Paper GH-06

Let SAS® Do the Downloading: Using Macros to Generate FTP Script Files

Arthur Furnia, Federal Aviation Administration, Washington DC

ABSTRACT

Analysts often need to obtain data generated by an office outside of their organization. Ideally, the analyst would have access to the database of that office, but sometimes security rules require that the data must be obtained in some intermediate format to be processed later. Downloading this intermediate data can be labor-intensive, and not only wasteful of time but also prone to human error resulting in bad or incomplete data downloads. Imagine changing a few macro parameters in a SAS program, that when executed creates its own code to download all required files, waiting for one set of files to download before writing more code to download the next set of files, until it reaches a pre-determined point at which processing stops. This paper shows an example of how to use a series of macros to construct an individual script file containing FTP commands that when executed downloads a specific group of files for a given month from a specific location on the FTP site to a specific directory on a local computer or network. As many script files are created as there are directories requiring data. After each script file is run, it is then deleted. Since only a few parameters at the beginning of the program need to be changed each month, even non-SAS users can easily run the program. Therefore an analyst can run a SAS program overnight, and spend no more time painstakingly downloading data from an FTP site.

INTRODUCTION

The purpose of this paper is to explore the use of certain SAS external file functions to construct a file residing in a folder on a network. This paper focuses on just five specific functions – FILENAME, FOPEN, FPUT, FAPPEND, and FCLOSE – which are all the functions that are needed to construct an external file. The particular file being created contains File Transfer Protocol (FTP) commands and so is called an FTP script file. When the FTP script file is executed (by the appropriate DOS command line entry) each successive FTP command contained therein is sequentially executed, resulting in the copying of a group of files from an FTP site. There are over 40 FTP commands but the script file discussed in this paper uses just a few of them. For a broader discussion the reader is referred to a more general reference on FTP.[1]

THE PROBLEM

The topic of this paper originated in the solution to a recurring task in which a large number of comma separated value (.CSV) files had to be copied once a month from folders on an FTP site to folders on a Local Area Network (LAN) drive. The .CSV files contain flight operations history data by facility, and the file size is proportional to the total operations at the facility. One of the easiest ways of grabbing data from an FTP site is to highlight the file(s) and either drag it (them) from the FTP site to a LAN folder or click on “Copy to Folder” in Windows, navigate to and highlight the folder to copy to, and click on the “Copy” button. Though the actual copying speed was satisfactory, the time to copy a month of files varied anywhere from a few minutes for a small facility to about 20 minutes for a large facility, so it was hard to concentrate on another task while waiting for one facility’s files to finish copying. In addition, a common error was to either miss one or more files for a given facility by failing to highlight them, or to accidentally drag them to the wrong folder on the network drive.

THE SOLUTION

If the files to be copied were somewhere else on the LAN, one could create a DOS batch file (.BAT) which employed the COPY and CD commands with wildcard characters. Although this would involve creating a large batch file upfront, it could be easily updated each month by running some global replace commands. The solution would not be quite that easy, since the files to be copied were on an FTP site. However, one can start an FTP client session and execute FTP client commands entered as sequential lines in an ascii script file, similar to a DOS batch file.

BASIC FTP COMMANDS

There are a number of basic FTP client commands that essentially mimic DOS-based commands. These commands include the following:

cd - change remote (i.e. FTP site) working directory
lcd – change local (i.e. network drive) working directory
get – receive file (from FTP site)
mget – get multiple files (from FTP site)

quit – terminate FTP session and exit

Two additional FTP commands involved in copying files are:

binary – set binary transfer type
ascii – set ascii transfer type

However, since the files we are copying are in CSV format, either binary or ascii transfer types will work, and so neither of these commands need to be entered.

RUNNING AN FTP SCRIPT FILE

All commands in an FTP script file can be executed using the following command-line syntax:

FTP -i -s:<path>\filename ftp-site

Keywords/Arguments/Options:

FTP
invokes command line FTP client

-i
turns off interactive prompting during multiple file transfers. (This is important since we will be copying as many files as there are days in a given month.)

-s:<path>filename
opens the script file *filename,* which will then automatically run. *Filename* may reside in the current directory, or it may reside in another directory folder, which would require the folder *path* to precede *filename*. (See discussion of *path* below.)

path
the directory path where the script file is located. For example, on a network in which the root drive is "S", the path might be "S:\folder1\folder2\...\foldern\"

filename
the ascii script file containing the FTP commands to be executed

ftp-site
this can be either the name of the FTP site (e.g. my.ftp.site) or the IP address of the FTP site (e.g. 12.345.678.9)

THE BASIC FTP SCRIPT FILE

To access a secure FTP site, the FTP script file must start with two lines, containing the required user ID and password, respectively. The next lines should contain those FTP commands required to copy the required files from a specific FTP site folder to a specific folder on the network drive. A very basic script file to copy just one file would have the following structure:

user ID
password
cd foldername
<binary/ascii>
<lcd command(s)>
mget filename
quit

The ***cd*** command is used to navigate to the folder on the FTP site where the requested file resides. There will only be one cd command required since each facility-specific folder is one level below the root directory on the FTP site.

Binary simply states that the file to be transferred is binary; ***ascii*** is the other option. It is shown here in brackets since either binary or ascii can be used when copying a comma-separated value file.

The ***lcd*** commands are optional and depend on where the script file is being executed from. If the script file is not located in the directory where the target file is to be copied to, then the lcd commands can be used to change the local directory so that the file to be copied ends up in the folder indicated by the final lcd command. It is shown here in brackets because the first command in the SIGNON macro (to be discussed below) copies the original "bare bones" script file (with only rows for user ID and password) in to the correct LAN folder, i.e. the folder corresponding to the facility whose .CSV files we are copying (from the FTP site). The final modified script will be run from that folder. And so the lcd commands should not be necessary, though they can be used to verify that the files are being copied to the correct facility folder on the LAN.

The ***Get*** command literally "gets" the file and copies it to the local folder. However, since we will be copying an entire month's worth of files each time a new script file is run, we will use the ***mget*** command to "get" each file and copy to the local folder. In the final modified script file for a given facility, there will be as many "mget" statements as there are days in the month.

Quit returns control to the DOS command prompt.

OVERVIEW OF MACROS TO BE USED

The FTP commands will be added to the script file by using SAS macros. To process a given facility's data for each day in a month we will use one overall macro to call a series of submacros, each of which when called will add one or more lines to the script file being created. The overall macro is called PROCESSM, and takes the parameter LOC, a macro variable representing the 3-character facility ID. Each submacro called in PROCESSM uses LOC as a parameter. PROCESSM appears as follows:

```
%MACRO PROCESSM(LOC);
%put _local_;
    %SIGNON(&LOC);
    %ADDLOC(&LOC);
    %ADDBIN(&LOC);
    %VERIFYDIR(&LOC);
    %ADDMONTH(&LOC,&MONTHNUM,&YEAR);
    %ADDQUIT(&LOC);
    %FTPCSV(&LOC);
    %DELSIGN(&LOC);
%let LOC=;
%MEND PROCESSM;
```

The SIGNON macro copies the bare bones script file that contains two lines: one with the required user ID and one with the password. The user ID and password must appear on lines one and two respectively so that we can access the FTP site.

The ADDLOC macro navigates to the subfolder on the FTP site corresponding to the current LOC (i.e. facility) being processed.

Recall from our earlier discussion (see **THE BASIC FTP SCRIPT FILE** above) that since we are copying a comma-separated value file, we don't need to declare the file to be copied as either binary or ascii, which is the purpose of the ADDBIN submacro. Also, the VERIFYDIR macro adds FTP commands that change the LAN drive folder to the one corresponding to the current facility being processed, but since the first submacro (SIGNON) copies the "bare bones" script file (to be appended by subsequent FTP commands) to the current facility's folder, the VERIFYDIR macro is also unnecessary. Nevertheless, we will discuss both ADDBIN and VERIFYDIR below.

The ADDMONTH macro will execute a nested macro once for each day in the given month. The nested macro, to be discussed below, will copy the .CSV file corresponding to a particular day from the FTP site.

The ADDQUIT macro adds the final line to the script file.

While the previous macros contained in PROCESSM, dealt with building the script file, the FTPCSV macro actually runs the final edited script file.

Finally, the DELSIGN macro performs the housekeeping role of deleting the just-run SIGNON script file from the network folder corresponding to the current facility, so that when we run the program next month, a clean copy (with just user ID and password) of SIGNON can be copied in to the folder for editing.

USING SAS EXTERNAL FILE FUNCTIONS TO MODIFY A FILE

Let's look at the SAS functions we will be using. All of the SAS functions we will be using are called external file functions, which is appropriate since we are using SAS to construct an external FTP script file.

FILENAME FUNCTION

The FILENAME function assigns or de-assigns a fileref to an external file, directory, or output device. It is needed here because the other SAS external file functions require that files be identified by fileref rather than by physical filename.

Syntax: FILENAME(fileref <,file-name> <,device-type> <,'host-options'> <,dir-ref>)

fileref
is a character constant, variable, or expression that specifies the fileref assigned to the external file.

device-type
is a character constant, variable, or expression that specifies the type of device or the access method that is used if the fileref points to an input or output device or location that is not a physical file

'host-options'
specifies host-specific details such as file attributes and processing attributes.

dir-ref
specifies the fileref that was assigned to the directory or partitioned data set in which the external file resides.

FOPEN FUNCTION

FOPEN opens an external file and returns a file identifier value.

Syntax: FOPEN(fileref<,open-mode<,record-length<,record-format>>>)

fileref
is a character constant, variable, or expression that specifies the fileref assigned to the external file.

open-mode
is a character constant, variable, or expression that specifies the type of access to the file:
A - APPEND mode allows writing new records after the current end of the file.
I - INPUT mode allows reading only (default).
O - OUTPUT mode defaults to the OPEN mode specified in the operating environment (i.e. device type) option in the FILENAME statement or function. If no operating environment option is specified, it allows writing new records at the beginning of the file.
S – SEQUENTIAL INPUT mode is used for pipes and other sequential devices such as hardware ports.
U - UPDATE mode allows both reading and writing.

record-length
is a numeric constant, variable, or expression that specifies the logical record length of the file.

record-format
is a character constant, variable, or expression that specifies the record format of the file. To use the existing record format, do not specify a value here.

FOPEN is required along with the argument A (indicating APPEND), since each macro is appending something to the script file to which FILENAME assigns a fileref.

FPUT FUNCTION
FPUT moves data to the File Data Buffer (FDB) of an external file, starting at the FDB's current column position.

Syntax: FPUT(file-id,cval)

file-id
is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

cval
is a character constant, variable, or expression that specifies the data to be "put".

FAPPEND FUNCTION
FAPPEND appends the current record to the end of an external file

Syntax: FAPPEND(file-id<,cc>)

file-id
is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

cc
is a character constant, variable, or expression that specifies a carriage-control character:
0 skips one blank line before this new line.
- skips two blank lines before this new line.
1 specifies that the line starts a new page.
+ specifies that the line overstrikes a previous line.
P specifies that the line is a computer prompt.
= specifies that the line contains carriage control information.
all else (including a blank) specifies that the line record starts a new line.

FCLOSE FUNCTION
FCLOSE closes an external file, directory, or directory member.

Syntax: FCLOSE(file-id)

file-id
is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

RETURN CODE AND FILENAME FUNCTIONS
The rc, or return code, is often assigned as a macro variable when using a SAS external file function. Technically, the return code gets assigned a value, but as long as we know that the given external file is correctly identified, we can assume the rc's value will be 0, and the fileref, henceforth referred to by its macro variable name FILRF, is correctly assigned. The rc is used with the FILENAME function as follows:

```
%let rc=%sysfunc(filename(filrf,<external file name>));
```

"rc" is the return code passed by a user function. The value of a SAS system return code can be interpreted as:
0: The operation was completed successfully.
>0: An error condition was detected during the operation.
<0: The operation was completed, but a warning or a note was generated.

It should be noted that in a macro statement character strings, such as the external file name used here, are not enclosed in quotation marks. However, we tested this code with and without enclosing the file name in quotes, and both ways worked. Once FILENAME has been used to assign the FILRF, the FOPEN function is used to open the external file, as specified by &FILRF, using the following line:

```
%let fid=%sysfunc(fopen(&filrf,a));
```

with the modifier in the second argument “a” indicating that the opened file shall be appended to. Macro variable FID is now the file identifier.

The FPUT function will be the external file function that varies the most between the individual submacro’s discussed below. The rc is used with the FPUT function as follows:

```
%let rc=%sysfunc(fput(&fid, cval));
```

where cval is a character constant, variable, or expression that specifies the data to be appended to the external file.

Finally, the rc is used with the FAPPEND and FCLOSE functions as follows:

```
%let rc=%sysfunc(fappend(&fid));

%let rc=%sysfunc(fclose(&fid));
```

AND NOW…THE SUBMACROS

COPY THE BASIC (UNEDITED) SCRIPT FILE

```
%MACRO SIGNON(LOC);
X copy c:\SIGNON P:\folder1\folder2\folder3\folder4\&LOC.\signon;
%MEND SIGNON;
```

The macro SIGNON uses the SAS X command, which is used to issue an operating environment command from within a SAS session. A DOS copy command copies the generic SIGNON script file, located on the root of the C drive to a subfolder (5 levels down) on the network drive “P”, corresponding to the value of the macro variable LOC, which designates the facility ID.

NAVIGATE TO CORRECT FOLDER ON FTP SITE

```
%MACRO ADDLOC(LOC);
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,"P:\folder1\folder2\folder3\folder4\&LOC.\signon"));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fput(&fid, cd &LOC.));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;  /* unsuccessful open processing */  %end;
%MEND ADDLOC;
```

The macro ADDLOC opens the generic SIGNON script (which the previous submacro copied to correct network folder) and appends the line – “cd &LOC” (where &LOC is evaluated as the 3-character facility ID) - to the two lines (for user ID and password) that are already in the SIGNON script. Recall that the CD command changes the remote directory (i.e. on the FTP site). For example, if we are running the facility PHL, then the line “cd PHL”, after &LOC resolves to PHL, will be tacked on to the end of the SIGNON file.

SET DOWNLOAD FILE TYPE (OPTIONAL)

```
%MACRO ADDBIN(LOC);
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,"P:\folder1\folder2\folder3\folder4\&LOC.\signon"));
%let fid=%sysfunc(fopen(&filrf,a));
```

```
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fput(&fid,binary));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;       /* unsuccessful open processing */   %end;
%MEND ADDBIN;
```

Macro ADDBIN sets the download file type by opening the generic SIGNON script and appending the line – "binary" - to the three lines (for user ID, password, and cd &LOC) that are already in the SIGNON script. Since we are actually using the SAS macros to download a comma-separated value file, one could instead add the line "ascii" instead of "binary", and so inclusion of this submacro was optional.

NAVIGATE TO CORRECT FOLDER ON NETWORK DRIVE (OPTIONAL)

```
%MACRO VERIFYDIR(LOC);
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,"P:\folder1\folder2\folder3\folder4\&LOC.\signon"));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fput(&fid,lcd ..));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fput(&fid,lcd &LOC.));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else %do;   /* unsuccessful open processing */   %end;

%MEND VERIFYDIR;
```

Macro VERIFYDIR opens the generic SIGNON script and appends two lines – "lcd .." and "lcd &LOC." - to the four lines (for user ID, password, cd LOC, and binary) that are already in the SIGNON script. These two lines essentially verify that the subdirectory where the CSV files are to be copied to is indeed the one corresponding to &LOC. Note that both these lines are appended to the script file within the main %do loop – the first FPUT/ FAPPEND lines move the active DOS directory up from the facility level folder (corresponding to the last facility processed) to the higher level folder4, and then the second FPUT/FAPPEND lines move the active directory down one level to the facility level folder corresponding to the new facility being processed.

COPY ONE FILE FROM FTP SITE TO FOLDER ON NETWORK DRIVE (NESTED MACRO ADDMGET)

```
%MACRO ADDMGET(LOC,YEAR,MONTHNUM,DAY);
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
"P:\folder1\folder2\folder3\folder4\&LOC.\signon"));
%let fid=%sysfunc(fopen(&filrf,a));  *a means APPEND mode;
%if &fid > 0 %then
   %do;
  %let rc=%sysfunc(fput(&fid,
     mget &LOC._Ops_History_&YEAR.-&MONTHNUM.-&DAY._Lcl.csv));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;       /* unsuccessful open processing */     %end;
%MEND ADDMGET;
```

Macro ADDMGET opens the generic SIGNON script and appends the line - mget &LOC_Ops_History_YYYY-MM-DD_Lcl.csv - to the lines that are already in the SIGNON script. Recall the FTP command MGET copies one file (in a series of files) from the remote location. The ADDMGET macro is not directly run by the PROCESSM macro, which is serving as the conductor, running all the submacros. Rather, since ADDMGET is just copying one day of data, and the whole point of the program is to copy a month of data for each facility, we will need to run ADDMGET as many times as there are days in the month. However, since some months have 30 days and some months have 31 days (and one has 28 or sometimes 29 days), SAS needs to know how many times to run ADDMGET. This leads us to our next macro, ENDMONTH.

GET THE LAST DAY OF THE CURRENT MONTH

```
%MACRO ENDMONTH;
    %global STOP MONTHNUM;
    %if &MONTH=JAN %then %let MONTHNUM=01;
    %else %if &MONTH=FEB %then %let MONTHNUM=02;
    %else %if &MONTH=MAR %then %let MONTHNUM=03;
    …
    %else %if &MONTH=OCT %then %let MONTHNUM=10;
    %else %if &MONTH=NOV %then %let MONTHNUM=11;
    %else %if &MONTH=DEC %then %let MONTHNUM=12;

%let THISMONTH=%sysfunc(mdy(&MONTHNUM,1,&YEAR));
%let ENDMONTH=%eval(%sysfunc(intnx(month,&THISMONTH,1))-1);
%let STOP=%sysfunc(day(&ENDMONTH));
%put THISMONTH=&THISMONTH;
%put ENDMONTH=&ENDMONTH;
%mend ENDMONTH;
```

Though not one of the actual submacros used to directly construct the FTP script files, the ENDMONTH macro is discussed here because it is required to generate the last day of the month (macro variable STOP) being processed, as well as the numeric equivalent of the MONTH parameter (MONTHNUM). Rather than call ENDMONTH each time we process a new facility, if we set the two critical values of STOP and MONTHNUM as global macro variables, they will be available each time a new facility is processed, so we can call ENDMONTH just once prior to running the macros required to generate the FTP script files.

ENDMONTH first employs a series of %IF-%THEN-%ELSE statements to convert the 3 capital letter format of the MONTH macro parameter to a two-digit format (MONTHNUM) used in the Ops History filenames we wish to copy from the FTP site. The logic to get the last day of the month is quite simple. First, use %SYSFUNCT and the MDY function to assign the first day of the current month as a SAS date value to the macro variable THISMONTH. Next, we use %SYSFUNC and the INTNX function* to increment the date to the first day of the next month and then, adding a "-1" and a %EVAL, we get the SAS date value corresponding to the last day of the current month, and assign it to the macro variable ENDMONTH. Finally, we use %SYSFUNC and the DAY function to convert the SAS date to day of the month, and assign it to the STOP macro variable.

RUN MACRO ADDMGET FOR EACH DAY OF THE CURRENT MONTH

```
%MACRO ADDMONTH(LOC,MONTHNUM,YEAR);
%do I=1 %to &STOP;
    %ADDMGET(&LOC,&YEAR,&MONTHNUM,%sysfunc(putn(&I.,Z2.)));
%end;
%MEND ADDMONTH;
```

The ADDMONTH macro runs the ADDMGET macro for each consecutive day in the given month. ADDMONTH takes three parameters – LOC, MONTHNUM, and YEAR in 4 digit format. YEAR is a global macro variable set at the very beginning of the SAS program. MONTHNUM and STOP are global macro variables created when the

* The first argument of the INTNX function is typically placed in quotation marks (e.g. 'month','week', etc.) However, since INTNX is used here inside a macro function (SYSFUNC), we do not enclose the character values in quotes as we would outside the macro setting.

ENDMONTH macro is run. A DO loop runs the ADDMGET macro for days 1 to the last day of the month, defined by the macro variable STOP. The end result – as many FTP MGET commands are appended to the script file as there are files for the given month (since there is one file per day).

TIME TO QUIT - LAST ADDITION TO THE FTP SCRIPT FILE

```
%MACRO ADDQUIT(LOC);
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
"P:\folder1\folder2\folder3\folder4\&LOC.\signon"));
%let fid=%sysfunc(fopen(&filrf,a));  *a means APPEND mode;
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fput(&fid,quit));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;     /* unsuccessful open processing */    %end;
%MEND ADDQUIT;
```

Macro ADDQUIT opens the generic SIGNON script file one last time and appends the line – “quit” - on to the end of the file, which when the script file is run will close the FTP environment and return control to the DOS prompt.

TIME TO RUN (THE EDITED FTP SCRIPT FILE)

Keep in mind that all the macros so far have been involved in building the FTP script file required to copy one month of files for a given facility from the FTP site. But imagine what a pain it would be after each time we ran the SAS program to generate an edited script file, to then have to leave SAS, go to the DOS prompt, and navigate to each individual subdirectory and manually run the FTP script file!

```
%MACRO FTPCSV(LOC);
X P: ;
X CD\;
X CD folder1;
X CD folder2;
X CD folder3;
X CD folder4;
X CD &LOC.;

X FTP -i -s:P:\folder1\folder2\folder3\folder4\&LOC.\SIGNON ftp.site.faa.gov;
%MEND FTPCSV;
```

Macro FTPCSV runs the final edited version of the SIGNON script file (that contains all FTP commands needed to obtain an entire month of CSV files from the FTP site) after temporarily exiting SAS and navigating to the P drive subfolder corresponding to the facility.

DELETE THE EDITED FTP SCRIPT FILE

```
%MACRO DELSIGN(LOC);
X erase P:\ folder1\folder2\folder3\folder4\&LOC.\SIGNON;
%MEND DELSIGN;
```

The last submacro called by the PROCESSM macro is called DELSIGN, which simply deletes the modified and executed SIGNON script from the P drive &LOC subfolder so that next month the generic SIGNON script can be copied back in to the subfolder to be modified and executed.

NESTING THE PROCESSM MACRO SO WE CAN RUN ALL FACILITIES AT ONCE

So now we have a quick, efficient way to copy all daily data files for a given facility for a given month with a minimum of effort. At this point, we could copy all the data for all facilities by entering the following in our SAS program:

```
%PROCESSM(LOC1);
%PROCESSM(LOC2);
…
%PROCESSM(LOCn);
```

(where n is the total number of facilities) and clicking on the "Submit" button. Presumably, if the same facilities are processed each month, then we would only have to do this once, so other than a lot of repetitive macro calls in the program code, we would still have accomplished a major savings in file copying time each month. But with just a bit more programming, we eliminate all these nearly identical macro calls in the code, and will always be guaranteed we are copying the data for all relevant facilities. Our solution is to keep the latest list of active facilities in an Excel® spreadsheet (see Figure 1) that can be imported into SAS, and run the PROCESSM macro for each facility in turn. We accomplish this in three steps: a PROC IMPORT, a PROC SQL, and a new macro in which the PROCESSM macro is nested.

Ops Sites		
LOCID	**Classification**	**Facility Name**
ABI	X	Abilene ATC Tower, TX
ACT	X	Waco ATC Tower, TX
ALO	X	Waterloo ATC Tower, IA
AMA	X	Amarillo ATC Tower, TX
ASE	X	Aspen ATC Tower, CO
AUS	X	Austin ATC Tower, TX
AVL	X	Asheville ATC Tower, NC
AVP	X	Wilkes-Barre/Scranton ATC Tower, PA
AZO	X	Kalamazoo ATC Tower, MI

Figure 1. Sample of lines from Excel spreadsheet to be imported

IMPORT THE LATEST LIST OF FACILITIES

```
proc import datafile="P:\folder1\folder2\folder3\folder4\FacilityList.xls"
                 out=LIST_VAL dbms=EXCEL2000 replace; sheet="Validated";
getnames=NO; run;

data ALL_LOCS(drop=F1 F3);
    length LOCID $3 NAME $35;
    set LIST_VAL(keep=F1 F3);
    LOCID=substr(F1,1,3);
    NAME=left(F3);
```

```
    if length(LOCID)=3 and anyalpha(LOCID)=1 and upcase(LOCID) not in ('COU','LOC');
run;
```

First we use PROC IMPORT to import records containing valid sites from a tracking spreadsheet called FacilityList.xls. Due to the unique structure of the spreadsheet being imported, we choose not to import column names directly (i.e. GETNAMES=NO).

Next, we take the raw SAS dataset created in the PROC IMPORT and extract the LOCID and NAME of each site. The LOCID values were in the first column of the spreadsheet, but so was some extraneous information, so we keep LOCID if it is 3 characters long, the first character is a letter, and it is not an illegitimate LOCID (such as LOC which comes from LOCID or COU which was extracted from the word COUNT which, though not shown in Figure 1, appears in the first column of the spreadsheet below the LOCID listings). The new SAS dataset is called ALL_LOCS.

CREATE A MACRO VARIABLE CONTAINING ALL FACILITY IDS

```
proc sort data=ALL_LOCS; by LOCID; run;

proc sql noprint;
select distinct LOCID into :LOCLIST separated by ' '
from ALL_LOCS
order by LOCID;
quit;

%put &LOCLIST;

%global NUM_FACILITIES;
%let NUM_FACILITIES=&SQLOBS;
```

Then, after sorting ALL_LOCS by LOCID, we use PROC SQL to move the distinct LOCID list into the space delimited macro variable called LOCLIST. A %PUT statement is included so that the list of LOCIDs being processed would print out in the SAS log. Finally, after declaring the global macro variable NUM_FACILITIES, we set it equal to the automatic macro variable SQLOBS, the number of distinct rows (i.e. LOCIDs) processed in the PROC SQL.

PROCESS ALL FACILITIES (WITH PROCESSALL)

```
%MACRO PROCESSALL;
%do J=1 %to &NUM_FACILITIES;
%let LOCID=%sysfunc(scan(&LOCLIST,&J,' '));
%PROCESSM(&LOCID.);
%end;
%MEND PROCESSALL;
```

The PROCESSALL macro loops through all LOCIDs in the LOCLIST macro variable, running PROCESSM for each facility. This is done by using a DO loop the upper bound of which is the macro variable NUM_FACILITIES, which is the number of distinct LOCIDs stored in the macro variable LOCLIST. As each successive facility is processed, the J counter increments and by using the SCAN function with a space character delimiter, the macro variable LOCID is assigned the next successive facility ID in the list.

CONCLUSION

This paper has shown how to use a few basic FTP commands and a few SAS external file functions to construct a file that when properly accessed can batch copy a large number of files from one location on an FTP site to a location on a local drive or network folder. By adding in some enhancements such as nesting macros and space-delimited macro variables, one can greatly increase the number of files and locations to copy from. Though it is quite possible that there may be more efficient methods to accomplish large scale batch copying, the reader is encouraged to think of routine data management tasks in the workplace where the techniques used in this paper, or other techniques employing additional SAS external file functions, might be of use in saving time.

ACKNOWLEDGMENTS

The views expressed in this paper are those of the author and do not necessarily reflect the position or policy of the Federal Aviation Administration, the U.S. Department of Transportation, or any other United States government agency.

I wish to thank Glen Buchanan, Manager of Workforce Planning, in the Financial Analysis Office of the Air Traffic Organization at the Federal Aviation Administration (FAA) for his support of my efforts to automate more of our office's workload using SAS. I also wish to thank Akira Kondo, organizer of the FAA SAS Users Group, for his continuing efforts to increase the visibility of SAS in the FAA and for his review of this paper. I wish to thank Anne Kearney of the U.S. Census Bureau at the U.S. Department of Commerce, Jason Ford of the Bureau of Labor Statistics at the U.S. Department of Labor, and Heidi Markovitz at the Federal Reserve Board of Governors for their careful review of this paper and for their extremely helpful edits.

REFERENCES

[1]NSFTOOLS.COM: Tools and Tips for Lotus Notes and Domino. "List of FTP commands for the Microsoft command-line FTP client " (http://www.nsftools.com/tips/MSFTP.htm).

CONTACT INFORMATION

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

Arthur Furnia
Federal Aviation Administration
AJF-45
800 Independence Avenue, SW
Washington, DC 20591
Work Phone: 202-267-9481
Email: Arthur.Furnia@faa.gov