Using the FTP Access Method

Howard Schreier, U.S. Dept. of Commerce, Washington, DC

ABSTRACT

Recent versions of the SAS® System provide the FTP Access Method, which can be invoked via the FILENAME statement to directly integrate the Internet's File Transfer Protocol with SAS input/output processes. This paper explains and illustrates the use of this feature.

INTRODUCTION

SAS has always provided tools for reading and writing external files (that is, files not structured or managed by the SAS System). Traditionally, this was limited to local files and devices such as keyboards and printers, and the only other software in the picture was the host operating system.

In today's computing environment, SAS applications need a wider variety of services. They have to be able to read and write information through other (non-SAS) applications and be able to access resources over network connections. In some cases, the target has a structure resembling that of a SAS data set, the solution is a conversion "engine", and your SAS program sees the target as if it were a SAS data set. In other cases, the content is more arbitrary and the solution must be more general, so a number of special "access methods" have been added to SAS to take care of such requirements. The FTP Access Method enables SAS applications to read and write files by means of the Internet's File Transfer Protocol (FTP).

QUALIFICATIONS AND CAVEATS

The examples in this tutorial were all run on a Windows 95 system, using SAS Version 6.12 TS020 interactively via the SAS Display Manager System. Behavior and results can also depend on the characteristics and configuration of the FTP server being accessed.

FTP uses bandwidth and server resources. In some cases, techniques which avoid repeated transfers of the same file may be more efficient and more consistent with "good citizenship" on the Internet.

The examples in this paper are not always as robust or generalized as production code would be.

A SIMPLE EXAMPLE

Here is a little program which downloads country codes used with foreign-trade data provided by the Bureau of the Census.

```
filename datadown ftp
 'ctryname.txt'
 host='ftp.census.gov'
 cd='pub/foreign-trade/schedules'
 user='anonymous' pass='Howard_Schreier@'
 debug;

data S_Am;
infile datadown firstobs=6;
input
 name $ 2-34 ccode $ 35-38 isocode $ 45-46;
if ccode=:'3'; * South America;
;
run;
```

Look at the DATA step first. It is very ordinary. In fact, it is exactly as it would be if the input file were on a local device. The invocation of the FTP Access Method is accomplished completely within the preceding FILENAME statement. This FILENAME statement, however, is far from ordinary. It has a distinctive keyword ("ftp") as its device-type parameter and a number of unusual options (which will be examined later).

Let's look at excerpts from the SAS log which is generated when this code is run. In addition to the usual record counts and so forth, there is a lot of detail about the FTP session which takes place as the DATA step executes. The sequence "<<<" identifies messages from the FTP server to the client SAS session and ">>>" identifies messages in the other direction (client to server). The DEBUG option echoes these to the log.

First, the FTP client embedded in the SAS System initializes its connection to the server. In this case, it happens to be an anonymous session.

```
NOTE: <<< 220 blue.census.gov FTP server
(BeroFTPD 1.3.4(1) Thu Mar 4 02:02:45 EST
1999) ready.
NOTE: >>> USER anonymous
NOTE: <<< 331 Guest login ok, send your
complete e-mail address as password.
```

Despite this message, this server accepts a truncated e-mail address. The password is masked in the log, however.

```
NOTE: >>> PASS XXXXXXXXXXXXXXXX
NOTE: <<< 230 Guest login ok, access
restrictions apply.
NOTE: >>> PORT 170,110,71,155,5,42
NOTE: <<< 200 PORT command successful.
```

Next, ASCII type is specified for the transfer. In all of the examples in this paper, SAS automatically made the appropriate choice.

```
NOTE: >>> TYPE A
NOTE: <<< 200 Type set to A.
```

Now SAS navigates to the specified directory on the server and requests retrieval of the file.

```
NOTE: >>> CWD pub/foreign-trade/schedules
NOTE: <<< 250 CWD command successful.
NOTE: >>> RETR ctryname.txt
NOTE: <<< 150 Opening ASCII mode data
connection for ctryname.txt (11060 bytes).
```

In addition to the log notes echoing the conversation with the FTP server, SAS generates the usual notes recording DATA step processing. These also include FTP-specific information; they appear whether or not the DEBUG option is on.

```
NOTE: User anonymous has connected to FTP
server  on Host ftp.census.gov .
NOTE: The infile DATADOWN is:
      Filename=ctryname.txt,
      Pathname=
      "/pub/foreign-trade/schedules"
```

FILENAME *fileref* FTP *'external-file' <ftp-options>* ;	
fileref	is a valid fileref.
FTP	is the access method.
'external-file'	is the name of the file to read from or write to on the remote host machine. Note: If you are not transferring a file but performing a task such as retrieving a directory listing, you do not need to specify a filename. Instead, put empty quotes in the statement.
ftp-options can be any of the following:	
HOST=	'host' either the name of the host (for example, server.pc.sas.com) or the IP address of the machine (for example, 190.96.6.96)
USER=	'username'
PASS=	'password'
PROMPT	specifies to prompt for the user login password
CD=	'directory '
RCMD=	'command ' command to send to the FTP server
LIST	issues the LIST command to the FTP server. This returns the contents of the working directory as records that contain all of the file attributes listed for each file. Note: The file attributes that are returned will vary depending on the FTP server that is being accessed.
DEBUG	writes to the SAS log informational messages that are sent to and received from the FTP server

```
      is current directory,
      Local Host Name=win95-01,
      Local Host IP addr=170.110.71.155,
      Service Hostname Name=ftp.census.gov,
      Service IP addr=148.129.129.31,
      Service Name=FTP,
      Service Portno=21,Lrecl=256,
      Recfm=Variable

NOTE: <<< 226 Transfer complete.
NOTE: >>> QUIT
NOTE: 230 records were read from the infile
DATADOWN.
      The minimum record length was 46.
      The maximum record length was 46.
NOTE: The data set WORK.S_AM has 14
observations and 3 variables.
NOTE: The DATA statement used 20.58 seconds.
```

The DATA step took more than 20 seconds to process a few hundred records. This is attributable to both low throughput (relative to the speed of accessing a local disk) and to the overhead involved in making the client-server connection and setting up the transfer.

This is the SAS data set which results. It will be used in a later example.

```
OBS  NAME              CCODE  ISOCODE

  1  ARGENTINA         3570     AR
  2  BOLIVIA           3350     BO
  3  BRAZIL            3510     BR
  4  CHILE             3370     CL
  5  COLOMBIA          3010     CO
  6  ECUADOR           3310     EC
  7  FALKLAND ISLANDS  3720     FK
  8  FRENCH GUIANA     3170     GF
  9  GUYANA            3120     GY
 10  PARAGUAY          3530     PY
 11  PERU              3330     PE
 12  SURINAME          3150     SR
 13  URUGUAY           3550     UY
 14  VENEZUELA         3070     VE
```

An important thing to understand is that SAS did **not** download the file (ctryname.txt), store it in a temporary location, and feed the DATA step from the copy. Rather, the FTP connection operated while the DATA step executed.

Summary: A specialized form of the FILENAME statement can be used to access files on remote FTP servers.

SYNTAX

The syntax information in the table is from the SAS on-screen Help facility. It is abridged here (a number of specialized and platform-specific options are omitted).

GETTING DIRECTORY INFORMATION

To use the FTP Access Method, you need information (server names, login policies, directory structures, file names, file and record structures). You can do the necessary research using other (non-SAS) FTP client software. But the FTP Access Method can be used for such purposes, and this can be particularly handy when you want to capture such information for purposes of automation.

The following program invokes the LIST option to retrieve a directory listing from the server, then extracts the file names. It does not actually transfer a file (note the null string where a file's name would appear).

```
filename dirlist ftp
 '' list
 host='ftp.census.gov'
 cd='pub/foreign-trade/schedules'
 user='anonymous' pass='Howard_Schreier@'
 debug;

data fnames;
infile dirlist firstobs=2 pad;
input c1 $ 1 @;
if c1='d' then delete; drop c1;
input fname $ 56-100;
if index(compress(lowcase(fname),'o'),'prt');
put _infile_;
server = 'ftp.census.gov';
path   = 'pub/foreign-trade/schedules';
uname  = 'anonymous';
pw     = 'Howard_Schreier@';
run;
```

The FIRSTOBS= option is used to bypass a heading, and the first IF statement filters out pointers to subdirectories (identified by a "d" in the first position). Now suppose we are interested in information pertaining to ports; the second IF statement restricts further processing to file names containing the string "port" or the string "prt". The PUT statement dumps these lines to the log (see box). The last four statements assign the access parameter values to SAS variables; they are strictly for later use and are not necessary to enable FTP access in the present DATA step (after all, these statements are not executed until the FTP connection is already open).

The log shows the change to the subject directory, then the communication of the LIST command.

```
-rw-r--r--   1 2662     5002       265461 Apr 23  1998 fprtcode.txt
-rw-rw-r--   1 2662     5002       264363 Jul 18  1997 fprtcode0.txt
-rw-r--r--   1 2662     5002       265464 Apr 23  1998 fprtctry.txt
-rw-rw-r--   1 2662     5002       264406 Jul 18  1997 fprtctry0.txt
-rw-r--r--   1 2662     5002       265459 Apr 23  1998 fprtname.txt
-rw-rw-r--   1 2662     5002       264407 Jul 18  1997 fprtname0.txt
-rw-rw-r--   1 2662     5002        28351 Oct 16  1995 portcode.txt
-rw-rw-r--   1 2662     5002        26635 Oct 16  1995 portname.txt
```

```
NOTE: >>> CWD pub/foreign-trade/schedules
NOTE: <<< 250 CWD command successful.
NOTE: >>> LIST
NOTE: <<< 150 Opening ASCII mode data
connection for /bin/ls.
NOTE: The infile DIRLIST is:
      Filename,
      Pathname=
      "/pub/foreign-trade/schedules"
      is current directory
```

Summary: Using the LIST option causes SAS to retrieve a directory listing from the server. This listing can be processed as input to a DATA step.

PROCESSING MULTIPLE FILES

Now that we have a list of presumably port-related files, it might be informative to print the first ten lines of each one.

This is pretty simple to do for one file at a time. Here's an example. The coding to access the remote file is very much like that used in the first example.

```
filename dump10 ftp
 'portcode.txt'
 host='ftp.census.gov'
 cd='pub/foreign-trade/schedules'
 user='anonymous' pass='Howard_Schreier@'
 ;

data _null_;
infile dump10 obs=10;
if _n_=1 then put 'portcode.txt';
input;
put _infile_;
run;
```

Here is the result:

```
portcode.txt
 Schedule D -- U.S. Customs Districts and
Port Codes

 PORT     PORT
 CODE     NAME
 ---------------
 0101     PORTLAND, ME
 0102     BANGOR, ME
 0102     BREWER, ME
 0103     CUTLER, ME
 0103     EASTPORT, ME
```

But we actually want to loop through the list of files and dump the first ten records of **each**. The program which was just demonstrated does this for one file, so we could run it over and over again, changing only the specifics. This would get pretty verbose, though a macro could help.

However, there is another technique, one which permits the entire loop to be managed within a single DATA step.

First, we'll see this method in action by reworking the program we just ran (which dumped the first 10 records of portcode.txt).

```
data _null_;
length rec $ 200;
rec = '';
put 'portcode.txt';
rc1 = filename(
 'dump10',
 'portcode.txt',
 'ftp',
 "host='ftp.census.gov'" ||
 " cd='pub/foreign-trade/schedules'" ||
 " user='anonymous' pass='Howard_Schreier@'"
 );
file_id = fopen('dump10');
do i = 1 to 10;
   rc2 = fread(file_id);
   rc3 = fget(file_id,rec,200);
   howlong = length(rec);
   put rec $varying. howlong;
   end;
rc4 = fclose(file_id);
rc5 = filename('dump10');
run;
```

Here is the output written to the log:

```
portcode.txt
 Schedule D -- U.S. Customs Districts and
Port Codes

 PORT     PORT
 CODE     NAME
 ---------------
 0101     PORTLAND, ME
 0102     BANGOR, ME
 0102     BREWER, ME
 0103     CUTLER, ME
 0103     EASTPORT, ME
```

This is exactly the same as the output of the first version of this program. As before, the FTP Access Method was used, and SAS contacted the FTP server to download the file. But here there is no FILENAME statement preceding the DATA step, no INFILE statement and no INPUT statements. Instead, there are calls to several of the file-handling functions which originated in Screen Control Language (SCL, now known as SAS Component Language) and have since been ported to base SAS software.

The FILENAME function does just what the FILENAME statement did in the earlier program. In fact, the function arguments correspond to the statement parameters, except that the order of the file name and the keyword "ftp" are switched and the options are concatenated into one argument. The FOPEN function opens the external source (which in this case is the FTP process), the FREAD function transfers a record from the external source to an intermediate buffer, and the FGET function transfers the contents of that buffer to a character variable. The FCLOSE function closes the external source. There is a second reference to the FILENAME function, to release the reference.

Aside: look at the fourth argument in the first call to the FILENAME function. It's tricky because it's a concatenation of character strings, some of which include quotation marks. Single and double quotes are nested, very carefully. When resolution of macro variables is an issue, even more care is required.

These functions provide an alternative way to do external input-output. In using them, one sacrifices much power and flexibility provided by "traditional" DATA step I-O. The advantage in this case is that **all** of the specifics can be determined at execution time and changed during the course of processing. Nothing has to be hard-coded. In this example, the server name, path and file name happen to be hard-coded, but that is just for illustration. They appear within function arguments, which are **expressions** that can be formed using variables rather than constants. Thus, the I-O can be completely controlled by data values and program logic.

Now we can put this flexibility to work. Here is a generalization of the last program. It is driven by the data set we prepared earlier (so now you see while the FTP parameter values were stored in the data set).

```
data _null_;
set fnames;
length rec $ 200;
rec = '';
put fname;
rc1 = filename(
       'anything',
       trim(fname),
       'ftp',
       'host='  || quote(trim(server)) ||
       ' cd='   || quote(trim(path))   ||
       ' user=' || quote(trim(uname))  ||
       ' pass=' || quote(trim(pw))
       );
file_id = fopen('anything');
do i = 1 to 10;
   rc2 = fread(file_id);
   rc3 = fget(file_id,rec,200);
   howlong = length(rec);
   put rec $varying. howlong;
   end;
rc4 = fclose(file_id);
rc5 = filename('anything');
run;
```

Aside: again, look at the fourth argument in the first call to the FILENAME function. This time, the QUOTE function is used to make the process of embedding quotation marks in the argument a bit less messy.

The important thing in this program is that nothing is hard-coded (except for the fileref "anything", which is in fact arbitrary because its significance is purely internal to this DATA step). It happens that only the value of FNAME varies from observation to observation. The other variables (SERVER, PATH, UNAME, and PW) do not vary; but they could, and this DATA step would work.

Since the SET statement drives the DATA step, an explicit DO loop is coded to traverse the remote files (one of which corresponds to each observation of the data set FNAMES).

Here are excerpts from the log. Ten records are dumped from the first file.

```
fprtcode.txt
Foreign  Foreign                Country
Code     Port Name              Name

01520    Hamilton, ONT          Canada
01527    Clarkson, ONT          Canada
01528    Britt, ONT             Canada
01530    Lakeview, ONT          Canada
01530    Mississauga, ONT       Canada
01530    Port Credit, ONT       Canada
01535    Toronto, ONT           Canada
```

The other files are similarly processed. To save space, we'll not show them all. Finally, the first ten records from the last file appear.

```
portname.txt
 Schedule D -- U.S. Customs Districts and
Port Codes

 PORT                                  PORT
 NAME                                  CODE
 -------------------------------------------
 ABERDEEN-HOQUIAM, WA                  3003
 ADDISON USER FEE AIRPORT, DALLAS, TX  5584
 AGUADILLA, PR                         4901
 AIR CARGO HANDLING SERVICES, INC.     2773
 AIR CARGO HANDLING SERVICES, SF CA    2871
```

So, all of the files were dumped, as intended. It's important to realize, however, that a separate FTP session was opened and closed for each file. This can aggregate into a great deal of overhead.

Summary: SCL-type functions provide an alternative way of implementing I-O. By itself, this has nothing to do with FTP. However, it complements FTP by enabling the DATA step to change files and options at execution time.

UPLOADING

So far, we have only downloaded. But the FTP Access Method is symmetric. Here is an example.

```
filename prestore ftp
 'samerica.sas.ftpdata'
 user='schreih' prompt
 host='ftp.ita.doc.gov'
 cd='/users/ftp/dist/industry/otea'
 debug;
```

Notice that the form of the FILENAME statement does not change. But this one refers to a different server, and to a filespace which does not allow anonymous FTP, so a real user name is supplied and the PROMPT option is coded to avoid embedding a password in stored code. This works nicely in an interactive session, but batch processing would require a different approach to password management.

This trivial program references the remote file.

```
data _null_;
file prestore;
run;
```

I deliberately omitted PUT statements so that no data would actually be transferred. Nevertheless, the FTP session takes place and the file is created. The log shows that the session is initialized.

```
NOTE: <<< 220 infoserv FTP server (Version
1.7.109.12 Fri Jan 31 19:43:51 GMT 1997)
ready.
NOTE: >>> USER schreih
```

```
NOTE: <<< 331 Password required for schreih.
NOTE: >>> PASS XXXXXXXX
NOTE: <<< 230 User schreih logged in.
```

As before, the type and working directory are set.

```
NOTE: >>> TYPE A
NOTE: <<< 200 Type set to A.
NOTE: >>> CWD /users/ftp/dist/industry/otea
NOTE: <<< 250 CWD command successful.
```

Finally, the STOR operation is requested.

```
NOTE: >>> STOR samerica.sas.ftpdata
NOTE: <<< 150 Opening ASCII mode data
connection for samerica.sas.ftpdata.
```

Here are the concluding notes.

```
NOTE: User schreih has connected to FTP
server  on Host infoserv.ita.doc.gov .
NOTE: The file PRESTORE is:
      Filename=samerica.sas.ftpdata,
      Pathname=
      "/users/ftp/dist/industry/otea"
      is current directory

NOTE: 0 records were written to the file
PRESTORE.
```

Summary: The FTP Access Method can upload as well as download.

SEQUENTIAL SAS DATA LIBRARIES

The examples up to this point have used FTP filerefs in FILE and INFILE statements. However, they can be used anywhere a fileref is appropriate (for example, as the source for a %INCLUDE or the target of the PRINTTO procedure).

In addition, the FTP Access Method can (with certain restrictions) be used to read and write SAS data libraries. The primary restriction is that such a library must use a sequential engine, which makes sense since FTP is a sequential process.

To set up an example, I created this data set (BORDERS) identifying pairs of contiguous South American countries:

```
OBS COUNTRY1       COUNTRY2       RECTYPE

  1 Brazil         Argentina       Ref
  2 Brazil         Bolivia         Ref
  3 Brazil         Colombia        Ref
  4 Brazil         French Guiana   Ref
  5 Brazil         Guyana          Ref
  6 Brazil         Paraguay        Ref
  7 Brazil         Peru            Ref
  8 Brazil         Suriname        Ref
  9 Brazil         Uruguay         Ref
 10 Brazil         Venezuela       Ref
 11 Chile          Argentina       Ref
 12 Ecuador        Colombia        Ref
 13 Ecuador        Peru            Ref
 14 French Guiana  Suriname        Ref
 15 Suriname       Guyana          Ref
 16 Guyana         Venezuela       Ref
 17 Venezuela      Colombia        Ref
 18 Colombia       Peru            Ref
 19 Peru           Bolivia         Ref
 20 Peru           Chile           Ref
 21 Chile          Bolivia         Ref
 22 Argentina      Bolivia         Ref
 23 Argentina      Paraguay        Ref
 24 Argentina      Uruguay         Ref
 25 Bolivia        Paraguay        Ref
```

Then I ran this little program to make the permutations explicit, generating a pair of observations for each original observation.

```
data link;
set borders (rename=
     (country1=country country2=neighbor))
    borders(rename=
     (country2=country country1=neighbor));
run;

proc sort data=link;
by country neighbor;
run;
```

Now we're going to copy these two SAS data sets (BORDERS, LINKS) into a SAS data library in transport format on the FTP server. Here is the program.

```
filename onserver ftp
 'samerica.sas.ftpdata'
 user='schreih' prompt
 host='ftp.ita.doc.gov'
 cd='/users/ftp/dist/industry/otea'
 rcmd='SITE chmod 664 samerica.sas.ftpdata'
 debug;

libname onserver xport;

proc datasets;
copy out=onserver;
select borders link;
quit;
```

For the moment, ignore the RCMD= option; it's incidental. The thing to notice is that the FILENAME and LIBNAME share a reference name ("onserver"). The log shows that this pairing is recognized, since the libref specifics come from a combination (engine from the LIBNAME statement and file name from the FILENAME statement).

```
NOTE: Libref ONSERVER was successfully
assigned as follows:
      Engine:        XPORT
      Physical Name: samerica.sas.ftpdata
```

Looking at additional log excerpts, we see that processing of the first SAS data set (BORDERS) triggers initialization of the FTP connection:

```
NOTE: Copying WORK.BORDERS to
ONSERVER.BORDERS (MEMTYPE=DATA).
NOTE: >>> TYPE I
NOTE: <<< 200 Type set to I.
```

Notice that SAS has automatically requested image ("I") type. If it hadn't, we could expect the output (a SAS data library in transport format) to become corrupted.

```
NOTE: >>> CWD /users/ftp/dist/industry/otea
NOTE: <<< 250 CWD command successful.
NOTE: >>> SITE chmod 664 samerica.sas.ftpdata
NOTE: <<< 200 CHMOD command successful.
```

At this point, SAS has passed a Unix command to the server, as specified in the RCMD= option in the FILENAME statement. This is just to do some housekeeping to set up one of the later examples. Notice that this remote command references the very file that is being written in this FTP session. SAS passes the command specified in the RCMD= option **before** initiating the "main event" (uploading the file), so this would trigger an error condition except for the fact that we created an empty file with this name in the previous example.

```
NOTE: >>> STAT
NOTE: <<<      TYPE: Image; STRUcture: File;
transfer MODE: Stream
NOTE: <<< 211 End of status
NOTE: >>> STOR samerica.sas.ftpdata
NOTE: <<< 150 Opening BINARY mode data
connection for samerica.sas.ftpdata.
```

Notice that as a consequence of the "image" type value, the mode is binary, which preserves the integrity of the SAS data library.

```
NOTE: The data set ONSERVER.BORDERS has 25
observations and 3 variables.
```

Finally, processing of the first data set is complete. PROC DATASETS continues and processes the second data set.

```
NOTE: Copying WORK.LINK to ONSERVER.LINK
(MEMTYPE=DATA).
NOTE: The data set ONSERVER.LINK has 50
observations and 3 variables.
```

PUBLIC ACCESS

We have created a SAS data library in transport library on the FTP server. This was a bit tricky, so we should confirm its readability.

I deliberately stored the library is in a filespace which permits anonymous FTP (which, by the way, was the reason for the Unix "chmod" in the previous example).

Aside: This anonymous FTP server accepts null passwords, whereas one used earlier requires the left portion of the user's e-mail address; others require complete e-mail addresses. This is a minor illustration of an important point: that FTP servers vary in functionality, configuration, and policy.

As before, we'll use a matched FILENAME-LIBNAME pair.

```
filename anyall ftp
 'samerica.sas.ftpdata'
 user='anonymous' pass=''
 host='ftp.ita.doc.gov'
 cd='dist/industry/otea'
 debug;

libname anyall xport;
```

Next comes a DATA step which invokes the libref in a SET statement.

```
data; set anyall.borders;
run;
```

Looking at the log, we see the expected confirmation of the libref.

```
NOTE: Libref ANYALL was successfully assigned
as follows:
      Engine:        XPORT
      Physical Name: samerica.sas.ftpdata
```

First, the session is initialized.

```
NOTE: >>> USER anonymous
NOTE: <<< 331 Guest login ok, send ident as
password.
NOTE: >>> PASS
NOTE: <<< 230 Guest login ok, access
restrictions apply.
```

As before, SAS automatically opts for image ("I") type. It should be possible to override these "type" determinations via the RCMD= option, but I did not find it necessary for any of the examples in this paper.

```
NOTE: >>> TYPE I
NOTE: <<< 200 Type set to I.
NOTE: >>> CWD dist/industry/otea
NOTE: <<< 250 CWD command successful.
NOTE: >>> RETR samerica.sas.ftpdata
NOTE: <<< 150 Opening BINARY mode data
connection for samerica.sas.ftpdata (6240
bytes).

NOTE: The data set WORK.DATA1 has 25
observations and 3 variables.
```

So we have closed the circle by bringing one of the SAS data sets back to the SAS session on the local machine.

Consider: (1) the SAS data library is in a filespace which permits anonymous FTP; (2) the library is in transport format, which is by definition cross-platform; and (3) the FTP Access Method is part of base SAS software. The implication is that this last program (five statements: FILENAME, LIBNAME, DATA, SET, and RUN) is universal, and should execute in any SAS session anywhere, as long as the computer is connected to the Internet and the version of SAS in use supports the FTP Access Method (and no local security measures intefere). Thus, this arrangement allows one to set up "public libraries" of SAS data.

Summary: The FTP Access Method can be used in contexts other than FILE and INFILE. SAS data libraries in transport format can be written and read.

A WEBSITE MAINTENANCE EXAMPLE

Quite commonly, FTP uploads are performed to add or refresh Web content. If the preparation of the content is data-driven, SAS can be a useful tool. Then the FTP Access Method makes it possible to update and upload in one step.

To set up the example, let's put the country codes into a format.

```
data isocode;
set S_Am;
fmtname = 'isocode';
type = 'C';
start = name;
label = isocode;
run;

proc format cntlin=isocode; run;
```

I previously loaded some U.S. export data, covering all mainland South American countries for three years. This is a subset.

```
COUNTRY    YEAR  Q1   Q2   Q3   Q4 RECTYPE

Argentina  1996  970 1158 1181 1207  Data
Argentina  1997 1233 1390 1479 1705  Data
Argentina  1998 1404 1567 1510 1404  Data
Bolivia    1996   48   49   65  107  Data
. . .
Peru       1996  407  458  477  425  Data
Peru       1997  449  454  514  543  Data
Peru       1998  532  508  470  546  Data
. . .
Uruguay    1998  130  150  152  159  Data
Venezuela  1996 1072 1233 1204 1232  Data
Venezuela  1997 1306 1651 1849 1802  Data
Venezuela  1998 1890 1781 1439 1410  Data
```

The task at hand is to place on the Web a set of cross-referenced tables based on these figures, and using the list of borders to link them. In other words, each country's data will be on a separate page, with links to the pages of neighboring countries.

Each upload of a file generates a separate FTP session; there are 13 countries in the data set, and we do not want to be prompted 13 times for the password. This macro prompts once for the password, and places it into a macro variable.

```
%macro loadpw;
   data _null_;
   window install rows=12 columns=60
color=blue
    #1 'Type password:' color = yellow
    #3 password $30. attr =rev_video
    #6 'ENTER to confirm' color=yellow
     attr=rev_video;
   display install;
   call symput('pw',password);
   stop;
   run;
   %mend loadpw;
```

This macro should be run before the code which does the FTP. Then, after the uploads have been completed, the single statement

```
%let pw=;
```

will clear the password from macrovariable space. Once again, keep in mind that processing in batch would require a different strategy for password management.

Here is the outline for the report-generator, in the form of pseudocode:

```
interleave data and references, by COUNTRY
if starting COUNTRY (
   open output file
   generate page headings
   )
if data record (
   if first (
      start table
      generate column headings
      )
   generate table row
   if last finish table
   )
if reference record generate reference link
if ending COUNTRY (
   finish page
   close file
   )
```

The idea is to write directly to the Web server. The FILE statement's FILEVAR= option would be a handy way to generate all of the individual files in one DATA step. But with the FTP Access Method, the FILEVAR= option cannot be used (during execution, file names are recognized, but the output is sent to files so named in the local default directory, ignoring the specifications coded in the FILENAME statement). Moreover, FILEVAR= only alters the file **name** at execution time; it would not permit changing remote directory and other parameters coded as **options** in the FILENAME statement.

So we'll again use the SCL-type functions to manage FTP connections at execution time. The appendix presents the DATA step. It's pretty dense-looking, because it intersperses HTML tags with the information content. Moreover, lines of HTML have to be assembled before being passed to the output buffer. The result is extensive use of character operators and functions.

As expected, each COUNTRY value initiates its own FTP session. We can look at log excerpts.

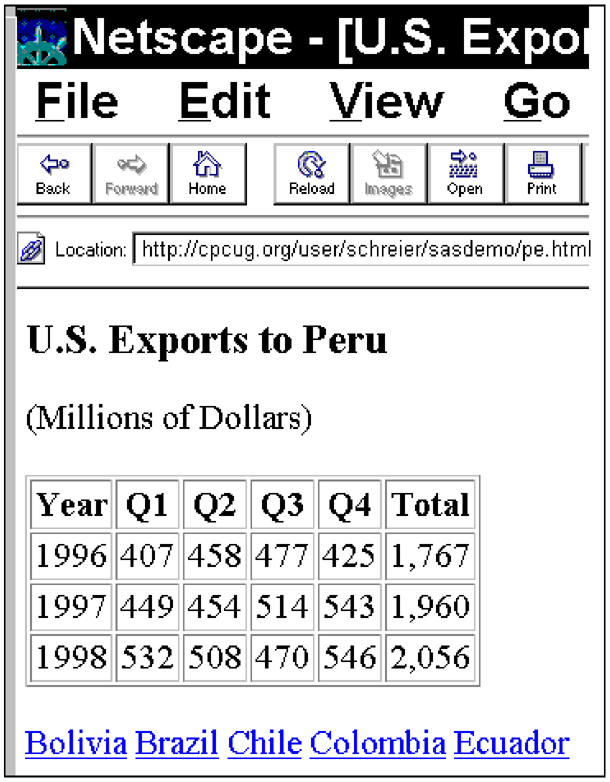
```
NOTE: 220 cpcug.org FTP server (Version 5.60)
ready.
NOTE: User schreier has connected to FTP
server  on Host cpcug.org .
NOTE: 220 cpcug.org FTP server (Version 5.60)
ready.
NOTE: User schreier has connected to FTP
server  on Host cpcug.org .
```

There are 11 more pairs of notes, 13 pairs in all, reflecting the separate FTP sessions for the 13 country files.

```
NOTE: The DATA statement used 6 minutes 56.06
seconds.
```

Note the long elapsed time. Considering the tiny volume of data, we can only conclude that overhead is to blame.

The screen shot shows the page for Peru. The links at the bottom are based on Peru's borders. The HTML for this page is in the Appendix.



http://cpcug.org/user/schreier/sasdemo/pe.html

U.S. Exports to Peru

(Millions of Dollars)

Year	Q1	Q2	Q3	Q4	Total
1996	407	458	477	425	1,767
1997	449	454	514	543	1,960
1998	532	508	470	546	2,056

Bolivia Brazil Chile Colombia Ecuador

Summary: The FTP Access Method can be used to implement one-step Web maintenance procedures which include data-determined links.

CONCLUSION

The FTP Access Method allows SAS programs to read and write remote files at execution time. However, there are significant performance and efficiency tradeoffs to be considered. Also, using it effectively requires some grasp of the target server's capabilities and configuration, and of the file management principles of the server's host operating system (often Unix). Finally, password management can be a source of difficulty. Nevertheless, the FTP Access Method can be a useful tool.

REFERENCES

DeAngelis, Roger (8 Oct 1996), “utlftpg Remote access to SAS transport datasets” (posting to SAS-L, archived as gopher://jse.stat.ncsu.edu:70/0R1554780-1560591-1m/othergroups/sasl/log9610)

SAS Institute Inc. (1990), *SAS Screen Control Language, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1996), *The SAS System for Windows, Release 6.12, TS Level 0020, Windows Version 4.0.950* (On-screen Help), Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact me:

Howard Schreier
Mail Stop H-2815
U.S. Dept. of Commerce
Washington DC 20230

(202) 482-4180

Howard_Schreier@ita.doc.gov

APPENDIX

A. WEB MAINTENANCE EXAMPLE: SAMPLE OUTPUT (PERU)

```
<HTML><HEAD>
<TITLE>U.S. Exports to Peru</TITLE>
</HEAD><BODY>
<H3>U.S. Exports to Peru</H3>
(Millions of Dollars)<P>
<TABLE BORDER CELLPADDING="2">
<TR>
<TD align=center><B>Year</B></TD>
<TD align=center><B>Q1</B></TD>
<TD align=center><B>Q2</B></TD>
<TD align=center><B>Q3</B></TD>
<TD align=center><B>Q4</B></TD>
<TD align=center><B>Total</B></TD>
</TR>
<TR>
<TD>        1996</TD>
<TD align=right>    407</TD>
<TD align=right>    458</TD>
<TD align=right>    477</TD>
<TD align=right>    425</TD>
<TD align=right>  1,767</TD>
</TR>
<TR>
<TD>        1997</TD>
<TD align=right>    449</TD>
<TD align=right>    454</TD>
<TD align=right>    514</TD>
<TD align=right>    543</TD>
<TD align=right>  1,960</TD>
</TR>
<TR>
<TD>        1998</TD>
<TD align=right>    532</TD>
<TD align=right>    508</TD>
<TD align=right>    470</TD>
<TD align=right>    546</TD>
<TD align=right>  2,056</TD>
</TR>
</TABLE><P>
<A HREF="bo.html">Bolivia</A>
<A HREF="br.html">Brazil</A>
<A HREF="cl.html">Chile</A>
<A HREF="co.html">Colombia</A>
<A HREF="ec.html">Ecuador</A>
</BODY></HTML>
```

B. SAS CODE FOR WEB MAINTENANCE EXAMPLE

```
data _null_;
length line $ 200;
* Generate one Web page per country;
set expsamer(in=indata)
    link(in=inlink);
by country rectype;
if first.country then do;
   * Open file and generate HTML
     preceding the data;
   rcode1  = filename(
    'webstore',
    lowcase(put(upcase(country),$isocode.))
     || '.html',
    'ftp',
    "host='cpcug.org '" ||
     "cd='/homed/web/httpdocs" ||
      "/user/schreier/sasdemo'" ||
     "user='schreier' pass='&PW'"
    );
   file_id = fopen('webstore','O');
   retain file_id;
   line = '<HTML><HEAD>';
   link line_out;
   line = '<TITLE>U.S. Exports to ' ||
          trim(country) || '</TITLE>';
   link line_out;
   line = '</HEAD><BODY>';
   link line_out;
   line = '<H3>U.S. Exports to ' ||
          trim(country) || '</H3>';
   link line_out;
   line = '(Millions of Dollars)<P>';
   link line_out;
   end;
if indata then do;
   * HTML presenting the numeric data;
   if first.rectype then do;
      * Column headings;
      line =
       '<TABLE BORDER CELLPADDING="2">';
      link line_out;
      line =  '<TR>';
      link line_out;
      line =
       '<TD align=center><B>Year</B></TD>';
      link line_out;
      line =
       '<TD align=center><B>Q1</B></TD>';
      link line_out;
      line =
       '<TD align=center><B>Q2</B></TD>';
      link line_out;
      line =
       '<TD align=center><B>Q3</B></TD>';
      link line_out;
      line =
       '<TD align=center><B>Q4</B></TD>';
      link line_out;
      line =
       '<TD align=center><B>Total</B></TD>';
      link line_out;
      line = '</TR>';
      link line_out;
      end;
   * Data for one year;
   line =  '<TR>';
   link line_out;
   line =  '<TD>' || year || '</TD>';
   link line_out;
   line = '<TD align=right>' ||
          put(q1,comma7.) || '</TD>';
   link line_out;
   line = '<TD align=right>' ||
          put(q2,comma7.) || '</TD>';
   link line_out;
   line = '<TD align=right>' ||
          put(q3,comma7.) || '</TD>';
   link line_out;
   line = '<TD align=right>' ||
          put(q4,comma7.) || '</TD>';
   link line_out;
   line = '<TD align=right>' ||
          put(sum (of q1-q4),comma7.) ||
          '</TD>';
   link line_out;
   line =  '</TR>';
   link line_out;
   if last.rectype then do;
      line =  '</TABLE><P>';
      link line_out;
      end;
   end;
if inlink then do;
   * Link to page for one bordering country;
   line = '<A HREF="' ||
          lowcase(put(upcase(neighbor),
          $isocode.)) ||
          '.html">' ||
          tranwrd
           (trim(neighbor),' ','&nbsp;')
          ||'</A> ';
   link line_out;
   end;
if last.country then do;
   * Generate HTML following the data,
     then close file;
   line =  '</BODY></HTML>';
   link line_out;
   rcode4  = fclose(file_id);
   rcode5  = filename('webstore','');
   end;
return;
line_out:
   rcode2 = fput(file_id ,line);
   rcode3 = fwrite(file_id);
   return;

run;
```