Paper 027-2007

Handling Large Stream Files with the @'string' Feature

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

This presentation examines a SAS® code example that demonstrates how to read a large HTML stream file (that doesn't contain carriage returns) by using the @'string' feature of the **INPUT** statement. Part of the discussion involves the use of the **LRECL=** option, and how to deal with searches that reach the end of file (EOF).

INTRODUCTION

HTML files are not required by browsers to contain carriage returns and line feeds (CRLFs). If CRLFs exist in the HTML file, they are ignored by the browser. However, if a DATA step is written to extract information from an HTML file, the DATA step needs CRLFs because the DATA step typically expects to encounter individual records in a file, not a continuous stream file without CRLFs. This paper examines a DATA step that operates on the HTML file as a continuous stream file without CRLFs.

THE HTML FILE

The HTML file looks like this; notice that there are no CRLFs in the file:

```
<HTML>
<HEAD>
<TITLE>This is a title</TITLE>
</HEAD>
<BODY>
<TABLE>
<TR>
<TD>1</TD>
<TD>2</TD>
<TD>3</TD>
</TR>
<TR>
<TD>4</TD>
<TD>5</TD>
<TD>6</TD>
</TR>
</TABLE>
<TABLE>
<TR>
<TD>7</TD>
<TD>8</TD>
<TD>9</TD>
</TR>
<TR>
<TD>10</TD>
<TD>11</TD>
<TD>12</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

We want to read the detail values (information between the <TD> and </TD> tags) and associate them with the proper table row number and column number. Each new row begins with a <TR> tag, and each new column begins with a <TD> tag.

Let's assume that the HTML stream file is well constructed, and that there are no stray tags. Let's also assume that the tags are all in uppercase. The purpose in the subsequent SAS code is to extract data, not to validate the HTML.

USING @'string' IN THE INPUT STATEMENT

Because we want to look for HTML tags in the HTML stream file, we can use the @'string' feature of the INPUT statement. Consider the following statement:

```
INPUT @'<TABLE>'
```

This statement positions us at the first column after the tag. It will then go from one record to the next record, as necessary, to find the text string. This is very convenient for coding the parsing of the HTML text. Note, however, that the text string being searched is case-sensitive, while HTML tags are not. For example, the <Table> tag will not be found by the above **INPUT** statement, even though the <Table> tag is perfectly acceptable to a browser and is considered equivalent to the <TABLE> tag.

EXAMINING THE HTML FILE

The way that a DATA step handles a stream file is by using **RECFM=N**. The **RECFM=N** option enables you to read the HTML file randomly and without record orientation. However, you cannot use the @'string' feature of the INPUT statement if you are using the **RECFM=N** option. The two are mutually exclusive. Therefore, we must find an alternative to the **RECFM=N** option in order to use the @'string' feature.

The **RECFM=** choices are F (fixed-length records) or V (variable-length records). You use **RECFM=V** if the file has CRLFs. We know that our file does not have CRLFs. If we determine the length of the file and use **RECFM=F**, we can also use the **LRECL=** option, and the DATA step will treat the entire file as a single record.

A quick way to determine the length of a file is through the FILESIZE attribute. We can access the FILESIZE attribute by using the **FOPEN**, **FOPTNUM**, **FOPTNAME**, and **FINFO** functions as follows, assuming that the file is called myfile:

```
data _null_;
  length filesize $200;
  filesize='1000000';
  found=0;
  fid=fopen('myfile');
  do i=1 to foptnum(fid);
     option=foptname(fid,i);
     info=finfo(fid,option);
     if option=:'File Size' then do;
        filesize=info;
        found=1;
        leave;
        end;
     end;
  if ^found
     then put 'Could not determine filesize, will use 1000000.';
  call symput('filesize',filesize);
  run;
```

The **FOPEN** function opens the file and returns a handle (which we put into the variable **FID**). The **FOPTNUM** function returns the number of option values that can be obtained from the file. Use **FOPTNAME** for each possible option to get option names. Then, use **FINFO** for each option's value, using the option name. The option name we are looking for is **File Size**, and the option value associated with **File Size** is the number of bytes in the file. Note that the **File Size** option works equally well in Windows and UNIX environments, although it is not available in z/OS. The code assumes an arbitrary file size of **1000000** bytes if the actual file size cannot be determined.

This DATA step does not read any data from the file.

Save the value of **File Size** into the macro variable &FILESIZE for later reference.

READING THE HTML FILE

Here is the SAS code that reads the detail values and associates the row and column values.

```
data temp(keep=table row col value);
     infile myfile recfm=f lrecl=&filesize. column=c eof=abc;
     length which $8;
     no_more_tables = 0;
     do while(1);
        which = 'TABLE';
        table+1;
        input @ '<TABLE>' @;
        if no_more_tables then leave;
        start_table = c;
        input @ '</TABLE>' @;
        end_table = c;
        input @start_table @;
        row=0;
        do while(1);
           row+1;
           which = 'TR';
           input @ '<TR>' @;
           if c >= end_table then do;
              input @end_table @;
              leave;
              end;
           start_row = c;
           input @ '</TR>' @;
           end_row = c;
           input @start_row @;
           col=0;
           which = 'TD';
           do while(1);
              col+1;
              input @ '<TD>' @;
              if c >= end_row then do;
                 input @end_row @;
                 leave;
                 end;
              start_col = c;
              input @ '</TD>' @;
              l = c-6-start_col+1;
              input @start_col valuec $varying20. l @;
              value=input(valuec,best20.);
              output;
              end;
           end;
        end;
     return;
abc:;
     if which = 'TABLE'
        then no_more_tables = 1;
     else if which = 'TR'
        then input @end_table @;
     else if which = 'TD'
        then input @end_row @;
     return;
     run;
```

The main **DO WHILE(1)** loop is used to process each **<TABLE>** definition. When there are no more tables, the loop terminates and the DATA step ends. The **WHICH** variable indicates which part of the HTML stream we are in—**TABLE**, **TR** (row), or **TD** (column). The **COLUMN=** option is very important here. After the @'string' feature positions us after the tag, the **COLUMN=** variable **C** is set to this location. We look for the end tag, such as </TABLE>, and find its column location, and then we deduce that the data we're looking for must be between the start and end tag.

We use the **RECFM=F** and **LRECL=&FILESIZE** options so that we can treat the entire input file as a single record. We use the **EOF=** option to prevent us from falling off the end—that is, when we look for <TABLE>, <TR>, or <TD>, and we do not find any of these tags, we create an end-of-file (EOF) condition, and the code at the **EOF=** option is executed. The code for the label **ABC** repositions us to the end of the higher-level section (end of row for TD and end of table for TR), or it marks that there are no more tables for TABLE. The DATA step will then resume execution.

Notice the code section that reads the numeric values. The numeric value is immediately followed by a **<**, with no intervening blank space. This means that we cannot use **LIST INPUT** to read the numeric value, because the **<** causes an invalid data condition. Instead, you should determine the length of the text between the > and the < and use $VARYING to read only that text into a character variable. Then, use the **INPUT** function to read the numeric value from the character variable.

A subsequent DATA step that displays the values of the output data set will show the following observations:

```
table=1 row=1 col=1 value=1
table=1 row=1 col=2 value=2
table=1 row=1 col=3 value=3
table=1 row=2 col=1 value=4
table=1 row=2 col=2 value=5
table=1 row=2 col=3 value=6
table=2 row=1 col=1 value=7
table=2 row=1 col=2 value=8
table=2 row=1 col=3 value=9
table=2 row=2 col=1 value=10
table=2 row=2 col=2 value=11
table=2 row=2 col=3 value=12
```

CONCLUSION

The judicious use of **RECFM=F**, **LRECL=**, **COLUMN=**, and **EOF=**, along with the @'string' feature of the **INPUT** statement, allow us to extract data from HTML stream files without CRLFs.

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