Paper 052-2009

# Creating SAS® Data Sets from HTML Table Definitions

Rick Langston, SAS Institute Inc.

## ABSTRACT

This paper is a follow-up to my 2007 paper "Handling Large Stream Files with the @'string' Feature." In this paper, I describe a SAS® program that will read an arbitrary HTML stream and create a separate SAS data set for each <TABLE> definition in the stream. The variable names are determined from the table headings provided in the HTML. I build upon the techniques first described in the 2007 paper and show new ways to accomplish the goals. Features shown include the MISSOVER option used in conjunction with COLUMN=, INPUT @'string', and INPUT @'expression'; the useful CAT and CATX functions; the ?? operator used with the INPUT function; and issues involving %INCLUDE versus CALL EXECUTE.

## INTRODUCTION

HTML tables are a common occurrence in web pages. They can contain any kind of tabular data, and you might want to extract that data into a SAS data set. Because the URL access method can be used to access Web site HTML streams, we have DATA step access to the HTML tables. With the macro described herein, SAS data sets can be automatically constructed from the HTML tables.

HTML tables are delimited by tags <TABLE> … </TABLE>. Each row of the table is delimited by tags <TR> … </TR>, and each row of the table has column detail data that is delimited by tags <TD> … </TD>. The data between the <TD>…</TD> tags are the values to be stored in individual variables. A given row corresponds to an observation in a SAS data set.

## DESCRIPTION OF THE MACRO

The SAS macro will read the contents of a URL, looking for all table definitions. It allows for the tags to be all uppercase or all lowercase. Within each table, the total number of row tags and max column tags within each row is determined. Then, a SAS data set is produced for each table, containing the number of determined rows and columns. The text of each column is stripped of any tags such as <small> or <a href=....>. If there are tables within tables, this will still work, but each table is treated as a separate entity and there is no apparent association between the tables. Each data set created is named table1, table2, and so on. The variable names are col1, col2, and so on. If a column is determined to be all blank, the variable is dropped. If the column appears to contain only dates, the variable is made numeric and is given a DATE9. format. If the column otherwise appears to contain only numeric data, the variable is made numeric but no format is associated.

## TECHNIQUES USED

There are several special techniques used in this macro implementation that might not be recognized by the reader, so they are explained in more detail below.

The contents of the URL are written to a temporary file byte by byte so we know the actual file size. Then all subsequent accesses of the file use RECFM=F LRECL=n COLUMN=C MISSOVER so that we treat the file as one big record. We need the @'text' feature of the INPUT statement, and because this feature does not work with RECFM=N, we use the RECFM=F and LRECL=n options instead. Using MISSOVER allows us to continue execution even though we hit EOF. The COLUMN= option will always let us know where we are.

In addition to INPUT @'text', we use the associated feature INPUT @(trim(…)). In general, an expression can appear after the @, and we are not limited to just a variable name or a quoted string. This feature proves very useful in our macro implementation.

The CAT and CATX functions are used throughout. These functions are new with SAS®9 and allow for simplified SAS code, because TRIM(LEFT(PUT(…))) can be replaced with a single function call.

This macro generates SAS code that is emitted to a temporary file (via the TEMP access method) and then includes that SAS code using the %INCLUDE statement. This method is used instead of CALL EXECUTE due to scoping issues for macro variables.

The macro makes use of the ?? operator in the INPUT statement. This operator ensures that the _ERROR_ flag is not set to 1 when invalid data are seen. We expect to encounter invalid data and do not want to have disruption in the logic.

**THE OUTER MACRO - %READHTML**

The %READHTML macro is the one the user will invoke. Within the macro body is the definition and invocation of the %READTABLE macro, which will be discussed later in this paper. %READHTML takes one argument, the URL containing table definitions.

```
%macro readhtml(url);
%global tabletag trtag tdtag closetags drops renames dates nobs ntables;
```

We start out with the FILENAME statement using the URL access method with the given URL. The entire contents of the web page is read into a temporary file, using the RECFM=F LRECL=1 attributes. This allows every byte to be properly counted for our complete file size. The FILESIZE macro variable is set to that file size, using the SYMPUTX CALL routine, which allows us to pass numeric values to be set as macro variable values (with trimmed leading and trailing blanks).

```
filename urltext url &url. &proxyinfo ;
filename myfile temp;
data _null_; infile urltext recfm=f lrecl=1 end=eof; file myfile recfm=f lrecl=1;
     input @1 x $char1.; put @1 x $char1.;
     if eof;
     call symputx('filesize',_n_);
     run;
```

This DATA step will make several passes through the file, looking for certain tags. First, it looks for <table, then <TABLE, then </table, and then </TABLE. HTML does not have case-sensitive tags, and any of these are possible. (In reality, mixed case such as <Table> are also permitted, but are not handled here). It is expected that an end tag (for example, </table) will be found since, otherwise, all tables are nested inside each other. We have to look for both 'TABLE' and 'table' since the subsequent use of input @'...' requires the proper casing. Note that the subsequent code expects that all tags of the same type (table, tr, td) use consistent casing throughout. This code will look for <tr> and <TR> tags and also <td> and <TD> tags and will output observations for each.

Note that the output observations always use uppercase for the tags, but the column number is saved as well. This code will also set the TABLETAG, TRTAG, and TDTAG macro variables with the proper casing. CLOSETAGS will be Y if end tags are found for TR and TD. NOBS is the number of observations emitted and NTABLES is the number of TABLE tags found. Since this code uses RECFM=F LRECL=&FILESIZE, the entire file is being treated as a single record. With the use of MISSOVER and COLUMN=, we will never go beyond the first record, and can conveniently reposition to column 1 for rescanning of the file. Note that the @(TRIM(TAG)) feature is used here, allowing for an expression. This is necessary since the tag might have trailing blanks that we do not want included in the search.

```
data detail(keep=text col);
     infile myfile recfm=f lrecl=&filesize. column=c missover;
     array which{3}    $8 _temporary_ ('table','tr','td');
     array whichsrc{3} $8 _temporary_;
     length tag $8;
     closetags='N';
     failure=0;
     do i=1 to 3;
        tag='<'||which{i};  link readfile; n_lower_open  = obscount;
        tag=upcase(tag);    link readfile; n_upper_open  = obscount;
        tag='</'||which{i}; link readfile; n_lower_close = obscount;
        tag=upcase(tag);    link readfile; n_upper_close = obscount;
        nobs+n_lower_open+n_upper_open+n_lower_close+n_upper_close;
        if which{i}^='table' and n_lower_close+n_upper_close>0 then closetags='Y';
        if (n_upper_open>0 and n_lower_open>0) or
           (n_upper_close>0 and n_lower_close>0) then do;
           put 'ERROR: There is a mixture of upper and lower case' which{i} ' tags';
           failure=1;
```

```
         end;
      if which{i}='table' then do;
         ntables=n_lower_open+n_upper_open;
         if ntables=0 then do;
            put 'ERROR: There are no tables defined in the HTML.';
            failure=1;
            end;
         else if n_upper_close+n_lower_close=0 then do;
            put 'ERROR: There are no closing tags for tables in the HTML.';
            failure=1;
            end;
         end;
      whichsrc{i}=which{i};
      if n_upper_open>0 then whichsrc{i}=upcase(whichsrc{i});
      end;
   call symput('tabletag',trim(whichsrc{1}));
   call symput('trtag',   trim(whichsrc{2}));
   call symput('tdtag',   trim(whichsrc{3}));
   call symput('closetags',closetags);
   call symput('nobs',     cats(nobs));
   call symput('ntables',  cats(ntables));
   if failure then abort;
   return;


readfile:;
   obscount=0;
   text=upcase(tag);
   input @1 @;
   do while(1);
      input @(trim(tag)) @;
      if c>&filesize then leave;
      col=c;
      obscount+1;
      output;
      end;
   return;
   run;
proc sort data=detail; by col; run;
```

This is the code that will determine how many rows and columns there are for each table. The %READTABLE macro will be invoked for each table based on that information so that the table1, table2, … tablen data sets can be created. This is done by first populating the taglist and tagstart arrays with the data from the detail data set. The tagend array elements are set based on the location of the end tags. We can then examine each <tr> tag and determine which table it is in. This is done by searching through the tablestart/tableend array to find a column range that contains the <tr> tag location. Note that multiple tables can contain this <tr> tag if a table is defined within another table, so we look for the surrounding table that is the smallest. Once we know the proper table, we increment the row count. Any <td> tags encountered will cause an incrementation of column count for the same table. Note that the column count is reset to 0 for each row and re-incremented since some rows might not contain all columns. This code also generates a series of %readtable macro invocations, but places them in the SASCODE temporary file for a later %INCLUDE. Although one could use CALL EXECUTE, I prefer to use the %INCLUDE statement with the /SOURCE2 option to have a better understanding of the code being invoked.

```
filename sascode temp;
data _null_;
   array taglist{&nobs} $8 _temporary_;      * tag text;
   array tagstart{&nobs} _temporary_;        * start loc for the tag;
   array tagend{&nobs} _temporary_;          * end loc for the tag;
   array tablestart{&ntables} _temporary_;   * start loc for each table;
   array tableend  {&ntables} _temporary_;   * end loc for each table;
   array tablenrows{&ntables} _temporary_;   * no. of rows in the table;
```

```
array tablencols{&ntables} _temporary_;   * no. of cols in the table;

*-----populate the arrays from the detail data set-----*;
do i=1 to &nobs;
   set detail point=i;
   taglist{i}=text;
   tagstart{i}=col;
   end;

*-----determine the end location for each tag if end tags given-----*;
do i=1 to &nobs;
   if taglist{i}=:'</' then do j=i-1 to 1 by -1;
      if substr(taglist{j},2)=substr(taglist{i},3) then do;
         tagend{j}=tagstart{i}-length(taglist{i});
         leave;
         end;
      end;
   end;

*-----set the table start/end arrays-----*;
j=0;
do i=1 to &nobs;
   if taglist{i}='<TABLE' then do;
      j+1;
      tablestart{j}=tagstart{i};
      tableend{j}=tagend{i};
      end;
   end;

jj=0;
do i=1 to &nobs;

   *-----find smallest table containing each <tr tag-----*;
   if taglist{i}='<TR' then do;
      minsize=1e10;
      do j=1 to &ntables;
         if tablestart{j}<=tagstart{i}<=tableend{j} then do;
            size=tableend{j}-tablestart{j}+1;
            if size<minsize then do;
               jj=j;
               minsize=size;
               end;
            end;
         end;
      if jj>0 then do;
         tablenrows{jj}+1;
         end;
      ncols=0;
      end;

   *-----increment column count for the <td tags-----*;
   else if jj>0 and taglist{i}='<TD' then do;
      ncols+1;
      tablencols{jj}=max(tablencols{jj},ncols);
      end;
   end;

*-----determine if there is overlap (which would be a problem)-----*;
overlap=0;
do i=1 to &ntables;
   put tablestart{i}= tableend{i}= tablenrows{i}= tablencols{i}=;
```

```
         if i>1 and tableend{i-1}>tablestart{i} then overlap=1;
         else if i<&ntables and tablestart{i+1} < tableend{i} then overlap=1;
         end;
      put overlap=;

      file sascode;
      do i=1 to &ntables;
         if tablestart{i}>0 and tableend{i}>0 and tablenrows{i}>0 and tablencols{i}>0
            then do;
            args=catx(',',i,tablestart{i},tableend{i},tablenrows{i},tablencols{i});
            put '%readtable(' args ');';
            end;
         end;

      stop;
      run;
proc delete data=detail; run;
```

At this point, we can include the generated SAS code using %INCLUDE to produce all the SAS data sets from the various table definitions. The SAS code consists of multiple %READTABLE invocations.

```
*-----invoke the generated code that calls the readtable macro-----*;
%include sascode/source2; run;
filename sascode clear;

%mend readhtml;
```

**THE INNER MACRO %READTABLE**

The %READTABLE macro will be invoked via %INCLUDE for each table definition in the HTML. The start and end column of the file (containing the entire table definition) is specified, along with the number of rows and columns so that the SAS data set can be properly defined with variables col1-coln. As in %READHTML, we read the entire file as a single record.

```
%macro readtable(tablenum,start,end,nrows,ncols);
data table&tablenum.;
      infile myfile recfm=f lrecl=&filesize. column=c missover;
      array col{*} $200 col1-col&ncols.;
      keep col1-col&ncols.;

      *-----start at the beginning of our table-----*;
      input @&start @;
      endrow=.;
```

```
*-----read each row-----*;
do i=1 to &nrows;

   *-----row starts with <TR or <tr tag-----*;
   input @"<&trtag" @;
   startcol=c;

   *-----determine where to stop, using </tr, next <tr, or next <table-----*;
   %if &closetags.=Y %then %do;
   input @"</&trtag" @;
   endrow=c-4;
   %end;
   %else %do;
   if i<&nrows then do;
      input @"<&trtag" @;
      endrow=c-4;
      end;
   else do;
      input @"<&tabletag " @;
      endrow=c-7;
      end;
   %end;

   *-----go back to start reading contents of row-----*;
   input @startcol @;

   *-----read all the column data for the row-----*;
   do j=1 to &ncols;

      *-----col starts with <TD or <td tag-----*;
      input @"<&tdtag" @;

      *-----blank out remaining columns if we hit the end-----*;
      if c>=endrow then do;
         do k=j to &ncols;
            col{j}=' ';
            end;
         input @endrow @;
         leave;
         end;

      *----get past end of tag-----*;
      input @'>' @;
      startcol=c;

      *-----compute where to end the column data using </td, <tr, or <table----*;
      %if &closetags.=Y %then %do;
      input @"</&tdtag" @;
      %end; %else %do;
      if j<&ncols then input @"<&tdtag" @;
      else if i<&nrows then input @"<&trtag" @;
      else input @"</&tabletag" @;
      %end;
```

```
         *-----read everything between----*;
         l=c-5-startcol+1;
         input @startcol text $varying32767. l @;

         *-----remove the prefixing tags like <small>, <a href=...>, etc.-----*;
         do while(left(text)=:'<');
            text=substr(text,index(text,'>')+1);
            end;

         *-----remove everything after a trailing <-----*;
         k=index(text,'<');
         if k then substr(text,k)=' ';

         *-----change escape sequences to the right characters-----*;
         text=tranwrd(text,'&amp;','&');
         text=tranwrd(text,'&lt;','<');
         text=tranwrd(text,'&rt;','>');
         text=tranwrd(text,' ',' ');

         *-----remove any stray crlf chars and convert tabs to blanks-----*;
         text=compress(text,'0d0a'x);
         text=translate(text,' ','09'x);

         *-----save this as our column value-----*;
         col{j}=text;
         end;
      output;
      end;
   stop;
   run;
```

Here, we attempt to improve the characteristics of the SAS data set. If a column is completely blank, we remove the variable. If a column has only numeric values, we change the type to numeric. If a column consists solely of dates, we convert to the proper SAS date values and associate the proper format. This conversion is done by introducing a mirror set of numeric variables (numcol1-numcoln) and setting them to the numeric value representation of col1-coln. If all values are found to be numeric, the &DROPS macro variable contains the DROP= list of col* variables, and the &RENAMES macro variable contains the list of renames from numcol* to col* variables. The &DATES macro variable contains a list of all variables whose every value could be successfully informatted to a date value via the ANYDTDTE informat.

```
data table&tablenum.; set table&tablenum. end=eof;
   array col{*} col1-col&ncols.;
   array numcol{*} numcol1-numcol&ncols.;
   keep col1-col&ncols. numcol1-numcol&ncols.;
   array status{&ncols.} $1 _temporary_;
   length text $1024;
   do i=1 to &ncols;
      if status{i}='C' then continue;
      text=left(col{i});
      numcol{i}=.;
      if text=' ' then continue;
      if status{i}=' ' then do;
         link try_numeric;
         if numcol{i}^=. then do;
            status{i}='N';
            end;
         else do;
            link try_date;
            if numcol{i}^=. then do;
               status{i}='D';
               end;
```

```
            end;
         if status{i}=' ' then status{i}='C';
         end;
      else if status{i}='D' then do;
         link try_date;
         if numcol{i}=. then do;
            status{i}='C';
            end;
         end;
      else if status{i}='N' then do;
         link try_numeric;
         if numcol{i}=. then do;
            status{i}='C';
            end;
         end;
      end;
   output;
   if eof;


   length renames drops dates $32767;
   do i=1 to &ncols;
      if status{i}='N' or status{i}='D' then do;
         renames=cat(trim(renames),' numcol',i,'=col',i);
         drops=cat(trim(drops),' col',i);
         end;
      else if status{i}=' ' then do;
         drops=cat(trim(drops),' col',i,' numcol',i);
         end;
      else if status{i}='C' then do;
         drops=cat(trim(drops),' numcol',i);
         end;
      if status{i}='D' then do;
         dates=cat(trim(dates),' col',i);
         end;
      end;
   if drops^=’ ‘ then drops='drop='||drops;
   if renames^=' ' then renames='rename=('||trim(renames)||')';
   if dates^=' ' then dates='format '||trim(dates)||';';
   call symput('drops',trim(drops));
   call symput('renames',trim(renames));
   call symput('dates',trim(dates));
   return;

 /* The TRY_NUMERIC link will use BEST32. on the field to see if it converts
    to a number. We use the INPUT function with the ?? operator to indicate
    that _ERROR_ will not be set and no warning message will appear about
    invalid data. This link will not be invoked if text is blank, so any
    other text causing numcol to become missing indicates an invalid numeric
    value (except for ., which we will assume here to mean non-numeric). The
    TRY_DATE link does the same except it uses ANYDTDTE, which allows for many
    different types of date representations, such as 2008/01/02 or 02JAN2008. */
try_numeric:;
   numcol{i}=input(text,?? best32.);
   return;
try_date:;
   numcol{i}=input(text,?? anydtdte32.);
   return;
   run;
```

```
*-----recreate the data set with the changes-----*;
data table&tablenum.; set table&tablenum.(&drops &renames);
     &dates;
     run;

*-----print the resultant table-----*;
options nocenter;
proc print data=table&tablenum.; title "table&tablenum."; run;
%mend readtable;
```

**SAMPLE INVOCATION**

Here we read from the European Central Bank Web site, from their page that contains exchange rates for the euro.

```
%readhtml('http://www.ecb.int/stats/exchange/eurofxref/html/index.en.html');
```

This resulted in a SAS data set containing col1 with the 3-character currency code, col2 containing the character description of the currency, and col3 as a numeric variable containing the exchange rate constant for that currency:

```
col1    col2                        col3

USD     US dollar                     1.57
JPY     Japanese yen                162.97
BGN     Bulgarian lev                 1.96
```

**CONCLUSION**

The %READHTML macro is a convenient way to create SAS data sets from HTML tables that can appear on Web sites. This macro uses a variety of techniques to produce an end result in the form of a SAS data set with numeric and date values whenever feasible.

## CONTACT INFORMATION

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

Rick Langston
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: rick.langston@sas.com