Adding Historical Stats to Your Logfile Analyzer

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Back in **GuruGram #28** , we looked at some custom **Logfile Reporter** routines that gave you amazingly complete stats on your website activity. Along with some added features that analyzed your **eBay** offer viewings and even detected stolen **ebay Images**.

I've recently added some tentative extensions to these **PostScript** logfile analysis routines. Which now can give you a **trailing 30 day history** of certain values.

At present, the **pageviews** and **banner ad** deliveries versus time get reported so trends can be assessed. In addition, cumulative **file popularity**, the **accumulated errors**, and most **popular referrals** are also sorted and listed.

The usual problem with logfile histories is that you can easily rack up many hundreds of Megabytes of data storage, coupled with very slow and cumbersome processing times. I elected to use a much simpler data structure that **saves only the essentials**. Often ending up as a 250K data file for an entire month. One that is easily processed in a few seconds.

There are three code modules involved in the code history process. **trail30.psl** is the data base itself, which starts out at zero length, builds till it has 30 days of info, and then continues to hold the 30 most recent days. Routine **dayproc1.psl** services this data base, both keeping it current and then using the latest info to generate appropriate reports as an **Acrobat Distiller** log file.

Finally, **logrptm1.psl** is the actual log file reader and report creator. It is similar to our older **logrpt01.psl** except for two minor differences: The previously host loaded logfile name has been split into a short name and a path to simplify date stripping; and **dayproc1.psl** is conditionally run at the end to create the trailing history report.

Operation is pretty much like **before**. You make your ISP log files locally available. Then you run **logrptm1.psl** once a day, changing its log filename and sending it to **Acrobat Distiller**. Your resultant Distiller log file should now report both today's info and a month's trailing stats.

Data Base Details

File **trail30.psl** defines a **PostScript** array proc named **t30data**. To enter the history data into **dayproc1.psl**, you simply run **trail30.psl**, being careful to use the **full path and filename**. And remembering that a **double reverse slash** is needed anytime you want a **single** reverse slash inside any **PostScript** filename.

Done this way, your history data can be of any length. And go well beyond the usual **PostScript** string limit of 65K characters.

Your data base size can be limited by restricting what you put into it and by permitting only a maximum number of file entries per day. I have found a **60 file daily limit** on popular files, on errors, and on referrals is more than useful for me. And still gets us by with a 250K or so total file size.

You always start off with a dummy **trail30.psl** file of…

```
/t30data  [  ] store
```

You can enter this manually, copy it from a different location, or provide auto reset code any time you need to start your trailing 30 day data over.

After a few days, your **t30data** in **trail30.psl** should look something like this…

```
/t30data [

 [ [(040418)]
   [(1623)]
   [(3906)]
   [[(hackar2.pdf) 230][(hackar1.pdf) 131]...]
   [[( /medint3.asp) 7][( /_vui_ban/search.htm0.idq) 6]...]
   [[(http://www.brou.com) 9][(http://www.epan.net) 5]...]
 ]

 [ [()][()][()] [[()a#][()b#...][[()c#][()d#...][[()e#][()f#...] ]

 [ (third oldest data array) ]
      . . . . . . . . . . .

 [ (most recent data array of last 30 days) ]

         ] store
```

There is **one subarray per day** and each subarray in turn holds **six** individual data arrays. The first array is a string integer holding the **date** in **YYMMDD** format. The second array is a string integer holding that day's **page views**. The third array is a string integer holding that day's **banner ad deliveries**.

As shown above, the fourth array consists of a collection of **popular files** data arrays. Each entry consists of a **string filename** and an **integer popularity count**. Similarly, the fifth array holds the most severe **error messages**, while the sixth one keeps our more **popular referrals** for us. All of the latter in a **[{name} count]** subarray format.

It is important to do each day's processing once and only once. And to process your days in order. The **YYMMDD** field can optionally be used to make sure the days are in order and only get listed once.

To convert from **YYMMDD** to a more conventional date string, you can use

```
t30data curday1 get 0 get 0 get /cryptedday exch store

cryptedday 2 2 getinterval cvr 1 sub cvi [(Jan )(Feb )(Mar )
(Apr )(May )(Jun )(Jul ) (Aug )(Sep )(Oct )(Nov )(Dec )]
exch get (    ) exch mergestr cryptedday 4 2 getinterval
mergestr ( - ) mergestr

t30data curday1 get 1 get 0 get 10 string cvs mergestr
(\n) mergestr print
```

This example would convert an **040415** string into an **Apr 15 -** string. As before, you run **logrptm1.psl** once daily. Besides generating the earlier report, it also now runs **dayproc1.psl**. Which maintains the data base through my usual stunt of **using PostScript to write PostScript**.

The popularity counts are maintained similar to before. Except that counts go up by the day's popularity rather than by a single count. Each new filename or error or referral gets compared against the existing ones in their respective arrays. If a match is found, the count is suitably bumped. If there is no match, a new entry gets added.

Using the usual trick of autoexpanding an array…

```
/myarray mark myarray aload pop [newstuff] ] store
```

I've purposely left the reports in straight text format. Naturally, with **PostScript** and my **Gonzo Utilities**, you can create arbitrarily fancy charts and graphs in infinite variety.

Bells and whistles galore.

You can click here for a **sample output report**. Sourcecode for this **GuruGram** appear as **histolog.psl**.

For More Help

As noted, this trailing month reporter is an expansion upon our custom **Logfile Reporter** routines.

Additional **PostScript**, **Acrobat**, **eBay**, and **Webmastering** assistance is available per the previously shown web links. Custom modification and design services are available at our standard consulting rates. Per our **InfoPack Services**. Or you can directly **email** me.

Further **GuruGrams** columns await your ongoing support as a **Synergetics Partner**.