Paper CC-17

Your Friendly Neighborhood Web Crawler: A Guide to Crawling the Web with SAS®

Jake Bartlett, Alicia Bieringer, and James Cox PhD, SAS Institute Inc., Cary, NC

ABSTRACT:

The World Wide Web has a plethora of information; from stock quotes to movie reviews, market prices to trending topics, almost anything can be found at the click of a button. Many SAS users are interested in analyzing data found on the Web, but how do you get this data into the SAS environment? Various methods are available, such as designing your own Web crawler in SAS DATA step code or utilizing the %TMFILTER macro in SAS® Text Miner. In this paper, we will review the general architecture of a Web crawler. We will discuss the methods of getting Web information into SAS, as well as examine experimental code from an experimental internal project called SAS Search Pipeline. We will also offer advice on how to easily customize a Web crawler to suit individual needs and how to import specific data into SAS® Enterprise Miner™.

INTRODUCTION:

The Internet has emerged as a useful source of information. Often there is data on the Web that we want to use from within SAS, so we need to find a way to get this data. The best solution is to use a Web crawler. SAS offers several methods of crawling and extracting information from the Web. You can use basic SAS DATA step code, or SAS Text Miner's %TMFILTER macro. While not currently available, the SAS Search Pipeline will be a powerful Web crawling product and will provide more tools for Web crawling. Each method has its advantages and disadvantages, so depending on what you wish to achieve in your crawl, it is best to review them all. First, it's important to understand how a Web crawler works.

You should be familiar with DATA step code, macros, and the SAS procedure PROC SQL before continuing.

WEB CRAWLER OVERVIEW:

A Web crawler is a program that, given one or more start addresses known as "seed URLs", downloads the Web pages associated with these URLs, extracts any hyperlinks contained in the Web pages, and recursively continues to download the Web pages identified by these hyperlinks. Conceptually, Web crawlers are very simple. A Web crawler has four responsibilities:

1. It selects a URL from a set of candidates.
2. It downloads the associated Web pages.
3. It extracts the URLs (hyperlinks) contained within a Web page.
4. It adds those URLs that have not been previously encountered to the candidate set.

METHOD 1: WEB CRAWLER IN SAS DATA STEP CODE

Begin by creating a list of the URLs of the Web sites where the Web crawler will start.

```
data work.links_to_crawl;
   length  url $256 ;
   input url $;
   datalines;
http://www.yahoo.com
http://www.sas.com
http://www.google.com
;
run;
```

To ensure that we don't crawl the same URL more than once, create a data set to hold the links that have been crawled. The data set will be empty at the start, but a Web site's URL will be added to the data set when the Web Crawler completes the crawl of that site.

```
data work.links_crawled;
   length url $256;
run;
```

Now we start crawling! The code takes the first URL out of our **work.links_to_crawl** data set. On the first observation "_n_ eq 1",the URL is put into a macro variable named **next_url** All remaining URLs are put back into our seed URL data set so they are available in the next iteration.

```
/* pop the next url off */
%let next_url = ;

data work.links_to_crawl;
   set work.links_to_crawl;
   if _n_ eq 1 then call symput("next_url", url);
   else output;
run;
```

Now download the URL from the Internet. Create a filename reference called **_nexturl**. We let SAS know it is a URL and it can be found at **&next_url,** which is our macro variable that contains the URL we pulled from our **work.links_to_crawl** data set.

```
   /* crawl the url */
   filename _nexturl url "&next_url";
```

After the filename reference of the URL is established, identify a place to put the file we download. Create another filename reference called **htmlfile** and put in there the information gathered from *url_file.html*.

```
   /* put the file we crawled here */
   filename htmlfile "url_file.html";
```

Next we loop through the data, write it to the **htmlfile** filename reference, and search for more URLs to add to our **work.links_to_crawl** data set.

```
   /* find more urls */
   data work._urls(keep=url);
      length url $256 ;
      file htmlfile;
      infile _nexturl length=len;
      input text $varying2000. len;

      put text;

      start = 1;
      stop = length(text);
```

Use regular expressions to help search for URLs on a Web site. Regular expressions are a method of matching strings of text, such as characters, words, or patterns of characters. SAS already provides many powerful string functions. However, regular expressions often provide a more concise way of manipulating and matching text.

```
if _n_ = 1 then do;
   retain patternID;
   pattern = '/href="([^"]+)"/i';
   patternID = prxparse(pattern);
end;
```

On the first observation, create a patternID that will remain throughout the DATA step run. The pattern to look for is: "/href="([^"]+)"/i"'. This means that we are looking for the string 'href="', then look for any string of characters that is at least one character long and does not contain a quotation mark ("), and ends in a quotation mark ("). The 'i' at the end means to use a case insensitive method to match our regular expression.

As a result, the Web crawler will find these types of strings:

- `href="sgf/2010/papers.html"`
- `href="www.yahoo.com"`
- `HREF="www.google.com"`
- `hReF="http://www.sas.com"`

Now match the regular expression to the text on a Web site. PRXNEXT takes five arguments: the regular expression we want to find, the start position to begin looking for the regular expression, the end position to stop looking for the regular expression, the position of the string if it is found, and the length of the string if it is found. *Position* will be 0 if no string is found. PRXNEXT also changes the start parameter so that searching begins again after the last match is found.

```
call prxnext(patternID, start, stop, text, position, length);
```

The code loops over text on the Web site to find all the links.

```
do while (position ^= 0);
   url = substr(text, position+6, length-7);
   output;
   call prxnext(patternID, start, stop, text, position, length);
end;
run;
```

If the code finds a URL, it will retrieve only the part of the URL that starts after the first quotation mark. For example, if the code finds HREF="http://www.new-site.com", then it should keep http://www.new-site.com. Use **substr** to remove the first six characters and the last character before outputting the remainder of the URL to the **work._urls** data set.

We now insert the URL that the code just crawled into a data set named **work.links_crawled** in order to keep track of where we have been and to make sure we do not navigate there again.

```
/* add the current link to the list of urls we have already crawled */
data work._old_link;
   url = "&next_url";
run;

proc append base=work.links_crawled data=work._old_link force;
run;
```

Next step is to process the list of found URLs in the data set **work._urls** to make sure that:

1. we have not already crawled them, in other words URL is not in **work.links_crawled**).
2. we do not have URLs queued up to crawl, (in other words URL is not in **work.links_to_crawl**).

```
/*
 * only add urls that we have not already crawled
 * or that are not queued up to be crawled
 *
 */

   proc sql noprint;
      create table work._append as
         select url
         from work._urls
         where url not in (select url from work.links_crawled)
            and url not in (select url from work.links_to_crawl);
   quit;
```

Then we add URLs yet to be crawled and not already queued to the **work.links_to_crawl** data set.

```
   /* add new links */
   proc append base=work.links_to_crawl data=work._append force;
   run;
```

At this point the code loops back to the beginning and grabs the next URL out of the **work.links_to_crawl** data set.

We now have a basic Web crawler! The source code to the full Web crawler can be found at http://www.sascommunity.org/wiki/Simple_Web_Crawler_with_SAS_Macro_and_SAS_Data_Step.

SOME PITFALLS OF METHOD 1:

ONLY FINDS GLOBAL URLS

Our simple Web crawler implementation in DATA step has many shortcomings. First of all, it is looking for URLs that are only found within <a href=" … "> tags. URLs can appear in many places throughout a Web page. A more robust method of extracting URLs would include more and more complicated Perl regular expressions. Secondly, our simple Web crawler can crawl URLs that only start with http:// . If we find a tag like <a href="../index.html"> we would have to include post processing code that converts ../index.html to http://www.new-site.com/index.html in order to be able to use our FILENAME statement to navigate to the Web page.

NOT SAFE

There are no built-in safeties. The Web crawler will continue until there are no more links in the **work.links_to_crawl** data set. Ideally, we would add some post processing checks to determine if we left our target domain. For example, if we added "http://www.yahoo.com" as a seed URL, we would want to crawl only those Web pages that contain "http://www.yahoo.com" , (in other words http://www.yahoo.com/autos and **not** http://www.google.com).

RUDE

There is a concept of politeness in Internet browsing. Our little Web crawler is simple and fast. It can find and extract hundreds of URLs in a second. Some Web sites like Yahoo! monitor rapid Web page download requests from a single source. If they detect too much traffic from a single source, they will throttle their response time in serving the Web page you requested. This can greatly slow down the crawler. Also, the crawler might not download the entire page due to these throttling issues.

METHOD 2: WEB CRAWLING WITH SAS TEXT MINER

All of the shortcomings of our Web crawler can be fixed. We can try to modify our simple SAS DATA step Web crawler, or opt to use a better one. To choose from a rich set of data mining tools that SAS Enterprise Miner provides, the SAS Text Miner plug-in offers a better Web crawler with its %TMFILTER macro. If you have licensed SAS Text Miner, then you have access to the %TMFILTER macro. This macro calls some optimized code for downloading and importing files.

%TMFILTER can perform two tasks:

1. It can crawl the Internet.
2. It can crawl local files and directories.

If given a URL, %TMFILTER can navigate to the site, download the file, save it as HTML, examine it for more URLs, and continue crawling.

If, instead of a URL, a local file directory is specified, %TMFILTER will recursively navigate the directory tree, converting the files it finds into HTML. These files could be a collection of Portable Document Format (PDF), Hypertext Markup Language (HTML), or Microsoft Office documents in a hard drive folder.

%TMFILTER works by first converting all files to HTML and storing them in a destination directory on the hard drive. Then it creates a SAS data set containing information about where each file is stored, its format, and so on. These HTML files can then be imported into SAS.

To convert a directory full of files into a SAS data set, you can use %TMFILTER like this:

```
%tmfilter(dataset=data-set-name, dir=path-to-original-documents,
          destdir=path-to-html-files);
```

To download a URL and crawl the Web, you can use %TMFILTER like this:

```
%tmfilter(dataset=data-set-name, dir=path-to-original-documents,
          destdir=path-to-html-files, url=URL, depth=integer);
```

The URL parameter tells %TMFILTER where it should start crawling. The depth parameter provides a limit by telling %TMFILTER how many levels down it should crawl.

SOME ADVANTAGES OF METHOD 2:

%TMFILTER is a superior solution to the Web crawler in SAS DATA step code. Because the core crawling module of %TMFILTER is written in the C++ programming language, it is a much faster and efficient crawler. Additionally, the macro can extract many more URLs. The SAS DATA step Web crawler can find only the types of URLs we tell it to. Much of that will depend on your ability to compose and debug regular expressions in the Perl programming language. Because Perl regular expressions are so compact and concise, it can be very difficult to find and debug errors when composing them. %TMFILTER has taken care of all that for us automatically. Finally, it is less error prone than our SAS DATA step Web crawler. %TMFILTER has been used for a number of years in production to crawl many hundreds of gigabytes of Internet data. It is a professional grade product.

More information about how to best use %TMFILTER can be found in the SAS Text Miner documentation.

METHOD 3: WEB CRAWLING WITH SAS SEARCH PIPELINE

If the other two methods do not meet your needs, then you will need the power of the SAS Search Pipeline. This is currently an internal project at SAS which we hope to make available to customers in the near future. It will have the ability to crawl the entire Web, extract and organize information from sites, and index the data to make it easy to search through later. It starts with a simple seed URL or multiple URLs if you prefer. From there, it works like other Web crawlers. It downloads the site, searches for links, downloads those linked sites, and so on. What makes it special? The answer is its processing technique. First of all, the SAS Search Pipeline will allow you to customize your own parser, and thus index files to meet your needs. For example, you might wish to make each downloaded page a separate article, or you might be interested in downloading pages containing multiple RSS feeds such that each feed is a separate article. The customizable parser provides an easy, convenient way to handle this distinction.

Not only does the SAS Search Pipeline offer pre-processing features, but it also will provide postprocessor options. Suppose you wish to remove the HTML tags from a site or to prevent the crawler from downloading the same site twice. The SAS Search Pipeline will take care of this for you. It will have many built-in postprocessor options, including extracting specific data, categorizing articles, and modifying field names for indexing. The index builder in SAS Search Pipeline stores all successfully processed articles in an index which can be modified by removal or appendance of more articles. Also provided is a query tool to search for one or more keywords in which you are interested. You are provided with a list of articles that feature the keyword or keywords and you can use this list for further research.

One of the best features of the SAS Search Pipeline is that it can produce results to be integrated with other SAS tools. The SAS Search Pipeline includes a postprocessor to export articles directly to a SAS data set or to a Comma Separated Values (CSV) file that can conveniently be read into SAS using PROC IMPORT. In this way you can use SAS to analyze the entire Web!

CONCLUSION:

The Internet should no longer be a barrier between getting the information we need and performing the analysis we need. SAS offers many methods of extracting data from the Web, and each method has its own set of advantages and disadvantages. These methods should be used, expanded upon, and combined to give you the results you need when downloading content from the Web.

RECOMMENDED READING:

- Cody, Ron, Robert Wood Johnson Medical School, Piscataway, NJ 2004. "An Introduction to Perl Regular Expressions in SAS 9." *Proceedings of the SAS Global Forum Conference*. Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/sugi31/110-31.pdf.
- Konchady, Manu. 2006., *Text Mining Application Programming (Programming Series).* Rockland, MA: Charles River Media, Inc.
- Najork, Marc. 2009. *Web Crawler Architecture*. Springer Verlag.Available http://research.microsoft.com/apps/pubs/default.aspx?id=102936
- SAS Institute Inc. 2004. *Getting Started with SAS® 9.1 Text Miner.* Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/em_tmgs_7693.pdf.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Jake Bartlett
SAS Campus Drive
SAS Institute Inc.
E-mail: Jake.Bartlett@sas.com

Alicia Bieringer
SAS Campus Drive
SAS Institute Inc.
E-mail: Alicia.Bieringer@sas.com

James Cox
SAS Campus Drive
SAS Institute Inc.
E-mail: James.Cox@sas.com