

# **Large Scale Data Building with MP-Connect**

## **The Precursor to Statistical Analysis & Modeling**

### **David Cedillo**

#### **Introduction**

In the financial service industry, collecting and storing data allow firms the ability to mine data for competitive advantage. The idea of using data to create this competitive advantage is simple in concept but becomes much more involved once you move from concept to reality. For instance, many firms record large amounts of data on a daily, weekly, monthly, quarterly, or yearly basis. With these historical snapshots, you must then decide which data will address your needs and on what time period your analysis should focus on. Once these preliminary questions are resolved, you then move to the daunting “data build” phase of the project. How can you pull massive amounts of data together into a standardized data format that will support the project analysis and reporting? More importantly, how can you pull all this data in a time efficient manner so that this process can support multiple “do-over” scenarios?

There are multiple ways to address data building, in fact, the simplest form in SAS is the data step. If your project is using small data sets and limited amounts of history, then the data step by itself is probably sufficient. When your projects incorporate large data sets and a number of historical snapshots, the standalone data step can still be used, but the process becomes more time inefficient with larger amounts of data. In this instance, more advanced techniques in the data build are needed to decrease the amount of time spent massaging data.

This paper will focus on utilizing MP-Connect to pull large amounts of data and then combine the resulting data sets from each MP-Connect session for down stream processing. The concept behind this paper is that SAS can manage/manipulate smaller chunks of data more quickly than one large chunk of data. Emphasis will be placed on an actual working MP-Connect template that passes data sets, libraries, options, and macro variables to and from the Parent & Child sessions. Additionally, the template will also assist with MP-Connect session management and passing an autocall macro library. The intent of this paper is to establish a consistent data build process that is re-usable across different projects.

#### **General Technical Information (for this paper)**

SAS: v8.2 (Base SAS & SAS/Connect)

Unix Server: Solaris O/S with 12 Dual Core Processors @ 1500Mhz

User Experience: Understand Macro Functions, Macro Looping, SAS Dates, and general understanding of multi-processing vs. non multi-processing.

#### **MP Connect Considerations & Concerns**

A few papers have previously been written on MP-Connect or multi-processing (MP) to explain general concepts of MP-Connect. Given a general understanding of multi-processing, determining if multi-processing can be useful on your project will be assisted by your answers to the following questions:

1. Are the data source(s) split so that they can be processed independently?
2. Can the program be split into multiple mutually exclusive SAS sessions that can run at the same time and not depend on joint non-sharable files?
3. Do you get a good return on investment of time used for program development of an MP-Connect program versus a non MP-Connect program?
4. Can the impact to the server and users from multi-processing be limited to an acceptable level?

Sources: Garner, SUGI25 & Bentley, SUGI27

For questions #1 & #2, the answer depends on an understanding of the data being used for the project. Also, this paper attempts to address question #3 & #4 by providing a working MP-Connect template to limit program development time and cap the number MP-Connect sessions issued at one time.

Additionally, question #4 addresses a valid concern with multi-processing. Numerous MP-Connect sessions requiring large data transfers can cause network and input/output (I/O) bottleneck issues. Each

MP-Connect sessions is a separate SAS session with it's own WORK space. If all SAS sessions utilize only 1 directory for WORK space then you can potentially eliminate any time saving to I/O issues. One way to combat this issue is ensure your I/O paths are maximized to the WORK directory. Conversely, you can separate your WORK area across multiple directories that correspond to different I/O channels.

### **Business Case: Predictive Behavior Model - Data Build Concept**

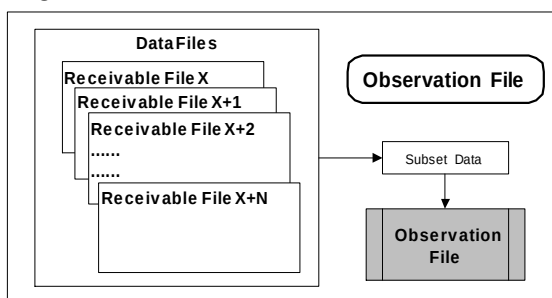
In the financial service industry, specifically the consumer lending firms, consumer delinquency is a primary concern. If a customer becomes too delinquent then the possibility of loan default increases thus exposing the firm to potential losses. Generally, when loan default occurs the customer is not likely to repay the outstanding loan balance therefore the firm loses the remaining monies not paid. If the loan is secured with some type of collateral then the firm can limit its losses by claiming the collateral for resell. In addition to this actual loss, the company also losses the future interest gains from the loan.

It is unlikely that any company will eliminate all loan defaults due to unexpected events to the customer or the economy. Given all firms have intentions of remaining profitable, then managing their loan default loss exposure through pre-emptive actions is one of the key tasks of many lending companies. In managing this loss exposure, the lending company can accomplish this in two ways: 1) At time of lending. 2) Managing their existing customer payment behavior. The business example will focus on creating data to support management of existing customer behavior.

Even with the emphasis on existing customer behavior, this data build concept is general enough to support various predictive modeling and vintage analysis projects. When gathering data for a behavior model, the idea is to pull records from a certain time period (or observation period) and track the performance and history of that record. The performance period occurs post observation, while the history occurs pre observation. The history data provides useful information that can be used in the model if found to be significant in predicting the delinquency outcome. The observation period contains all information, including history, that can be used for modeling. The performance period allows the analyst to test the accuracy of the model by comparing the model to actual behavior post observation.

The data build will be broken into 3 sections: 1) Pulling Observation records. 2) Pulling History/Performance records for observation records. 3) Combining the Observation and Hist/Perf data. See Figures 1-3 for a visual representation of these 3 sections.

**Figure 1: Observation Process**



**Figure 2: Hist/Perf Process**

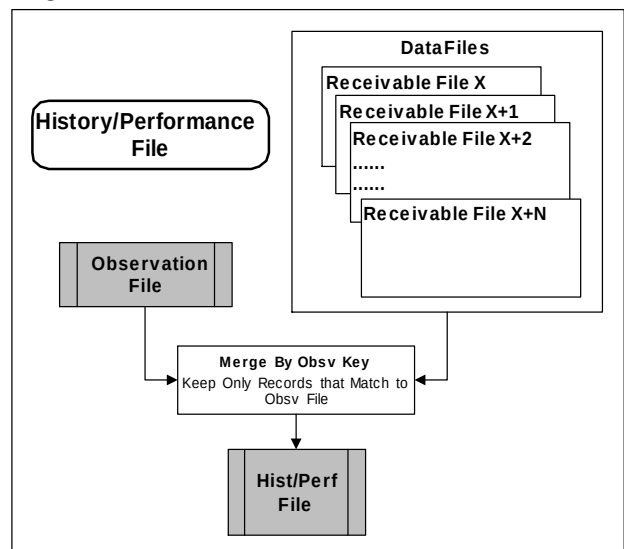
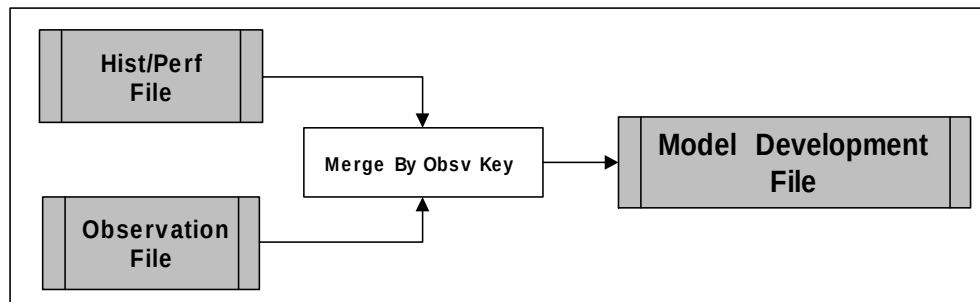


Figure 3: Combine For Model Dev File



Since the goal is to monitor customer payment behavior, this process will pull information from an account receivable system. From Figures 1 & 2, we are pulling multiple files to create these datasets. In these two steps, the pulling of data from each receivable file can be considered independent steps, meaning pulling information from one receivable file does not depend on any information from the other receivable files. Due to the independent nature of steps 1 & 2, these steps are great candidates for multi-processing. The final step is meant to combine the outcome of the two previous steps, via a simple merge, and cannot benefit from multi-processing therefore the remainder of this paper will focus on the first two steps.

#### Data Build Testing – Setup & Explanation

In order to determine the usefulness of multi-processing, a comparison of an equal data build without multi-processing is needed as a baseline. Using the data build concepts previously explained, this test will compare a non MP Connect data build process to an MP Connect process. Both programs will be virtually the same in terms of dataset options and statements.

Using hypothetical time periods to support the customer behavior model, this test will create an Observation period over Jan2005-Jun2005 of customer that are 1 to 29 days late with their payment. Additionally, 12 months of history and 12 months of performance data will be pulled for each observation period.

The data files being used in the process follow a standardized naming convention. They contain a 2-4 letter prefix followed by a year-month stamp (yyyymm). For example, a January 2006 Master File is listed as “mf200601”. The MP-Connect template assumes a standardized naming convention on filenames.

From a server standpoint for this paper, SAS is setup to utilize 3 UNIX file systems for SAS Work spaces. A UNIX script is utilized to assign each SAS session a WORK space area in the least used file system at the point of SAS session startup. Loading WORK spaces across multiple file systems is not necessary to use MP-Connect. MP-Connect can still be used on SAS sessions that assign WORK space to one file system and will likely outperform non multi-processing jobs assuming I/O is at a manageable level.

#### Basic Method – Non MP-Connect

This method consists of a simple data step to pull all files for the observation and hist/perf sections with one data step for each section. The following is an example of creating an Observation population.

```
data ds_obsv (keep=obsv_: acct_key) ;
  set out.mf200501 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
    out.mf200502 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
    out.mf200503 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
    out.mf200504 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
    out.mf200505 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
    out.mf200506 (where=(1 <= acct_dpd < 30) keep=acct_key acct_dpd viewmon)
  ;
  obsv_mon = viewmon ;
  obsv_key = acct_key ;
```

```
run ;
```

A similar step is issued for the Hist/Perf data, except the where option is removed because we want to track all available records on non-subset files. Also, more files are listed in the set statement to accommodate the 12 months of history and performance.

### **MP-Connect Method**

Creating a multi-processing job does involve additional programming time, but if many of your projects are similar in nature then a one time investment in creating the reusable process will pay dividends with future projects. The process being explained will support similar type data build projects in a dynamic fashion. SAS Macros will be used extensively to pass information to and from parent and child sessions. Recall, MP-Connect creates separate SAS sessions with a WORK space specific to that session. The main program that issues the MP-Connect syntax is considered the Parent session in this template and the new SAS sessions that result from the MP-Connect syntax are considered the Child sessions.

The basic MP-Connect syntax being used in this template is as follows:

```
options sascmd="runsas" ;

signon process = job2 ;
rsubmit process = job2 wait=no ;
    /** INSERT SAS CODE ***/
endrsubmit ;
signoff job2 ;
```

The SASCMD option contains the SAS command needed to launch a new SAS session. In addition to the SAS command, a script that launches SAS can also be used with this option. The SIGNON statement creates the process that links the Parent & Child sessions and assigns a process name. The RSUBMIT statement indicates the starting point of SAS code that will be issued to the process listed. The WAIT option on the RSUBMIT is needed to identify if the session should operate in multi-processing or non multi-processing mode (WAIT=YES: multi-processing ; WAIT=NO: non multi-processing). The ENDRSUBMIT statement is the ending point of SAS code that will be issued to the process listed. The SIGNOFF statement closes the process listed. Please see SAS documentation for a detailed explanation of the SAS/CONNECT options and statements.

In this test, a script is used to spread SAS sessions over 3 different WORK file systems on the server. Recall, one of the potential problem points for MP-Connect is an I/O bottleneck that can occur if too much transferring occurs in one directory. Any benefits gained from utilizing MP-Connect can be negated by I/O issues if the I/O paths are small. I/O issues are always a concern regardless of the number of file systems used for WORK space. Also, the author is unaware of any SAS configuration feature that limits the number of remote sessions created per SAS session. For this reason, this MP-Connect template self-imposes limitations on remote sessions.

The template will be explained in the following manner:

1. Remote Session Limits
2. Creating Parent Session Macro Variables
3. Passing Parent Information to Child Sessions
4. Writing files back to Parent WORK Space

#### **1. Remote Session Limits**

This template creates multiple sessions using a Marco Do-Loop based on dates to pull files for various dates. A given do-loop will continue to loop until the do-loop expression resolves to False, therefore arbitrarily issuing MP-Connect statements in a do-loop will likely produce more sessions than your server can handle thus slowing the performance of all users on the server. This template combats this issue by limiting the number of MP-Connect sessions within the do-loop. By limiting the separate sessions to a respectable level, we allow MP-Connect to flourish and not create any bottleneck issues.

Recall, this macro assumes a standardized filename convention. The following are descriptions of the macro parameter and variables:

**edate** – End Date

**num\_mon** – Number of Months backward from End Date

**sdate** – Start Date calculated using End Date & # of Months

**mp\_maxsesn** – Max number of MP SAS sessions allowed at one time (GLOBAL variable)

The actual MP-Connect statements used in this macro follow the basic MP syntax previously explained.

The main purpose of this macro do-loop is to create and control the independent SAS sessions created from MP-Connect.

```
%macro pull_files(edate,num_mon) ;

%let sdate = %sysfunc(intnx(month,&edate,-&num_mon)) ;

%let i = 0 ;
%let int = 0 ;

/**** BEG: Do-Loop For Remote Jobs ****/
%do %while (&sdate < &edate) ;

%if &i > 0 %then %let int = 1 ;
%let sdate = %sysfunc(intnx(month,&sdate,&int)) ;
%let sdatefmt = %sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

%let i = %eval(&i+1) ;

/**** Cap Remote Jobs to &maxsesn (GLOBAL) ****/
%if &i > &mp_maxsesn %then
%do ;
%let y = %eval(&i - &mp_maxsesn) ;
waitfor _any_ job&y ; /* MP-Connect Statement */
signoff job&y ; /* MP-Connect Statement */
%end ;
%else
%do ;
%let y = 0 ;
%end ;

signon process = job&i ; /* MP-Connect Statement */

/**** BEG: Submit Remote Job ****/
rsubmit process = job&i wait=no ; /* MP-Connect Statement */

endrs submit ; /* MP-Connect Statement */
/**** END: Submit Remote Job ****/

%end ;
/**** END: Do-Loop For Remote Jobs ****/

/**** Sign-Off Remaining Remote Jobs ****/
%let remjob = %eval(&y + 1) ;
%let z = %eval(&i - 1) ;
%do x = &remjob %to &i ;
signoff job&x ; /* MP-Connect Statement */
%end ;

%mend pull_files ;
```

The %pull\_files macro creates MP jobs in sequential order up to the session limit and then waits for the oldest job to complete before a new session is created. For example, if your project requires you to pull 5 files independently and you limit your MP sessions to 3, then %pull\_files macro will create the following MP-Connect statements.

```
signon process = job1 ;
rsubmit process = job1 wait=no ;
endrs submit ;
signon process = job2 ;
```

```

rsubmit process = job2 wait=no ;
endrssubmit ;
signon process = job3 ;
rsubmit process = job3 wait=no ;
endrssubmit ;
waitfor _any_ job1 ;
signoff job1 ;
signon process = job4 ;
rsubmit process = job4 wait=no ;
endrssubmit ;
waitfor _any_ job2 ;
signoff job2 ;
signon process = job5 ;
rsubmit process = job5 wait=no ;
endrssubmit ;
signoff job3 ;
signoff job4 ;
signoff job5 ;

```

Given only 3 MP sessions are allowed at one time, job #4 will not be issued until job #1 completes due to the WAITFOR and SIGNOFF statement preceeding the SIGNON job4 statement. In order to prevent unclosed MP sessions, the macro issues SIGNOFF statements for the remaining jobs not closed in the macro. Avenues do exist to optimize this macro so that newest job is not directly tied to the oldest job, but the coding involved to accomplish this optimization could prove to be very cumbersome to incorporate.

## 2. Creating Parent Session Macros

The primary purpose for parent session macros in this template is pass the Parent work directory and options to the Child sessions. The following statements capture the physical work directory, observation option and user defined SAS macro library path. These macro variables will be issued and used in the Child sessions.

```

%let mp_obs      = %sysfunc(getoption(obs)) ; /* Observation Option */
%let mp_wrk      = %sysfunc(pathname(work)) ; /* Work Lib */
%let mp_mcr      = %sysfunc(getoption(sasautos)) ; /* Autocall Library */

```

The observation option is useful when developing your program and checking syntax as you can limit the number of observation in both the Parent and Child sessions. Both the observation and autocall options allow the user to make a change in the Parent session and know that the changes will occur in the Child sessions. The WORK path macro is instrumental in allowing each Child session to create datasets in the Parent WORK space.

## 3. Pass Parent Information to Child Sessions

SYSLPUT is a SAS/Connect macro that creates a new macro variable on a remote session. Using the macro variables created in the Parent session, this MP template passes information to each Child session.

```

%syslput robs    = &mp_obs ;
%syslput rmcr    = &mp_mcr ;
%syslput wrk     = &mp_wrk ;

```

## 4. Write Child Files to Parent WORK Space

Once the physical path for the Parent WORK space is captured, it can be used in the Child session as any other library for data via the LIBNAME statement. For ease of understanding, the Child library that points to the Parent WORK space is called "wrk". In the Child session, any files written to library "wrk" will actually write to the Parent WORK space.

```

signon process = job2 ;
%syslput robs    = &mp_obs ;
%syslput rmcr    = &mp_mcr ;
%syslput wrk     = &mp_wrk ;

rsubmit process = job2 wait=no ;

options obs=&robs sasautos=&rmcr ;

```

```

libname wrk    v8 "&wrk" ;    /* Parent Work Library */

data wrk.parent_file ;
    set child_file ;
run ;

endrsubmit ;
signoff job2 ;

```

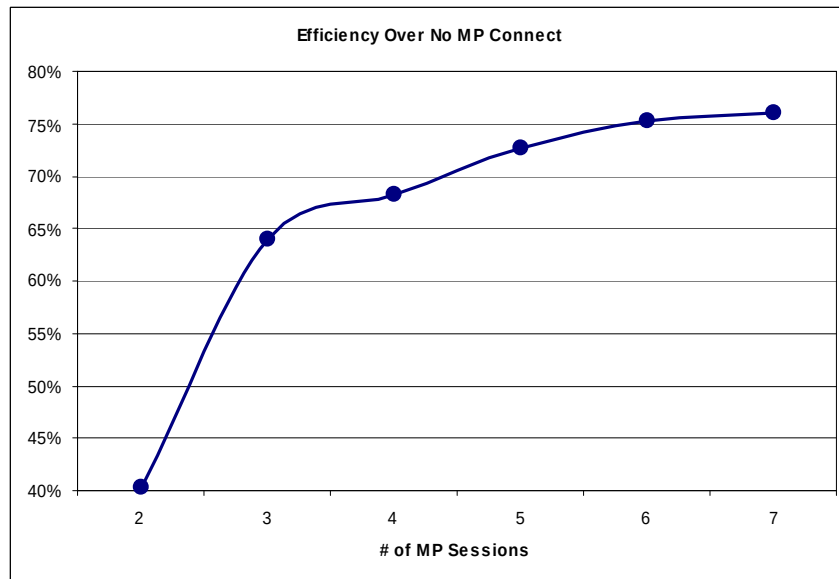
### **Test Cases & Results**

This section lists the actual savings and efficiencies of using MP-Connect. These test are not needed for each MP-Connect program/project. Two test are used to gauge the time savings of multi-processing and the optimal number of MP sessions for use. The test cases cover 3 different file size ranges and each case uses the same number of files. The test case programs are based on Observation and Hist/Perf of the model data build described in the Business Case. See Table 1 for data specifics on the test cases.

| Test Case | Avg Rows  | Avg Columns | Num of Files |
|-----------|-----------|-------------|--------------|
| #1        | 4,348,660 | 427         | 30           |
| #2        | 794,654   | 135         | 30           |
| #3        | 273,065   | 88          | 30           |

**Table 1: Data File Specifics**

Before using MP-Connect on your server, a one time test for optimal multi-processing sessions is useful so that your program does not suffer from I/OU issues. The test is only needed once as you can use the results on all MP-Connect programs. Using test case #2 as a base, the same program is issued with different MP session cap limit numbers. The resulting value from this test will be used to populate the GLOBAL macro variable (mp\_maxsesn) in the MP-Connect template. Figure 4 indicates that incremental savings are smaller for programs that use 6 or more jobs at one time.



**Figure 4: Efficiency Gains Per MP Session**

With this information, 5 multi-processing sessions are used throughout this paper which provides an average 73% efficiency gain over non multi-processing jobs. Additionally, 3 or 4 could be used instead as the max MP session value and the time increase from using less MP sessions would still be minimal when compared to a 5 session limit.

After identifying the number of max sessions to use with the MP-Connect jobs, the next test was based on how multi-processing performed on different files sizes. The following table shows the actual time savings and efficiency increase over non multi-processing jobs. Refer to the test case table for information on the file sizes for each test case. Table 2 show the test to be consistent with an expected 73% efficiency gain on average.

| Test Case | Sessions | Time     |          |         | MP-Connect     |              |
|-----------|----------|----------|----------|---------|----------------|--------------|
|           |          | Start    | End      | Program | Savings        | Efficiency % |
| # 1       | No-MP    | 19:19:03 | 20:49:20 | 1:30:17 |                |              |
|           | MP       | 20:49:21 | 21:12:08 | 0:22:47 | <b>1:07:30</b> | <b>74.8%</b> |
| # 2       | No-MP    | 19:08:14 | 19:16:22 | 0:08:08 |                |              |
|           | MP       | 19:16:25 | 19:19:02 | 0:02:37 | <b>0:05:31</b> | <b>67.8%</b> |
| # 3       | No-MP    | 19:06:06 | 19:07:48 | 0:01:42 |                |              |
|           | MP       | 19:07:49 | 19:08:14 | 0:00:25 | <b>0:01:17</b> | <b>75.5%</b> |

**Table 2: Time Savings & Efficiency (hh:mm:ss format)**

The test information is specific to the sever being used for these test (see Gen Tech Info). You results will vary based on the size and speed of the server being used. In the end MP-Connect is always faster and more efficient than non MP-Connect programs.

## **Conclusion**

Using multi-processing, I was able to perform certain task in roughly ¼ of the original processing time. As the testing shows, some of the actual time savings are small (eg. 5 mins), but these test were simplified to showcase the efficiency for model development data builds. In practice, data builds for model development will typically include additional files that will slow down the Non-MP approach thus further justifying the use of MP Connect. If your projects require pulling a number of files and the files are situated to be mutually exclusive, then creating a dynamic MP-Connect template will be useful for each project. In the end, the time investment for MP coding is worthwhile in the long run if many of your projects are similar in nature. Additionally, the MP-Connect template is suited for dynamic features which will make this template more robust and reusable (see Appendix).

## **Reference & Resources**

Garner, Cheryl, "Multiprocessing with Version 8 of the SAS System", SUGI 25, #16-25.

Bentley, John E, "Passing Values to a Remote Multi-Process SAS/Connect Session", SUGI 27, #107-27.

Kearns, William & Keefer, Tom, "Performance Tuning & Sizing Guide for SAS Users and Sun System Administrators – Updated Edition", SUGI 28.

"SAS/Connect User's Guide", SAS Online Doc, Version 8.

## **Appendix: Data Build Example for Model Development**

**Note: %dsstmt is a macro that list the dataset names for a SET statement. This macro allows the template to be more dynamic and reusable.**

```
options obs=max nomlogic mprint nosgen msglevel=i linesize=120 ps=96 mautosource
autosignon=yes sascmd="runsas" sasautos = ("/sas_macros") mrecall mergenoby=error
fullstimer compress=yes nocenter orientation=landscape ;
```

```
libname out v8 "/data" ;
```

```
/******
/****          BEG: Macro Setup          ****/
/******
/****          BEG: MP Connect Parameters          ****/
/**** MP-Connect Parameter - EDIT ****/
%let mp_out_lib = %sysfunc(pathname(out)) ; /* Data Lib */

/**** MP-Connect Parameter - NO EDIT ****/
%let mp_maxsesn = 5 ;
%let mp_obs = %sysfunc(getoption(obs)) ;
%let mp_wrk = %sysfunc(pathname(work)) ; /* Work Lib */
%let mp_mcr = %sysfunc(getoption(sasautos)) ;
/****          END: MP Connect Parameters          ****/
/******

/*****          BEG: Macro Parameters          *****/
/**** Perf Files - Obsv Popl ****/
%let operf_edate = %sysval('01JUN2005'd) ;
%let operf_edatefmt = %sysfunc(intnx(month,&operf_edate,0),yymmnn6.) ;
%let operf_num_mon = 5 ;
%let operf_dsn = obsv ; /* Data Set Prefix */
%let operf_lib = work ; /* Libname */

/**** Perf Files - Perf ****/
%let pperf_edate = %sysval('01JUN2006'd) ;
%let pperf_edatefmt = %sysfunc(intnx(month,&pperf_edate,0),yymmnn6.) ;
%let pperf_num_mon = 29 ;
%let pperf_dsn = perf ; /* Data Set Prefix */
%let pperf_lib = work ; /* Libname */
/****          END: Macro Parameters          *****/
/******

/*****          BEG: Data Set Listing of Data Files          *****/
%macro dsstmt(edate,num_mon,dsn,lib) ;

%let sdate = %sysfunc(intnx(month,&edate,-&num_mon)) ;

%let i = 0 ;
%let int = 0 ;

%do %while (&sdate < &edate) ;

%if &i > 0 %then %let int = 1 ;
%let sdate = %sysfunc(intnx(month,&sdate,&int)) ;

%let i = %eval(&i+1) ;

%let dsname = &lib..&dsn.%sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

/** Note: The &fnlname at the end of the below line does */
/** not end with a ; purposely. The macro works */
/** without erros because of where it is called. */
/** The &fnlname immediately follows the ; to help */
/** with spacing when called. */
%let fnlname = &dsname (in=a&i) ; &fnlname

%end ;
```

```

%mend dsstmt ;
/****      END: Data Set Listing of Data Files      ****/
/*****
/****      END: Macro Setup      ****/
/*****

/****      BEG: Create Obsv Population - Remote Processing      ****/
%macro pull_files(dsn,&date,num_mon) ;

%let sdate = %sysfunc(intnx(month,&date,-&num_mon)) ;

%let i = 0 ;
%let int = 0 ;

/****      BEG: Do-Loop For Remote Jobs      ****/
%do %while (&sdate < &date) ;

%if &i > 0 %then %let int = 1 ;
%let sdate = %sysfunc(intnx(month,&sdate,&int)) ;
%let sdatefmt = %sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

%let i = %eval(&i+1) ;

/****      Create Monthly Dataset Name      ****/
%let dsname = &dsn.%sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

/****      Cap Remote Jobs to &maxsesn      ****/
%if &i > &mp_maxsesn %then
%do ;
%let y = %eval(&i - &mp_maxsesn) ;
waitfor _any_ job&y ;
signoff job&y ;
%end ;
%else
%do ;
%let y = 0 ;
%end ;

signon process = job&i ;

/****      Pass Parent Macro Variables      ****/
%syslput robs = &mp_obs ;
%syslput rmcr = &mp_mcr ;
%syslput rlib = &mp_out_lib ;
%syslput wrk = &mp_wrk ;
%syslput dsname = &dsname ;
%syslput vmon = &sdate ;
%syslput vmonfmt = &sdatefmt ;

/****      BEG: Submit Remote Job      ****/
rsubmit process = job&i wait=no ;

options obs=&robs mprint nosgen nomlogic sasautos=&rmcr msglevel=i ;

libname mfs v8 "&rlib" ; /* Data Library from Parent */
libname wrk v8 "&wrk" ; /* Parent Work Library */

/****      BEG: PULL DATA FILE - INSERT DATA/STEP LOGIC      ****/
data wrk.&dsname (keep=obsv_ key) ;
set mfs.mf&vmonfmt (where=(not missing(dpd30)) keep=key dpd30) ;

obsv_mon = &vmon ;
obsv_key = key ;

run ;
/****      END: PULL DATA FILE - INSERT DATA/STEP LOGIC      ****/

endrsubmit ;
/****      END: Submit Remote Job      ****/

```

```

%end ;
/**** END: Do-Loop For Remote Jobs ****/

/**** Sign-Off Remaining Remote Jobs ****/
%let remjob = %eval(&y + 1) ;
%let z = %eval(&i - 1) ;
%do x = &remjob %to &i ;
    signoff job&x ;
%end ;

%mend pull_files ;

%pull_files(&operf_dsn,&operf_edate,&operf_num_mon) ;
/**** END: Create Obsv Population - Remote Processing ****/
/*****/

/*****/
/**** BEG: Combine Obsv Files ****/
data ds_obsv ;
    set %dsstmt(&operf_edate,&operf_num_mon,&operf_dsn,&operf_lib) ;
run ;

proc sort data = ds_obsv (keep=key) out = uniq_key nodupkey ;
    by key ;
run ;
/**** END: Combine Obsv Files ****/
/*****/

/*****/
/**** BEG: Pull Performance - Remote Processing ****/
%macro pull_files(dsn,edate,num_mon) ;

    %let sdate = %sysfunc(intnx(month,&edate,-&num_mon)) ;

    %let i = 0 ;
    %let int = 0 ;

    /**** BEG: Do-Loop For Remote Jobs ****/
    %do %while (&sdate < &edate) ;

        %if &i > 0 %then %let int = 1 ;
        %let sdate = %sysfunc(intnx(month,&sdate,&int)) ;
        %let sdatefmt = %sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

        %let i = %eval(&i+1) ;

        /**** Create Monthly Dataset Name ****/
        %let dsname = &dsn.%sysfunc(intnx(month,&sdate,0),yymmnn6.) ;

        /**** Cap Remote Jobs to &maxsesn ****/
        %if &i > &mp_maxsesn %then
            %do ;
                %let y = %eval(&i - &mp_maxsesn) ;
                waitfor _any_ job&y ;
                signoff job&y ;
            %end ;
        %else
            %do ;
                %let y = 0 ;
            %end ;

        signon process = job&i ;

        /**** Pass Parent Macro Variables ****/
        %syslput robs = &mp_obs ;
        %syslput rmcr = &mp_mcr ;
        %syslput rlib = &mp_out_lib ;
        %syslput wrk = &mp_wrk ;
        %syslput dsname = &dsname ;
        %syslput vmon = &sdate ;
    %end ;
%end ;

```

```

%syslput vmonfmt = &sdatefmt ;

/**** BEG: Submit Remote Job ****/
rsubmit process = job&i wait=no ;

options obs=&robs mprint nosgen nomlogic sasautos=&rmcr msglevel=i ;

libname mfs v8 "&rlib" ; /* Data Library from Parent */
libname wrk v8 "&wrk" ; /* Parent Work Library */

/**** BEG: PULL DATA FILE - INSERT DATA/STEP LOGIC ****/
data wrk.&dsname ;
merge wrk.uniq_key (in=a) mfs.mf&vmonfmt (in=b keep=key dpd: syuszan) ;

by key ;

viewmon = &vmon ;

if a and b then output wrk.&dsname ;
run ;
/**** END: PULL DATA FILE - INSERT DATA/STEP LOGIC ****/

endrsubmit ;
/**** END: Submit Remote Job ****/

%end ;
/**** END: Do-Loop For Remote Jobs ****/

/**** Sign-Off Remaining Remote Jobs ****/
%let remjob = %eval(&y + 1) ;
%let z = %eval(&i - 1) ;
%do x = &remjob %to &i ;
signoff job&x ;
%end ;

%mend pull_files ;

%pull_files(&pperf_dsn,&pperf_edate,&pperf_num_mon) ;
/**** END: Pull Performance - Remote Processing ****/
/*****

/*****
/**** BEG: Combine AIMS & ALS Perf Files ****/
data ds_perf ;
set %dsstmt(&pperf_edate,&pperf_num_mon,&pperf_dsn,&pperf_lib) ;
run ;
/**** BEG: Combine AIMS & ALS Perf Files ****/
/*****

```