Reading From Alternate Sources:
What To Do When The Input Is Not a Flat File

Michael Davis, Ambler, PA

ABSTRACT

Most SAS® programmers are comfortable creating applications when the input is in a SAS data set or flat file on the same computer. However, what do you do when your data is to be accessed from a web page or TCP socket? Can you read data on another computer accessible via FTP without first transporting the file? How do you read a SAS catalog entry? What if the data is from a device connected to your computer via an RS-232 interface? How about reading database files as if they were SAS data sets.

This presentation will illustrate SAS software features to handle these and other data input situations. Special attention will be paid to new features introduced with Versions 7 and 8 (and available in V9), such as the SAS/ACCESS® LIBNAME engines.

FTP Access Method

Most persons who have been using SAS software for an extended period are comfortable writing programs that use flat files as input. But the level of comfort often drops rapidly when the input source is not on the local drives or mapped network drives.

Many of those computer users, especially those with some familiarity with Unix operating systems, are acquainted with FTP. FTP, which stands for File Transfer Protocol, is a technique which allows for the transfer of files to and from a remote network site.

Perhaps the most obvious technique would be to use FTP to transfer the input file to a local drive or a network drive mapped to the computer on which SAS is installed. The drawback is that if the input file is large, it can be inconvenient to park the file prior to reading it.

SAS has the ability to define an FTP data source as a fileref. This feature has been available since Version 6. The FTP access method can be used to define a fileref with any remote computer accessible by TCP/IP

TCP/IP (Transport Control Protocol/Internet Protocol) is often called TCP. TCP is a widely used protocol, employed for both corporate network and dial-up connections.

The general syntax for a FILENAME statement that employs the FTP Access Method is:

```
FILENAME fileref FTP 'external-file' <ftp-options> ;
```

Commonly used FTP options include:

cd=	set current directory
debug	show informational messages
host=	set host name or IP address
pass=	specify password
recfm=	record format, f - fixed, v - variable
user=	specify user id (account)

The debug option is very useful when setting up a connection with a new FTP host. This option causes all of the messages sent to and received from the FTP to be shown in the SAS Log.

For a complete list of options that may be used with the FTP Access Method, consult the SAS OnlineDoc®. Navigate to the relevant portion by selecting -> Base SAS Software -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Statements -> FILENAME, FTP Access Method.

If you use SAS primarily under Microsoft Windows™, you may be thinking, “This is great but since my company does not provide an FTP server, I have no way to actually try out the FTP Access Method.”. Well, if you have access to SAS Version 6 (e.g, Release 6.12) or later and any form of Internet connection (even a dial-up modem connection), you should be able to run the following example.

The example employs a SAS data set of 1986 Major League Baseball batter statistics, converted to a comma-separated value (CSV) text file. The file is hosted on an FTP server that can be accessed with the following FILENAME statement.

```
filename myfile ftp 'baseball.csv'
 user= 'ftpdownload'
 pass= 'test123'
 host='bassettconsulting.com'
 recfm=v debug ;
```

The baseball.csv file is found in the anonymous subdirectory on an FTP server referenced by the internet domain bassettconsulting.com (current IP address 209.35.121.141). Being a comma-separated file, the length of each record varies, hence the variable-length record format (recfm=v). The debug parameter is optional. It is included in this example for instructional purposes and is usually omitted from production code.

A comma-separated value example is shown in part because it allows for the columns names to be passed within the file. Also, the SAS code to read the file can be automatically generated by the Import Wizard. The Import Wizard is available from the File pmenu in SAS Release 6.12 and later versions. An edited version of the code it generated is shown below.

```
data WORK.baseball ;
    infile myfile delimiter = ','
      MISSOVER DSD lrecl=32767
      firstobs=2 ;
       informat NAME $20. ;
       informat TEAM $12. ;
       informat NO_ATBAT best32. ;
       informat NO_HITS best32. ;
       informat NO_HOME best32. ;
       informat NO_RUNS best32. ;
       informat NO_RBI best32. ;
       informat NO_BB best32. ;
       informat YR_MAJOR best32. ;
       informat CR_ATBAT best32. ;
       informat CR_HITS best32. ;
       informat CR_HOME best32. ;
       informat CR_RUNS best32. ;
       informat CR_RBI best32. ;
       informat CR_BB best32. ;
       informat LEAGUE $8. ;
       informat DIVISION $4. ;
       informat POSITION $2. ;
       informat NO_OUTS best32. ;
       informat NO_ASSTS best32. ;
       informat NO_ERROR best32. ;
       informat SALARY best32. ;
       format NAME $20. ;
       format TEAM $12. ;
       format NO_ATBAT best12. ;
       format NO_HITS best12. ;
       format NO_HOME best12. ;
       format NO_RUNS best12. ;
       format NO_RBI best12. ;
       format NO_BB best12. ;
       format YR_MAJOR best12. ;
       format CR_ATBAT best12. ;
       format CR_HITS best12. ;
       format CR_HOME best12. ;
       format CR_RUNS best12. ;
       format CR_RBI best12. ;
       format CR_BB best12. ;
       format LEAGUE $8. ;
```

```
       informat DIVISION $4. ;
       informat POSITION $2. ;
       informat NO_OUTS best32. ;
       informat NO_ASSTS best32. ;
       informat NO_ERROR best32. ;
       informat SALARY best32. ;
       format NAME $20. ;
       format TEAM $12. ;
       format NO_ATBAT best12. ;
       format NO_HITS best12. ;
       format NO_HOME best12. ;
       format NO_RUNS best12. ;
       format NO_RBI best12. ;
       format NO_BB best12. ;
       format YR_MAJOR best12. ;
       format CR_ATBAT best12. ;
       format CR_HITS best12. ;
       format CR_HOME best12. ;
       format CR_RUNS best12. ;
       format CR_RBI best12. ;
       format CR_BB best12. ;
       format LEAGUE $8. ;
       format DIVISION $4. ;
       format POSITION $2. ;
       format NO_OUTS best12. ;
       format NO_ASSTS best12. ;
       format NO_ERROR best12. ;
       format SALARY best12. ;
    input
       NAME $ TEAM $ NO_ATBAT
       NO_HITS NO_HOME NO_RUNS
       NO_RBI NO_BB YR_MAJOR
       CR_ATBAT CR_HITS CR_HOME
       CR_RUNS CR_RBI CR_BB
       LEAGUE $ DIVISION $
       POSITION $ NO_OUTS NO_ASSTS
       NO_ERROR SALARY ;
    run;
```

Although it is beyond the scope of this paper, the FTP file access method can be used to create files as well as read them. However, the anonymous FTP access supplied in the example FILENAME statement only allows read-access.

Fans of “big iron” might wonder at this point whether the FTP Access Method is available to them. The good news is that users of SAS on mainframes are often able to use the FTP Access Method as well. On the following page, an example is shown to demonstrate the use of the FTP Access Method is shown for an IBM z/OS (“mainframe”) computer.

```
/* set the FTP fileref */

filename rsrawdat ftp
  "'<data set name>'"
  user='<user account>'
  host='<IP address>'
  prompt rcmd='site rdw' debug ;

/* read the FTP data */

data testraw ;
  infile rsrawdat lrecl=1080
  missover ;
  input
   @1  var1 $ebcdic2.
   @3  var2 s370fpd5.0
   @8  var3 s370fpd4.0
   @12 var4 $ebcdic5.
   @17 var5 $ebcdic21.
   @38 var6 $ebcdic8.
   @46 var7 $ebcdic4.
   <…more variables read…>
   ;
run ;
```

In the mainframe FTP Access Method example above, please notice the use of double quotes enclosing the single quotes surrounding the data set name. This is because with z/OS, if a data set name is not quoted, the operating system will prefix the data set name with the user account, which is not desired.

The prompt option was used in this example so that it was not necessary to embed the password in the FILENAME statement. Also notice the use of the rcmd= (remote command) option. The option rcmd='site rdw' is required for IBM FTP servers. It causes the record descriptor word to be included as part of the data. The record descriptor word, which is needed, would otherwise be removed.

Moving to the data step portion of the above sample, please notice the use of the $ebcdic and s370fpd informats. As those familiar with IBM mainframes may know, characters in their files are encoded using EBCDIC representation instead of ASCII. If the information in mainframe files are being moved using SAS/CONNECT® or a terminal emulation program, the translation from EBCDIC to ASCII would probably be handled transparently.

However, when FTP is used to transport a file from a mainframe to either a personal computer or a mid-range Unix server, the EBCDIC character representation remains. So instead of translating the contents of the file character by character, we use the $ebcdic and S370fpd informats.

In our mainframe FTP Access Method example, a SAS supplied utility, COB2SAS ("COBOL to SAS") was used to convert the COBOL copybook (sometimes known as an "RD" or "Record Description") into a SAS INPUT statement. COB2SAS was employed primarily to automatically calculate the input field offsets and informat widths. The resulting INPUT statement was edited to substitute $ebcdic informats for the COB2SAS-generated character informats. The $ebcdic translates the EBCDIC character data to ASCII. Similarly, the numeric fields were converted into s370fpd informat, which reads packed data in IBM mainframe format

There are two other cautions to observe when using the FTP Access Method with mainframe files. First, this access method only works with disk files. Tape data cannot be accessed via FTP. Second, at many sites, mainframe disk files are periodically archived. So before employing the FTP Access Method, it may be necessary to issue an HRECALL or similar command to cause an archived file to be restored to disk.

URL Access Method

How often have you looked at a web page in a browser and said to yourself, "I wish that I could use that as input to a SAS program". The need for this service has grown with growth of Common Gateway Interface (CGI) applications such as SAS/IntrNet®. CGI tools and other web technologies can serve as an application program interface (API) for computer programs to pass parameters to an application server and return the results to a web browser window.

The application server usually returns the results in the form of a hypertext mark-up language (HTML) table. For the results to be used as part of a program to program interface, we need an automated way to read the character stream returned from the CGI or other web application.

Fortunately, the fine folks who brought us the FTP Access Method have also have provided the URL Access Method. URLs (Universal Resource Locators) are the "http://…" strings that either you or the previous web page types into the location box at the top of a browser to show the desired web page.

Similar to the FTP Access Method, the URL Access Method is defined through a FILENAME statement.

The syntax for the FILENAME statement employing the URL Access Method is:

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

If you wish, the "URL" after the fileref can be replaced with "HTTP". The "external-file" is the URL of the web page to be read. The format consists of

http://<*hostname*>/<*filename*>

plus any additional parameters which may follow after a question mark (?) delimiter. If the port for the HyperText Transfer Protocol Daemon (HTTPD) needs to be specified, the "external-file" format becomes:

http://<*hostname*>:<*port*>/<*filename*>

The most common HTTPD port is 80. We will return to this subject at the end of URL Access Method section.

The URL Access Method is often used without any options. At some sites, the proxy= option may be required to accommodate a proxy server. For a complete list of options that may be used with the URL Access Method, consult the SAS OnlineDoc. Navigate to the relevant portion by selecting -> Base SAS Software -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Statements -> FILENAME, URL Access Method.

Unlike the FTP Access Method, the tricky part is not making the connection but parsing the returned byte stream and converting it into a SAS data set. Fortunately, HTML pages contain tags, denoted by the "<" and ">" characters. Scanning the URL byte streams for selected tags, coupled with an understanding of HTML tables and other formatting techniques allow us to extract the desired information as a SAS data set.

The heart of the typical page returned from a CGI application, such as SAS/IntrNet, or from the SAS web-formatting macros, or from the SAS Output Delivery System (ODS) is an HTML table. Within an HTML page, tables are enclosed between <TABLE> and </TABLE> tags. Each row in a table is enclosed by <TR> and </TR> tags. Within each row, each cell is enclosed between <TD> and </TD> tags.

For the purpose of demonstrating the URL Access Method, the author created an HTML page from the baseball SAS data set used earlier in this paper. He used ODS to create the HTML page. The page can be viewed at the following URL:

http://bassettconsulting.com/baseball.htm

The page is 843 KB so it may take a few seconds to view if the web browser connected to the Internet via a dial-up connection.

While not a requirement of HTML, the output generated by SAS ODS puts the <TD> and <TR> tags at the start of a new line. The data and formatting tags for each cell appears on a separate line. As a final stroke of luck, all of the data cells, and only the data cells are prefixed by a font tag which ends with:

color="#000000"

One last detail that might be useful to some readers is that all of the <TD> tags contain "ALIGN=LEFT" when the cell contains character data and "ALIGN=RIGHT" when the data is numeric. While useful in some applications, this information was not needed for the SAS coding shown subsequently.

One of the techniques that the author uses to "fine-tune his programming approach is a "quick and dirty" program such as the following example to generate a data set to show the position of all the relevant words.

```
filename readhtml url
'http://bassettconsulting.com/base
ball.htm' debug;

data testhtml(drop=buffer) ;
  length buffer $ 200 word $ 25 ;
  infile readhtml lrecl=200 pad ;
  input @1 buffer 200. ;
  if input(buffer,$3.) eq '<TD' ;
  word= scan(buffer, 1, ' <>"') ;
  if word eq 'TD' then
   do i = 1 to 20 ;
    word= scan(buffer, i,' <>"') ;
    output ;
  end ;
run ;
```

Please note that the above Filename statement is wrapped to the next line only to fit into the column.

In this example, the baseball.html file is opened as a URL fileref. Input rows that do not begin with the <TD> tag are immediately dropped. The resulting SAS data set, testhtml, shows the contents of each "word" identified by the Scan function. The Scan function is using blanks, "<", ">", and double

quotation marks as the delimiter. The loop variable "i" shows the position of each word.

The test program shows that for the rows that contain data to be extracted, the word "#000000", which set the color of the displayed font to black, is the thirteenth word extracted. The fourteenth word is always the data to be extracted. However, in the case of the player name, the fourteenth word is the player's last name, with a comma. The player's first name is the fifteenth word.

So the first part of our program to extract the table values from the baseball.html page should be revised as shown in the following version:

```
filename readhtml url
'http://bassettconsulting.com:80/bas
eball.htm' debug;

data testhtml(drop=buffer) ;
  length buffer     $ 200
         word word2 $  25 ;
  infile readhtml lrecl=200 pad ;
  input @1 buffer 200. ;
  if input(buffer,$3.) eq '<TD' ;
   do ;
    word= scan(buffer,13,' <>"') ;
    if word eq '#000000' then do ;
      word= scan(buffer,14,' <>"') ;
      word2=scan(buffer,15,' <>"') ;
      output ;
    end ;
  end ;
run ;
```

The resulting data set, testhtml, has 7084 observations, one observation for each of the 22 variables in the baseball data set. However, what we really wanted was just one observation with 22 variables for each row of data shown in on the web page. The following program transforms the testhtml data into the desired format.

```
data testhtml2(drop=word word2 count)
;
   length
     name       $ 18
     team       $ 12
     no_atbat no_hits no_home
     no_runs no_rbi no_bb yr_major
     cr_atbat cr_hits cr_home
     cr_runs cr_rbi cr_bb 8
     league division position $ 8
     no_outs no_assts no_error
     salary 8 ;
   retain
     name team no_atbat no_hits
     no_home no_runs no_rbi no_bb
     yr_major cr_atbat cr_hits
     cr_home cr_runs cr_rbi
     cr_bb league division
     position no_outs no_assts
     no_error salary ;
   retain count 0 ;
   format salary 8.2 ;
   set testhtml ;
   if count eq  0 then name =
    trim(word) || ' ' || word2 ;
   else if count eq  1 then
    team = word ;
   else if count eq  2 then
    no_atbat = input(word, 8.) ;
   else if count eq  3 then
    no_hits = input(word, 8.) ;
   else if count eq  4 then
    no_home = input(word, 8.) ;
   else if count eq  5 then
    no_runs = input(word, 8.) ;
   else if count eq  6 then
    no_rbi = input(word, 8.) ;
   else if count eq  7 then
    no_bb = input(word, 8.) ;
   else if count eq  8 then
    yr_major = input(word, 8.) ;
   else if count eq  9 then
    cr_atbat = input(word, 8.) ;
   else if count eq 10 then
    cr_hits = input(word, 8.) ;
   else if count eq 11 then
    cr_home = input(word, 8.) ;
   else if count eq 12 then
    cr_runs = input(word, 8.) ;
   else if count eq 13 then
    cr_rbi = input(word, 8.) ;
   else if count eq 14 then
    cr_bb = input(word, 8.) ;
   else if count eq 15 then
    league = word ;
   else if count eq 16 then
    division = word ;
   else if count eq 17 then
    position = word ;
   else if count eq 18 then
    no_outs = input(word, 8.) ;
   else if count eq 19 then
    no_assts = input(word, 8.) ;
   else if count eq 20 then
    no_error = input(word, 8.) ;
   else if count eq 21 then
    salary = input(word, 8.2) ;
   count+1 ;
   if count ge 22 then do ;
      count= 0 ; output ;
   end ;
run ;
```

The preceding code was customized for the baseball data set and would have to be adapted for different web pages. If the web page had been produced by a different technique, different "landmarks" may be required. However, the general approach described can be used to extract tables from other web pages.

When testing the code when this paper was created, the author received the error message that indicated that the httpd service (daemon) was not available. A search of the SAS Technical Support web site identified note V6-SYS.SYS-C065, which also applied to Version 8. Adding the line:

```
httpd 80/tcp  #world wide web access
```

to the computer's services file solved the problem. An alternative would have been to append the port number (:80) the domain in the URL supplied in the FILENAME statement. The example furnished on the previous page shows this technique.

One last caveat about the URL Access Method is that in contrast with the FTP Access Method, the URL Access Method only allows one to read web data. For those who have a requirement to use SAS to update the content of a website, many situations can be handled by using the FTP Access Method to transfer SAS created output to the web host.

For those whose requirements cannot be met by the FTP and URL Access Methods, another way to communicate with web hosts follows.

Socket Access Method

At Northeast SAS Users Conference held in Philadelphia in 2000, David Ward presented a wonderful paper on using the Socket Access Method, which is cited in the bibliography. Those who are interested in experimenting with and perhaps using the Socket Access Method are commended to seek a copy of David's paper, along with consulting the the SAS OnlineDoc.

The portion of the SAS OnlineDoc relevant to the Socket Access Method can be found by selecting -> Base SAS Software -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Statements -> FILENAME, SOCKET Access Method.

The syntax for using the Socket Access Method in a client (reading data) mode is:

FILENAME *fileref* SOCKET '*hostname*:*portno*' <*tcpip-options*>;

As a subsequent example will demonstrate, it is often not necessary to supply the hostname parameter.

Where does this method fit into our toolbox? The URL Access Method is probably simpler to implement. If one merely needs to use a returned web page as program input, then it probably is a good idea to start with this method.

However, if you need to use the communication between a browser or similar web client with a web host, the Socket Access Method is probably the tool to get this done. For example, the Socket Access Method will show the cookie(s) exchanged between a client browser and web host. You can also read the HTTP (Hypertext Transfer Protocol) headers using the Socket Access Method.

The following sample illustrates how a socket is referenced through a filename. The port reference ":80" is the port used by web browsers

```
filename web socket ':80' server
termstr=CRLF ;
```

Notice in the above code that the hostname was not specified. In this example, the host would default to localhost.

When the above example and a related DATA step were issued in an earlier version of both Windows and SAS, one could monitor the URL requests made through a web browser via the SAS Log. Unfortunately, with the current versions of SAS and Windows, this example no longer works due to security features. However, the Socket Access Method is still available in SAS and could be a useful tool in the hands of sophisticated programmers.

CATALOG Access Method

By way of background, a catalog is a special type of file that allows SAS to store different types of information in partitions called catalog entries. Each entry type identifies the purpose of the entry to the SAS system.

Many, if not most, SAS users would find it relatively easy to write a program to read a straight-forward flat text file as program input. However, were one to take the contents of that file and convert it to a SAS catalog source entry, some of those users might hesitate or not be able to complete the task.

However, the principles behind reading a SAS catalog entry are not very different from those

employed in reading flat text files. The Catalog Access Method allows one to read log, output, source and catams SAS catalog entries.

The syntax for the Catalog Access Method is:

FILENAME *fileref* **CATALOG** '*catalog*' <*catalog-options*>;

The portion of the SAS OnlineDoc relevant to the Catalog Access Method can be found by selecting -> Base SAS Software -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Statements -> FILENAME, CATALOG Access Method.

While in some applications, the "catalog" portion of the FILENAME statement can be shortened to the essential three or two parts, in most situations, it is usually a good idea to specify all four parts. Those four parts would be:

library.catalog.entry.type

While the focus of this paper is how to read data into SAS programs, perhaps the easiest way to illustrate how the Catalog Access Method works is to submit the following program within a SAS session.

The FILENAME statement in the program shown in the next column creates the source catalog entry *dummydat* in the *mycat* catalog in the *work* library.

```
filename dummycat catalog
'work.mycat.dummydat.source' ;

data _null_ ;
   file dummycat ;
   put 'here is some sample data' ;
run ;

data stuff ;
   length buffer $ 20 ;
   infile dummycat ;
   input @1 buffer $20. ;
run ;

proc print; run ;
```

The DATA _null_ step that follows the FILENAME statement writes a character string to the *dummycat* fileref. The fileref was opened by the preceding FILENAME statement employing the Catalog Access Method.

The following DATA step creates the SAS data set *stuff* which reads the source catalog entry created and written by the DATA _null_ step. The final PRINT procedure shows that the preceding steps worked as designed.

One might ask, "Why would anyone want to read from a SAS catalog? Would it not be more straight-forward to write and read flat text files?". The first part of the answer would have to be that if one is given data in the form of a SAS catalog, then one has to employ SAS to read the contents into SAS data sets.

But to really answer the question, one ought to ask the additional question, "Why would anyone want to write to a SAS catalog in the first place?". Three reasons could be advanced.

First, some advanced features of SAS software expect, if not require, the use of SAS catalogs to contain program input and output. Second, using catalogs can help make program cross-platform portable. One does not have to worry about which direction the slashes slant, the nature of the directory structure, and the other nuances that differ among operating systems.

However the author's favorite reason to employ SAS catalogs in applications as program input and output is that if you write scratch information to the *work* libref, the scratch area is automatically deleted at the end of the SAS session.

Reading Data Using Named Pipes

Until the author started researching this paper, he had assumed that the ability to use pipes was restricted to the Unix platforms. Much to his surprise, he recently discovered that named pipes may be used under Microsoft Windows NT and Windows 2000 (and now Windows XP).

Named pipes are a technique to allow bi-directional exchange of data among applications. The applications can be on the same computer or among computers connected via a network. Named pipes are documented in SAS OnlineDoc. Look in the *SAS Companion for the Windows Environment*, "Part 2, Using SAS with Other Windows Applications, Using Unnamed and Named Pipes".
The syntax for a Filename statement to set up a named pipe is:

FILENAME *fileref* **NAMEPIPE** '*pipe-specification*' <*named-pipe-options*>;

To demonstrate how named pipes can be implemented using SAS software, consider this example supplied in an earlier version of SAS Online Doc. The computer transmitting the data would run the following program.

```
/* Creates a pipe called WOMEN,  */
/* acting as a server. The       */
/* server waits 30               */
/* seconds for a client to       */
/* connect.                      */

filename women namepipe
  '\\.\pipe\women'
  server retry=30;

/* This code writes three        */
/* records into the named pipe   */
/* called WOMEN.                 */

data class;
   input name $ sex $ age;
   file women;
   if upcase(sex)='F' then
      put name age;
   cards;
MOORE M 15
JOHNSON F 16
DALY F 14
ROBERTS M 14
PARKER F 13
;
```

The receiving computer, which is the same computer in this example, runs the following program.

```
/* Creates a pipe called WOMEN,   */
/* acting as a client.  The       */
/* client waits 30 seconds for a  */
/* server to connect.             */

filename in namepipe
'\\.\pipe\women' client
         retry=30;
data female;
   infile in;
   input name $ age;
proc print;
run;
```

If you have trouble making this example work, try increasing the retry value or try starting the receiving program first. To use this example across a network, replace the period in the Filename statements on the receiving computer with either the network name or IP address of the transmitting computer.

When the author revisited this example using SAS Release 9.1.3 on a Windows XP laptop, the receiving SAS session did not print the results until the transmitting SAS session was terminated.

Sending data from one SAS session to another on the same computer may be viewed as a curiosity. However, when the technique is applied across a network, it can solve real problems, such the lack of SAS/CONNECT software.

Another problem that this technique can solve is how to read data into a second SAS session when only the computer running the first session has the required SAS/ACCESS software. For non-SAS programs that support named pipes, this technique can be used to allow SAS to read data from that program.

A practical example of using unnamed pipes to determine which SAS V9 programs are installed can be found at the following URL:

http://support.sas.com/ctx/samples/index.jsp?sid=1732

Reading Data from Serial Communication Ports

If the ability to read data from FTP and web pages using SAS seems fantastic, consider that SAS can also read data from (and write data to) other computers using an RS-232 serial communication ports. For most personal computer users, that would be the 25 pin or 9 pin port that is used to connect external modems and palm-size personal organizers.

Given the inroads of USB devices, RS-232 serial interfaces are probably more useful to the users of laboratory and service equipment who need a way to automatically record test results. Serial communication ports are also valuable when it is necessary to interface SAS with older computers whose operating system or disk file structure make difficult, if not impossible to obtain the data they contain in any other manner.

The details for reading data from serial communication ports can be found in the SAS OnlineDoc. The relevant section can be found by selecting -> Base SAS Software -> Host Specific Information -> Microsoft Windows Environment -> Getting Started -> Using External Files -> Reading Data from the Communications Port.

In order for serial communication to work, it is necessary match the port setting of the SAS client computer with the other computer with which one

wishes to connect. For a Microsoft Windows computer, the ports are set through the Control Panel.

The serial communications port is associated with SAS using a FILENAME statement. The syntax is:

FILENAME *fileref* **COMMPORT** *"port:"* ;

The following program, taken from the OnlineDoc section cited, reads in data, byte by byte until an end-of-file (hex **'1a'x**) is encountered:

```
data acquire;
   infile test lrecl=1 recfm=f
    unbuffered;
   input i $;
   if i='1a'x then stop;
run;
```

One last tip regarding reading data from serial communication ports is that it is often useful to visually examine the data stream before writing the SAS program to receive and process it.

One way to review the byte stream being sent is to use a terminal program such as Hyper Terminal, supplied with Microsoft Windows. A second way to capture the byte stream for review is to pipe the communication port to a file and then use a hex editor to examine the captured byte stream.

SAS/ACCESS LIBNAME Engines

In the preceding sections, all of the techniques offered involve making data not stored as a flat text file appear to SAS as if it were actually in text file format. If one reviews the syntax used in the previous examples, each program begins with a FILENAME statement.

However, if one takes an expansive view of the title of this paper, one of the most common sources of program input not stored as a flat text file is data stored in a database management system (DBMS). SAS users who can read text files and SAS data sets with confidence sometimes lose their confidence when they must get data from a DBMS.
One of the more useful features introduced with the Nashville release of SAS software (Versions 7 & 8) is the introduction of the LIBNAME access engines. If you recall from the earlier sections, we can specify that a FILENAME statement points to something other than a flat text file by the use of key words such as FTP and URL.

Similarly, we can specify that a LIBNAME statement points to something other than a SAS data set or view by the use of key words such as ODBC or ORACLE. Such a LIBNAME statement tells SAS that the data is already stored as columns and rows and to use specified SAS/ACCESS to make the DBMS appear as if it were a SAS data set.

The general syntax for using the SAS/ACCESS LIBNAME engines is:

LIBNAME *libref SAS/ACCESS-engine-name*
<*SAS/ACCESS-engine-connection-options*>
<*SAS/ACCESS-engine-LIBNAME-options*>;

The details for reading data from DBMSs using the SAS/ACCESS LIBNAME engines can be found in the SAS OnlineDoc. The relevant section can be found by selecting -> Other SAS Software -> SAS/ACCESS Software. Each version of SAS/ACCESS has its own chapter and lists the options applicable to that DBMS.

Some SAS sites do not license any SAS/ACCESS software products. Also, the availability of SAS/ACCESS for specific DBMS products varies by platform. Last, for some of the common DBMSs, a full discussion of the various options could supply enough content for another SAS conference paper. So a "follow-along" example is not provided for this section.

However, to give a flavor of how SAS/ACCESS LIBNAME engines are used, the following example was culled from a program used to access data from Oracle®.

```
libname ora_prod oracle user=userid
  password=password path="@path"
  schema=schema ;
```

Since SAS/ACCESS for Oracle at this site is only licensed and installed on a Windows SAS Server, the preceding code is either submitted from the console of the Windows SAS Server or from a desktop instance of SAS via SAS/CONNECT.

Another variation of this technique is to define a libref to a desktop instance of SAS using a remote engine. A SAS/CONNECT session between the Windows SAS Server and the desktop instance of SAS must be started before submitting a libname statement with the follow syntax.

```
libname ora_prod rengine=oracle
  server=server roptions=
  "user=user password=password
  path='@path' schema=schema" ;
```

One important caveat: If you are using a SAS/ACCESS LIBNAME remote engine to read the schema used for an Enterprise Reporting System (ERP) with hundreds or thousands of tables and you accidentally double-click on the SAS Explorer icon for the libref (ora_prod in this example), you could be in for a lengthy wait for the screen to refresh. This difficulty may have been fixed in later releases of SAS.

The SAS/ACCESS LIBNAME engines offer some attractive features over alternative DBMS access methods. The LIBNAME engine makes all of the tables in the schema instantly available. By contrast, SAS/ACCESS views only define a single table at a time.

When compared to PROC SQL Pass-Thru, the SAS/ACCESS LIBNAME saves one the trouble of learning and using a different dialect of SQL tailored to the DBMS being queried. Further, once the libref has been defined, ordinary DATA step code can be used to access the DBMS data. In most situations, SAS performs the same optimization that previously required the use of pass-thru coding.

The IMPORT, DBF, and DIF procedures can be good tools to convert files from Microsoft Access® and Excel® into SAS data sets. However, the SAS data set from converted data does not change when the source data changes. By contrast, the LIBNAME Engine provides an updated view when the source data changes.

CONCLUSION

The author hopes this survey of SAS tools to read data in formats other than flat text files will increase the reader's appreciation of beauty and flexibility of SAS software. He also hopes that he has helped to demystify the various access methods that have been covered.

One last tip: This paper directs the reader to the SAS OnlineDoc for details about the subjects covered. However, OnlineDoc may not be installed at every site. Fortunately, free access to SAS OnlineDoc is available on the World-Wide Web.

http://support.sas.com/documentation/index.html

BIBLIOGRAPHY

Ward, David L., "You Can Do THAT with SAS Software? Using the socket access method to unite SAS with the Internet," *Proceedings of the 2000 NorthEast SAS User Conference*, 2000. 405-11

ACKNOWLEDGEMENTS

The author would like to thank the Hartford Area SAS User Group Steering Committee, which encouraged him to prepare this paper. Special thanks go to Robert Krajcik, Charles Patridge, and Peter Prause.

Please note that the code supplied in this paper is designed only to illustrate the concepts being discussed and may need to be modified to work in other applications. The author of this paper does not support modified code.

CONTACT INFORMATION

The author may be contacted as follows:

Michael L. Davis
533 Tennis Avenue
Ambler, PA 19002
E-Mail: Michael.Davis@alumni.duke.edu