Paper DP-05

Automated Data Collection Using SAS and FTP

Phil Busby, Live Data Systems, Inc.

ABSTRACT

This paper discusses several coding techniques and considerations in using the FTP file access method from SAS to gather data from a remote server. The examples shown are from an application developed by a large financial company in Minneapolis to automate the collection of financial data from the Bloomberg server in New York. Issues discussed include request file preparation, reply file parsing, and error handling.

INTRODUCTION

A large financial institution in Minneapolis needs to collect historical data from the Boomberg server in New York. Bloomberg provides on-line tools for requesting information such as closing prices of equities, commodities, interest rates, exchange rates, and company ratings, but the user must pay for those tools, learn how to use them, customize the tools for his application needs, and run them. Running the on-line queries requires additional preparation because the user's data needs change over time. For example, new risk factors are added to the Value at Risk computations, resulting in new ticker symbols that the user must request data for, or, new counterparties are added, and ratings for those companies are needed for the counterparty risk assessment, or, new futures contracts are started for new futures months, and expired contracts do not need to be requested because there are no more trades after the last trade date for a contract. The user must often make manual changes to programs and request files to get new data. Then the user has to format and convert the result data from Bloomberg into his own applications. In addition, there is considerable time spent just maintaining and executing the requests and the programs that process the results.

This paper presents a streamlined solution to the data collection problem using the SAS FTP access method. A SAS/Base program is invoked to query the Bloomberg server for updates to the futures contract ticker symbols. Once these are added to the user's database in Oracle, a second Bloomberg request program asks the Bloomberg server for updates for all of the ticker symbols corresponding to the risk factors in Oracle, parsing the reply from Bloomberg directly into the Oracle database.

A third program asks for counterparty ratings needed by the user's counterparty Value at Risk analysis. Al three of these programs handle the data transfers automatically and seamlessly, requiring only a few minutes of user monitoring each month.

This system was written for SAS 8.2 running under Windows XP.

REQUEST FILE PREPARATION

We have three programs that ask the Bloomberg server for information. The first asks Bloomberg for information about new ticker symbols for futures or options contract months. The second asks for historical data for all of the symbols in our database. The third asks for the Fitch, Moodys, and S&P ratings for all of our counterparties.

PROGRAM 1: GET NEW TICKER SYMBOLS

This program uses our %get_bloomberg macro to ask for the new symbols for futures chain extensions. (A futures chain is a series of futures contracts expiring in successive future months).

The Bloomberg server is set up to process data requests in a request file transmitted to it. The format of the file includes a header, a list of data fields, and a list of ticker symbols wanted. All fields for all tickers are returned - a matrix structure similar to a SAS data set. The header contains Bloomberg login information and the parameters describing the type of request wanted.

Here is the mainline of this program, shown without the processing macros invoked, to give an overview of how the Bloomberg request file is prepared and submitted, and how the reply file is processed. Notice how macro variables are passed as parameters and global macro variables are avoided. This gives each routine some protection against macro variable name collisions.

```
%macro mainline;
%local startdate;  /* Starting date of
                      request  date range.  */
%local enddate;    /* Ending date of
                         req date range.
                      Usually Yesterday. */
%local nsymbol;    /* Number of
                      symbols  requested. */
%local bbstatus;   /* Bloomberg job
                                 status. */
%local nbadtick;   /* Number of
          symbols with  error message
                       from Bloomberg. */
%local ngoodtick;  /* Number of
         symbols  processed without
                    error by Bloomberg.  */
%local result;     /* Result data set
                                 name. */
%local replyds;    /* Reply data set
                                                 name. */
%let bbstatus =None;
%let nbadtick =0;
%let ngoodtick=0;
%put +++ &progname;
%put +++ &progname Bloomberg
      FTP  request for Futures chain
      extension +++;
%put +++ &progname;

%put VAR library is &ora_dir;
%put Bloomberg folder is &bbfolder;

/* If request file creation fails, do not
                          use old  request file. */
%erase_request_file
       (&bbfolder,&reqfile);

/* Set up a default date range for the
```

```
                    Bloomberg request. */
   %let curdate=
         %sysfunc(today(),date9.);
   %let enddate=%sysfunc(intnx(day,
         "&curdate"d, -1), date9.);
%select_futures_series_curves(curveds);

   /* select Futures commodity series
      risk_factor  curves */
   %get_template_risk_factors(curveds,
     Future_Last_Deliv_Date,
     Future, 'Future', futureds);
   /* get Option Volatility risk_factor
      curves */
   %get_template_risk_factors(curveds,
      Option_Asset_Maturity_Date,
      Option, 'Option Volatility',
      optvolds);

   /* Combine Futures and Option
      Volatility  risk_factors into one
      request file. */
   %combine_futures_and_optvols
    (&future_request_months,futureds,
      optvolds,symbolds);

   /* The symbol data set has the
      found_risk_factor_ids, not the
                              tickerds. */
   %drop_duplicate_ticker_requests
     (symbolds,tickerds);


%create_request_file(tickerds,&fieldstr,
      &bbfolder,&reqfile,
      &startdate,&enddate,nsymbol);
   %get_bloomberg
      (&reqfile,replyds,bbstatus);
   %if "&bbstatus" = "Finished" %then
   %do;
      /* Verify the tickers that came back
         valid.  Create result data set. */
   %get_reply_data(replyds,&fieldstr,
      resultds,nbadtick,ngoodtick);
   %put &ngoodtick symbols were
     processed.;
   %match_replies_to_curves
      (resultds,symbolds,matched);
    %drop_expired_futures(matched);

%generate_update_data_sets(matched,
      new_rfs,new_futures,new_optvols);
    %check_update_validity(new_rfs,
      new_futures,new_optvols);
    %update_oracle(new_rfs,
      new_futures,new_optvols);
      %put &progname completed
         successfully.;
      %end;
   %else %do;
      %put &progname execution failed.;
      %end;
   %if &nbadtick ne 0 %then %do;
      %error(-1,WARNING: &nbadtick
        symbol(s) listed above had
        errors.);
      %end;
   %put +++ end &progname +++;
%mend mainline;

%mainline;
```

PROGRAM 2: GET HISTORICAL DATA

The second Bloomberg request is for historical prices for every ticker in our portfolio. We extract the Bloomberg ticker symbols and produce a request file.

If only one field is wanted, the vertical format (one field per returned record) is OK, but for futures, we want the implied volatility of the nearby options as well. The horizontal format yields a smaller replyfile in the case of multiple fields wanted, but requires that

the SAS program parse the resulting data record fields. So when preparing the request file, we read the table of risk factors and remember the risk factor ID and the Bloomberg field it pertains to (either the last trade price or the implied volatility) so that we can match the reply for the Bloomberg ticker back to the risk factor we want data for. In the case of a commodity future, there will be two risk factors (price and volatility) but only one request for the corresponding Bloomberg ticker.

Note also that the header date range of the request is applied to the entire list of tickers. We have enhanced the request to generate a date range wanted for each ticker, because most of the time we already have data for tickers up to the last time the program was run, but in the case of newly-added risk factors, we need to fetch historical data going back many months, so it is better to specify the date range by ticker to minimize transmission of redundant data.
Here is our enhanced request file (just two ticker symbols are shown):

```
START-OF-FILE
FIRMNAME=dl000000
COMPRESS=yes
FILETYPE=pc
DATERANGE=20040322|20040322
HIST_FORMAT=horizontal
PROGRAMNAME=gethistory
START-OF-FIELDS
PX_LAST
HIST_CALL_IMP_VOL
END-OF-FIELDS
START-OF-DATA
ADUSV1D Curncy|TICKER|20040313|20040322|
NGZ3 Comdty|TICKER|20031126|20040322|
END-OF-DATA
END-OF-FILE
```

PROGRAM 3: GET COUNTERPARTY RATINGS

In the counterparty VaR process, we use ratings from three rating services. These fields all have their own field names on Bloomberg, so we request all of them for each ticker symbol. Here is a sample abbreviated request file for counterparty ratings. The only thing that changes for each generation of this request file is the list of ticker symbols between 'START-OF-DATA' and
'END-OF-DATA' (only two ticker symbols are shown below):

```
START-OF-FILE
FIRMNAME=dl000000
COMPRESS=yes
FILETYPE=pc
HIST_FORMAT=horizontal
secmaster=YES
PROGRAMNAME=getdata
START-OF-FIELDS
NAME
RTG_FIBCA_SEN_UNSECURED
RTG_FITCH_ISSUER_RATING
RTG_FIBCA_LT_FC_DEBT
```

```
RTG_FIBCA_LT_LC_DEBT
RTG_FITCH_LONG
RTG_FITCH_OUTLOOK
RTG_MDY_SEN_UNSECURED_DEBT
RTG_MDY_ISSUER_RATING
RTG_MDY_LT_FC_DEBT_RATING
RTG_MDY_LT_LC_DEBT_RATING
RTG_MOODY_LONG
RTG_MDY_OUTLOOK
RTG_SP_LT_LC_ISSUER_CREDIT
RTG_SP_LT_FC_ISSUER_CREDIT
RTG_SP_ISSUER_RATING
RTG_SP_LONG
RTG_SP_OUTLOOK
END-OF-FIELDS
START-OF-DATA
1002Z AV Equity
GM US Equity
END-OF-DATA
END-OF-FILE
```

Now, all three of these programs invoke the same processing macro GET_BLOOMBERG. This macro is stored in our SAS Macro autocall library, because the process of submitting a reply file and getting back a result file is the same for all programs.

Here is what GET_BLOOMBERG looks like:

```
%macro get_bloomberg
   (l_reqfile,l_replyds,l_bbstatus);
   *      Fetch data from Bloomberg.
   * Input:
   *   &l_reqfile   Name of flat file of
       formatted Bloomberg request.
   * Output:
   *      l_replyds    Name of SAS data
          set of  unparsed Bloomberg reply
          lines to request.
   *   l_bbstatus   Name of SAS macro
       variable  loaded with result status.

 %local bbuser bbpass bbhost bbbkup
    bbekey  qbbuserq qbbpassq qbbhostq
    qbbbkupq;
 %local qreplyfnq;  /* Single-quoted
                              replyfile. */
 %local starttm;   /* Bloomberg job start
                                  time. */
 %local replyfn;    /* Replyfile name
                              from BB */
 %let replyfn  =None;

 %local bbfolder;
 %let bbfolder=&results_base;
 %local copysec;
 %local pollsec;
 %local outsec;
 %let copysec=10;       /* Number of
       seconds to wait for BB .copied file
       to appear.*/
 %let pollsec=12;       /* Number of
        seconds to delay before next poll
        for output. */
%let outsec=250;        /* Number of
       seconds to wait for BB to process a
       request.  */

 %if not %sysfunc(fileexist(
    "&bbfolder.\&l_reqfile..txt"))
    %then %do;
    %error(-1,Bloomberg request file not
       found);
```

```
    %end;

   /* Read Oracle for Bloomberg login
      parms */
%get_bbuserid(bbuser,bbpass,bbhost,
bbbkup,bbekey,qbbuserq,qbbpassq,
qbbhostq,qbbbkupq);

   /* the replyfile name depends on
       whether or  not previous requests
       were made today: */
   %get_replyfile_name(&l_reqfile,
         &qbbuserq,  bpassq,&qbbhostq,
         replyfn,qreplyfnq);
   %if "&replyfn" = "None" %then %do;
      /* Unable to get ANYTHING from
        the  Bloomberg server. Try
        backup server: */
      %let qbbhostq = &qbbbkupq;
      %get_replyfile_name(&l_reqfile,
          &qbbuserq,&qbbpassq,
          &qbbhostq,
          replyfn,qreplyfnq);

      %if "&replyfn" = "None" %then
         %error (Neither Bloomberg host
                  could be accessed.);
      %end;
%submit_request_file(&bbfolder,
     &l_reqfile,&qbbuserq,&qbbpassq,
     &qbbhostq,starttm);
   /* Give the server time to copy our
      request */
%await_request_copy(&starttm,
   &copysec,&l_reqfile,&qbbuserq,
   &qbbpassq,&qbbhostq);
%let bbstatus=Processing;

   /* Poll server until replyfile ready */
%await_request_completion
(&starttm,&l_reqfile,&replyfn,&pollsec,
   &outsec,
   &qbbuserq,&qbbpassq,&qbbhostq,
   &l_bbstatus);

   %if "&&&l_bbstatus" = "Finished"
   %then  %get_ftp_results
     (&bbfolder,&l_reqfile,
      &replyfn,&qreplyfnq,&qbbuserq,
      &qbbpassq,&qbbhostq,&bbekey,
      &l_replyds);
%mend get_bloomberg;
```

In our SAS startup, we specify

```
options xsync noxwait;
```

so that the SAS program waits for the X command to finish, but does not make user type “exit.”

Now, you can see from the code above that if primary server is down, then the program tries the backup Bloomberg server automatically.

If the request file is not valid, Bloomberg creates an .err file in the Bloomberg server's user directory, so we must check for the existence of error files and halt if one is found. Error files must be deleted by FTP because the SAS FTP access method does not support DEL of a file.

The poll for the existence of a particular file on the Bloomberg server is done by asking for a directory listing, and looking at the list returned to us.

A 1-second pause is done after submitting the request file so that the Bloomberg server has time to copy the request file.

The poll for reply file is done every 12 seconds to assure the user that the program is running (by displaying a SAS Log message). Here is the get_bloomberg polling loop:

```
%macro await_request_completion
    (l_starttm,
     l_reqfile,l_replyfn,
     l_pollsec,l_outsec,
     l_qbbuserq,l_qbbpassq,
     l_qbbhostq,b_bbstatus);

   *   Wait for the submitted request to
       be  processed by Bloomberg.
   *   When replyfile seen "bbftp.out.gz",
       the request has been finished.
   * Input:
   *    starttm     - the Bloomberg job
       start time
   *    reqfile     - the request file name
   *    replyfn     - the reply file name
   *    pollsec     - the number of seconds
         to poll  Bloomberg for the output
   *    l_qbbuserq  - the single-quoted
                      Bloomberg UserID
   *    l_qbbpassq    - the single-quoted
                      Bloomberg password
   *    l_qbbhostq    - the single-quoted
                        Bloomberg IP addr
   *   Directory listing of our files on the
       BB   server.
   * Output:
   *   SAS macro variable b_bbSTATUS
       =  Finished, TimedOut, BREAK,
          or ReqError;
%local bbstat;
%let bbstat=Processing;
%put Bloomberg replyfile is
  &l_replyfn;
%put;
%do %while(("&bbstat" ne "Finished")
      and
     ("&bbstat" ne "BREAK") and
     ("&bbstat" ne "TimedOut") and
     ("&bbstat" ne "ReqError"));
  /* Stop the mainline loop if user breaks
     out. */
  /* User will break out if he is impatient
     but  */
  /* also if the FTP poll goes down and
     the  data step below hangs up...   */
%let bbstat=BREAK;  /* Assume user
   breaks out of data step below. */
data _null_;
     format starttm endtm duration
        time8.;
     starttm = &l_starttm;
     endtm = time();
     duration = endtm - starttm;
     put 'Bloomberg job duration is '
        duration
       ", pausing for &l_pollsec seconds,
       waiting on Bloomberg.";
   call symput('bbstat','BREAK');
   starttm = time();
   do while
      (intck('second',starttm,time()) <
             &l_pollsec);
   end;
   call symput('bbstat', 'Processing');
   run;

 /* Get directory until req.out.gz  file
    seen. */
```

```
filename dir FTP ' ' ls
    user=&l_qbbuserq
    pass=&l_qbbpassq
    host=&l_qbbhostq;

data _null_;
     /* Assume user breaks out of data
        step: */
     if _n_=1 then call
        symput('bbstat','BREAK');
     infile dir end=eof;
     input;
     /* Make sure request was valid. */
     short =
       index(_infile_,"&reqfile..req.err");
     long  =
       index(_infile_,"&reqfile..req.err.");
     if short > 0 and long = 0 then do;
         call symput('bbstat','ReqError');
         put "&reqfile..req.txt";
         %data_warning(You must delete
           .err  files on BB server before
            rerunning.);
         %data_error(-1,Request file
              contains  errors.);
         stop;
         end;
   /* Get the exact reply file from
      request. */
   /* Actually, we can have more than 9
      replyfiles.  If there is a file with
      suffix   ".10" then we cannot be
      looking for   a file with suffix ".1"
      since they are sequentially made.
      */
    if index(_infile_,"&l_replyfn") > 0
      then call
      symput('bbstat','Finished');
  status = symget("bbstat");
  if status='Finished' or eof then do;
       if status ne 'Finished'
          then call
          symput('bbstat','Processing');
       end;
   run;
filename dir clear;

%if "&bbstat" ne "Finished" %then
%do;
    data _null_;
     length starttmc $ 12;
     starttmc = symget('starttm');
     starttm = input(starttmc,12.);
     if intck('second',starttm,time()) >
        &l_outsec
     then do;
          call symput('bbstat', 'TimedOut');
          put "Replyfile sought was
             &l_replyfn";
          %data_error(-1,Waited >
             &l_outsec  seconds for
            .out.gz.yyyymmdd file.);
          end;
     run;
%end;
   %else %do;
      data _null_;
          format starttm endtm duration
             time8.;
          starttm = &starttm;
          endtm = time();
          duration = endtm - starttm;
          put 'Bloomberg job completed,
             duration was: ' duration;
          run;
    %end;
%end;
%put;
%let &b_bbstatus=&bbstat;
%mend await_request_completion;
```

The GET_FTP_RESULTS macro gets the replyfile back from Bloomberg. Then it must be decrypted and decompressed.

```
filename bbfile FTP &l_qreplyfnq user=&l_qbbuserq
   pass=&l_qbbpassq  host=&l_qbbhostq recfm=v;
   filename locfile "&l_bbfolder.\&b_reqfile..out.gz";
   data _null_;
         retain linecount 0;
         infile bbfile end=eof;
         input;
         file locfile;
         put _infile_;
         linecount = linecount + 1;
         if eof then call symput('linecount',
            trim(left(put(linecount,12.))));
      run;
   filename bbfile clear;
   filename locfile clear;stat;
   filename locfile clear;
```

Watch out for long command strings. There is a maximum length of 200 characters on the command passed to the SYSTEM function. Since our pathnames are long, we test the command string to make sure it ends in "gz" before invoking SYSTEM with it. PUSHD sets our working directory for the subsequent decryption. Decompression works with gzip, which is free. Here are those two command strings:

```
   command =  "pushd ""&l_bbfolder"" &  ""%sysget(ERMSysCom)\tools\des.exe""
     -D -u -k ""&l_bbekey"" &reqfile..out.gz  &reqfile..gz";

    command =  "pushd ""&l_bbfolder"" &  ""%sysget(ERMSysCom)\tools\gzip.exe""
       -d -f &b_reqfile..gz";
```

Now that's all the GET_BLOOMBERG macro does. The caller has to parse the reply file.

REPLY FILE PROCESSING

Let's go back to that second program. It has to get the values for each risk factor out of the reply file and into a SAS data set ready to update the risk factor values in Oracle in three steps:

1. Extract the value for each field for each ticker symbol into an observation.
2. Match it back to our original request.
3. Delete observations where Bloomberg had no data to supply.

The reply file is mostly an echo of the request file header, with results returned in

horizontal format like this:

NGF00 Comdty
|0|2|04/26/1999|2.695000|32.700000|

That reply says "For Natural Gas January 2000 futures on 26APR1999, return code=zero (OK), two data fields, first value (last trade price) is 2.695, and 32.7 is the nearby option implied volatility."

Here is the replyfile parsing code:

```
data t_result(keep=symbol px_date value
     fieldname);
     length symbol $ 20;
     set &l_replyds end=eof;
     retain dataflag  0;       /* Switch for
            predicting a ticker data line */
     retain fieldflag 0;       /* Switch for
            predicting a ticker field line */
     retain nempty   0;        /* Good ticker
            but no  price for date range. */
          /* (Some securities are priced
            weekly. */
     retain nbadtick  0;
     retain ngoodtick 0;
     retain dataseen  0;    /* 1=got
                               symbol data */
     length fieldname $ 32;
     retain fieldname;
     retain maxfield 0; /* number of data
                           fields */
     length field1-field&maxfield $ 32;
     array field[&maxfield] field1-
         field&maxfield;
     retain field1-field&maxfield;
     length charvalue $ 20;
     format px_date date9.;
     if line =: 'START-OF-FIELDS' then
     do;
        fieldflag = 1;
        delete;
        end;
     if line =: 'END-OF-FIELDS' then do;
        fieldflag = 0;
        delete;
        end;
      if line =: 'START-OF-DATA' then
      do;
        dataflag = 1;
        delete;
        end;
     if line =: 'END-OF-DATA' then do;
        dataflag = 0;
        delete;
        end;

if fieldflag then do;
     /* load the field name array */
     maxfield = maxfield + 1;
     field[maxfield] = trim(line);
     end;

if dataflag then do;
     symbol = upcase(scan(line,1,'|'));
     security_rc =
        input(scan(line,2,'|'),5.);
     if security_rc ne 0 then do;
        put 'ERROR: Bloomberg request
            resulted in error for symbol: ';
        put security_rc= symbol= line=;
        %print_bb_message(security_rc);

       /* pfb 14nov2003 we request all
          fields for all symbols, but some
          symbol/field  combinations may
          not match any  risk_factor_id,
          since they were not requested.
          Therefore the -13 error,  Field
          does not apply to security,
          should  not be an error if we get
          back any data for the symbol.  */
```

```
        /* We can check for dataseen
           immediately in horizontal
           format.  For vertical format,
must look at all replyfile entries for the
symbol. */

        if security_rc ne -13 then
            nbadtick = nbadtick + 1;
        end;
        else do;
            ngoodtick = ngoodtick + 1;
            if dataseen = 0 then do;
                nempty = nempty+1;
                /* put 'No data in date range
                   for ticker: ' symbol;*/
                end;
            end;

       /* Third value is the count of the
          number of fields requested. */
       fieldcount = input(scan(line,3,'|'),5.);
       if fieldcount=. then fieldcount=0;
       px_date =
          input(scan(line,4,'|'),mmddyy10.);

       /* horizontal format: all fields on 1
            line */
       do fieldnum=1 to fieldcount;
            charvalue =
               scan(line,4+fieldnum,'|');
          if charvalue='N.A.' then value = .;
          else value =
          input(charvalue,best.);
          fieldname = field[fieldnum];
          output;
          end;
      end;


if eof then do;
      call symput("&l_nbadtick",
          trim(left(put(nbadtick,8.))));
      call symput("&l_ngoodtick",
           trim(left(put(ngoodtick,8.))));
      end;
run;

/* Add the risk_factor_id key variable to
   the result data set. */

proc sql noprint errorstop;
     create table t1_result as
     select a.*,b.px_date,b.value
       from &l_tickerds a, t_result b
      where a.ucsymbol=b.symbol
        and a.fieldname=b.fieldname;
     quit;

%put;
%put &l_result dataset created, number
   of observations is &sqlobs;

proc sql noprint errorstop;
     /* some risk_factors may be for
        expired futures. Since no actual
        data will come back from BB, do
        not update Oracle for these. */
   create table &l_result as
   select * from t1_result
   where px_date is not null
   and value is not null;
   quit;

%if &sqlobs=0 %then %do;
     /* Running for just Saturday, all
        markets closed.  Zero obs is
        expected. */
      %warning(No obs found in result
          data set.);
      %end;
```

Finally we update Oracle with our results. The code below checks what is already in

Oracle, and warns us if there are conflicts.

```
proc sql noprint errorstop;
     create table already as
     select a.*, b.value,
       f.external_index_name
       from var.risk_factor_value a,
       &l_result b, var.risk_factor f
       where
       a.risk_factor_id=b.risk_factor_id
         and
        a.risk_factor_id=f.risk_factor_id
         and a.valuation_date =
         b.px_date;

   create table already_ids as
   select distinct risk_factor_id,
       external_index_name
   from already

   %warning_list(already_ids,rc,Risk
      factor  values from Bloomberg
      were ignored because they
      already exist in Oracle:);

   create table diff as
   select risk_factor_id,
      external_index_name,
      valuation_date,
      risk_factor_value,
      value as bloomberg_value
   from already
   where put(risk_factor_value,best.)
        ne put(value,best.);

   %warning_list(diff,rc,Risk factor
     values from Bloomberg had
     different values than those in
     Oracle and were ignored:);
   quit;

/* See just what risk factor values will
   be inserted. */

proc sql noprint errorstop;
   create table seeum as
   select px_date, risk_factor_id, value
   from &l_result b
   where not exists
        (select 1 from already a
          where a.risk_factor_id =
                b.risk_factor_id
          and a.valuation_date =
                b.px_date);
   quit;

/* Insert new data from Bloomberg
   into Oracle RFV table. */

proc sql noprint errorstop;
     insert into var.risk_factor_value
        ( valuation_date
         , risk_factor_id
         , risk_factor_value)
       select px_date
          , risk_factor_id
          , value
       from seeum;

/* Null out all
external_request_start_date
values in the Risk_Factor table. This
field is  set manually when we want to
add daily data for prior months for a risk
factor that  previously had just monthly
data in Oracle.*/

   update var.risk_factor f
   set external_request_start_date = .
   where external_request_start_date
   is not null
      and risk_factor_id in
   (select distinct risk_factor_id
   from &l_result);
   quit;
```

CONCLUSION

The SAS FTP access method is ideally suited for extracting data from a remote server when you want to validate all of the returned data in your own program. User authentication can be done with a minimum of hassle. The preparation of the request file is automated and is driven by a control table stored in Oracle. The user invokes the SAS System to run one of three programs, and watches on-line while the programs submit a request to the Bloomberg server and await the reply, which usually takes just 2-3 minutes.

CONTACT INFORMATION

Contact the author at:

Phil Busby
Live Data Systems, Inc.
342 Calico Drive
Apex, NC 27523
Work Phone: 919/303-8902
Email: fransioli@hotmail.com
Web: www/cdbaby.com/fransioli