Paper 062-2011

Turn Web Source Code into SAS® Data in Two Easy Steps

Ryan Massa-McKinley, AdvanceMed, Nashville, TN

ABSTRACT:

In the ever-expanding World Wide Web, there are millions of new websites every month with potentially valuable information. Many SAS users would like to convert source code from a url or list of url's into SAS data. SAS offers tools for web crawling and text mining to turn the mounds of textual data into information. This paper will provide the readers with an easy way to extract the text from websites into SAS data sets using Base SAS in just two data steps.

INTRODUCTION:

The FILENAME Statement (URL Access Method) in Base SAS, enables users to access the source code from a web site and read it into a data set. The syntax for this statement is:

FILENAME *fileref* URL *'external-file'<url-options>;*

This paper will provide a step-by-step explanation of how you can use this method to read the source code from a url and more importantly how to extract the textual data within the source code into a SAS data set for analysis.

DATA STEP 1:

Once we initialize the FILENAME statement, we can use the fileref name in an INFILE Statement to read in the source code as raw text data:

```
FILENAME SOURCE URL "%STR(http://www.usatoday.com)" DEBUG;
DATA SOURCE1;
      FORMAT WEBPAGE $1000.;
INFILE SOURCE LRECL=32767 DELIMITER=">";
INPUT WEBPAGE $ @@;
RUN;
```

With this bit of code, the data set SOURCE1 will contain all the source code from the *USA Today* url in the variable WEBPAGE.

FORMAT Statement: The variable WEBPAGE was given a format of $1000. so it will be long enough to hold the majority of HTML code lines.

LRECL= Option: The LRECL= option on the INFILE statement was set to 32767 because that is the maximum record length for a line in SAS. The default is 256, so if the record length were not expanded, you would risk truncating each line and losing some of the source code you are attempting to input.

DELIMITER= Option: The DELIMITER= option is set to use the symbol > as the delimiter. In most webpage languages text strings are preceded by this symbol. The SAS default delimiter is a space, so without specifying a delimiter, you would lose the ability to maintain text phrases because SAS will interpret each word as a separate value.

@@ ***Option:*** The @@ option in the INPUT statement tells SAS to place each value in the next observation instead of placing it in the next variable. Since we do not know the maximum number of values that could appear in a single line of code, we do not want to specify multiple variables, so we keep all the raw data in a single variable. Without the @@ option, only the first value on each line of source code will appear in your

data set. Without the DELIMITER= option or the @@ option, only the first word from each line of source code would appear in your data set.

RAW SOURCE CODE EXAMPLES:

After this portion of code, here are some examples of the values we have in the variable WEBPAGE:

1. `Snapshot: What affects the most in buying a new car?</a`
2. `10/20/2010 - Updated 08:01 PM ET</span`
3. `Apple stock: Take a bite, if you can handle the risk</a`
4. `<meta http-equiv="X-UA-Compatible" content="IE=9"/`
5. `<script type="text/javascript"`
6. `<li class="navItem"`

From these examples, values 1-3 contain text we want to capture, but values 4-6 contain only source code with no textual data we want to keep. In most web languages, text strings are found in the following pattern: >*text string*<. Since we used the symbol > as a delimiter, we expect our text strings to be at the left-most position in our variable and ending before the symbol <.

DATA STEP 2:

The following data step will remove almost all unwanted source code, leaving us with the textual data we are trying to obtain from the website:

```
DATA SOURCE2;
SET SOURCE1;
       WHERE WEBPAGE LIKE "_%<%";
       TEXT=SUBSTRN(WEBPAGE,1,FIND(WEBPAGE,"<")-1);
       TEXT=TRANWRD(TEXT,"%NRSTR( )"," ");
       TEXT=TRANWRD(TEXT,"%NRSTR(&amp;)","&");
       IF ANYALPHA(TEXT) + ANYDIGIT(TEXT) LT 1 THEN DELETE;
       KEEP TEXT;
RUN;
```

LIKE Function: The LIKE function enables us to specify the text pattern to look for: the underscore (_) is like a wild card and represents any single character, one and only one character. Two underscores would represent any two characters. The % symbol represents any number of characters. Unlike the underscore, the % symbol can also represent zero characters.

- So in our WHERE statement: `WHERE WEBPAGE LIKE "_%<%";` the text pattern we are looking for is _%<% This means our data set will only include values that contain the < symbol and have at least one character (_) prior to the < symbol. It can be any length and there can be any number of other characters, located anywhere in the string. Since almost all text strings will appear between these symbols >*text string*< the remaining values will contain the textual data.
- This data step should return the same results without this particular statement. But on webpages with many lines of source code, this may make the program more efficient, which is important if you use a macro program to loop through hundreds of webpages.

SUBSTRN Function: We will use the SUBSTRN function to pull out only the text string from the remaining source code: `TEXT=SUBSTRN(WEBPAGE,1,FIND(WEBPAGE,"<")-1);`

- The syntax for the SUBSTRN function is: SUBSTRN(string, position ,< length>)
- Our string in this case, will be our variable WEBPAGE.
- Since our textual data is expected to be at the left-most position in our variable, our starting position will be 1.
- Again, since our textual data is expected to be between these symbols >*text string*<, we need the end of our substring to be before the < symbol.

- In order to input the numeric length of the substring we want to pull, we will use the FIND function, for which the syntax is: FIND(character-value, find-string <,'modifiers'><,start>)
- With the FIND function, we search our variable WEBPAGE for the first occurrence of the symbol <, which will return a numeric value into the third argument of the SUBSTRN function. However, the value that is returned by the FIND function will be the position of the symbol <, which we do not want to include in our substring. For this reason we subtract 1 from the value that is returned by the FIND function within this third argument of the SUBSTRN function. This will achieve our goal of extracting all characters between the two symbols >< into our new variable TEXT.

TRANWRD Function: To insert spaces and ampersand symbols into text in HTML and XML code, you use   and & respectively. We will use the TRANWRD function twice to remove each of these from the text strings that we have.

- The syntax for the TRANWRD function is: TRANWRD(source,target,replacement)
- The source will be our TEXT variable.
- The target will be   and &
- We will replace each of these with a space and an ampersand symbol respectively.
- If our statement were written as: `TEXT=TRANWRD(TEXT," "," ");` then SAS would interpret the   as a macro variable, not to mention that the semi-colon that we are searching to replace would also be interpreted as the end of the statement.

%NRSTR Function: In order to let SAS know to ignore these two symbols inside of our TRANWRD statement, we will need to use the %NRSTR function, with syntax %NRSTR (character-string)

- The %NRSTR function, will simple insert any character string enclosed in parentheses into your SAS statement, ignoring special characters.
- So our next two statements with the TRANWRD and %NRSTR functions will be:

```
TEXT=TRANWRD(TEXT,"%NRSTR( )"," ");
TEXT=TRANWRD(TEXT,"%NRSTR(&amp;)","&");
```

ANYALPHA and ANYDIGIT Functions: Since our previous statements have extracted the textual data we are looking for, we will use the following statement to delete rows with only source code that we want to exclude:

```
IF ANYALPHA(TEXT) + ANYDIGIT(TEXT) LT 1 THEN DELETE;
```

- The ANYALPHA and ANYDIGIT functions will return the numeric position of the first alphabetic or numeric character from a character string.
- If there are no alphabetic characters or numerals in our TEXT variable, then each of these functions will return a value of zero and the corresponding observations will be deleted by this single statement.

KEEP Statement: Lastly, we use a KEEP statement to keep only our TEXT variable and we will be able extract the textual data from most websites with just two data steps.

LIMITATIONS:

After utilizing the steps described in this paper, you may find there are still a few observations that contain source code that was not excluded from the final data set. Since there are multiple web languages, there is no one way to eliminate all source code without a list of all functions and macros used in each web language. However, the method described in this paper will get you very close to the final product of what you are attempting to do. You can add a few extra statements to customize your program to the websites you are attempting to input into SAS.

CONCLUSION:

Without purchasing expensive web crawler and text mining software, you can extract textual data from websites with two simple Data Steps in Base SAS:

```
FILENAME SOURCE URL "%STR(http://www.usatoday.com/)" DEBUG;
DATA SOURCE1;
      FORMAT WEBPAGE $1000.;
INFILE SOURCE LRECL=32767 DELIMITER=">";
INPUT WEBPAGE $ @@;
RUN;

DATA SOURCE2;
SET SOURCE1;
      WHERE WEBPAGE LIKE "_%<%";
      TEXT=SUBSTRN(WEBPAGE,1,FIND(WEBPAGE,"<")-1);
      TEXT=TRANWRD(TEXT,"%NRSTR( )"," ");
      TEXT=TRANWRD(TEXT,"%NRSTR(&amp;)","&");
      IF ANYALPHA(TEXT) + ANYDIGIT(TEXT) LT 1 THEN DELETE;
      KEEP TEXT;
RUN;
```

RECOMMENDED READING:

Bartlett, J., Bieringer, A., Cox, J., SAS Institute Inc. “Your Friendly Neighborhood Web Crawler: A Guide to Crawling the Web with SAS” Proceedings of the SAS Global Forum 2010. Cary, NC: SAS Institute Inc. Available at: http://support.sas.com/resources/papers/proceedings10/053-2010.pdf

Cody, R., SAS Institute Inc. SAS Functions by Example (Chapter 1: Character Functions). Second Edition 2010. Cary, NC: SAS Press, SAS Institute Inc. Available at: http://support.sas.com/publishing/pubcat/chaps/59343.pdf

Helf, G., Hitachi Global Storage Technologies, San Jose, CA. “Extreme Web Access: What to DO When FILENAME URL is Not Enough” Proceedings of the SAS Users Group International 30. Cary, NC: SAS Institute Inc. Available at: http://www2.sas.com/proceedings/sugi30/100-30.pdf