminate any need for cross reference rebuilding.

We then create our full filenames and convert them to readable and writable PostScript file objects…

```
/sourcefilename diskfileheader diskfilesourcename
        mergestr store
/targetfilename diskfileheader diskfiletargetname
        mergestr store
/readfile sourcefilename  (r) file store     % set input read
/writefile targetfilename (w+) file store   % set output write
```

Note that the (w+) tells us to **append** file info, rather than overwriting it.

There is a 65K limit on PostScript string length, which can easily be exceeded by a PDF or other general file. Thus, several passes might be needed for full processing. Our main processing loop looks like this…

```
/workstring 60000 string def          % file read workstring
/oops 4000 def                        % infinite loop lockout
oops {  readfile workstring readstring
       {searchandreplace}
       {searchandreplace exit} ifelse
        } repeat
```

We simply read as much of the file as fits our workstring and then process that piece. An unlikely infinite loop is avoided with **oops** should something go very wrong with the disk reading process.

The search and replace code could look like this…

```
/searchandreplace {oops { wuz search   % seek match
{ writefile exch writestring           % save previous
pop wilby writefile exch writestring}  % do replacement
{ writefile exch writestring exit}     % till nothing left
    ifelse} repeat  } def
```

Note that several trips through are needed for multiple replacements. The remaining unprocessed string end gets passed back in to do this.

Some extra code is added to the actual utility to report each replacement done or if no replacements took place.

Multiple searches and replacements can be done at once by extending the code into a **forall** loop or a **forall** dictionary.

Two big gotchas: First make sure wuz and wilby have **exactly** the same number of characters if you wish to avoid a risky PDF cross reference rebuild. A padding space can usually be added if needed.

Also, make sure wuz and wilby don't take out anything unexpected. Ferinstance, a wuz of **4 0 R** will also take out **14 0 R** or **24 0 R**. A leading space cures this particular problem. Also, if your **text** contains a **4 0 R**, add some kerning or something else to prevent an inadvertent substitution.

Altering PDF Transparency

We saw back in **GuruGram #8** several approaches to creating transparent PostScript artwork by using raw PostScript code sent to Distiller. As of this writing, a PDF post patch seems to still be needed to actually link the correct gstates.

This S&R utility greatly simplifies the process...

1. **In a word processor, search for all PDF Xobjects.**
2. **Write down the wuz and wilby form object numbers.**
3. **Enter wuz, wilby and filenames into the S&R.**
4. **Run the S&R program for automatic conversion.**

On any repeated editing of the PDF file, simply rerun S&R for an automatic opaque to transparency conversion.

Consulting assistance on any and all of these topics can be found at **http://www.tinaja.com/info01.asp**.

Additional **GuruGrams** await your support as a **Synergetics Partner**.