

The Philotic Connection: The Health Data Analysis System

An Adhoc Reporting System

Authors: Charlie Bastnagel & Kevin Gates
Healthsource Provident Administrators, Inc., Chattanooga, TN

ABSTRACT

The Health Data Analysis System is a SAS/AF® PC application designed to allow users to analyze, manipulate, and produce output(reports and graphics) on data in various platforms (OS2, Windows, MVS Mainframe, Red Brick Relational Database, and DB2).

HealthSource Provident, a managed care insurance company, made a strategic decision to begin using SAS so that they could analyze their health care claims on a larger and more accurate scale. We had data in IMS, MVS Db2 tables, MVS SAS, OS2 SAS, and Windows SAS, but our 45 health data analysts had no SAS experience. The question was: How could we give them the benefits of SAS data analysis without changing them from health data analysts to programmers (which they did not want to become)?

INTRODUCTION

By taking a long hard look at how health data was analyzed, and building in the flexibility to meet the 80/20 rule, we were able to build a SAS AF application that functioned as an interface and worked with data on the LAN/PC, on the MVS mainframe, in DB2, in Red Brick, or any other place that allows SQL Passthru and gives SAS the ability

to retrieve information from it.

This interface was designed specifically to:

1. Allow new and experienced users to take advantage of the multitude of analytical tools that SAS provides regardless of what platform the data was on(MVS, mainframe, PC/LAN, DB2).
2. Remove the need for the user to become a programmer in order to complete a project. We allowed the user's request to drive what the system did to accomplish it.
3. Allow the user to SAVE their analytical requests, where clauses, and library profiles so that they didn't always have to start from scratch when they began an analysis or project.
4. Offer output options to external software such as Lotus, Excel, Dbase, Paradox, Geonetworks, and others by developing an Export application. This application created the appropriate export files and text files using SAS Access to PC File Formats as well as the Data Step.

Since Healthsource Provident Health Data Analysts must be able to competently use 6+ MVS applications and 12 or more different software applications, adding SAS to the equation was overwhelming and unrealistic. A whole new department would have to have been created to support and train those employees, who while they were being trained would still continue to have to produce analytical results on a daily basis.

This realization gave birth to an idea of an Adhoc Reporting System that would allow point and click analytical analysis of data on multiple platforms. Same look, same analytical feel, no matter where the data was: **A Philotic Connection**. Throughout the rest of this paper we will refer to this application as by its more functional name: Adhoc Reporting System, for this is its primary strength.

Methodology

The key point behind our methodology was to ask ourselves what do the Health Data Analysts do when they analyze data?

We discovered that most often:

1. They Summarized data on various platforms (MVS mainframe and LAN/PC) using various statistics (sums means variances counts) creating datasets or reports.
2. They usually Grouped by certain variables, usually character, (State City Name).
3. This summarized data might then need to be merged or appended to other data and then that data would be analyzed as above in steps 1-2.
4. Once they had this information they would want it downloaded from the mainframe, if the data was not already on the LAN.
5. In possession of the data they needed for a final report, they wanted the information exported to Lotus, Excel, Paradox, or Dbase where they could combine it with other information or dress it up with Graphics and Fonts.

In asking ourselves this question we discovered our methodology. In the answer to that question lay the main areas that our Adhoc system would need to address:

1. Location of the Data

2. What to do with the data once it was selected (creation of new data from existing datasets, analysis, reporting, graphs)

3. Exporting the data to external software(i.e.Lotus, Paradox, Dbase, Excel, Text, etc...).

Since each of the above is a common need in many applications, we decided to make each of them an application in and of themselves. This way they could be not only be used with our application but with others as well. This decision has incredibly increased our productivity and speeded up application development tremendously.

What each of the three applications above share is a library of SCL Methods that accomplish specific tasks. SCL methods are like procedures in other languages. You send them parameters, you have them do something with that information, and then return to the calling program with new information. Described below are some of our SCL methods, some of them are listed in detail in the Appendix:

Librefs.SCL - Will assign and clear librefs on multiple platforms, MVS, OS2, and Remote Library Services, and check the return codes to ensure the library is truly available.

Lists.SCL - Handles SCL lists. This method is used to take a list of user selections and turn them into a By Variable line (By state city etc...) or into a line for SQL (Prov_nam, City, State), etc....

Listvars.SCL - Method used to assign statistics to variables selected by users in a Listbox. When a user selects a variable a list will popup with statistics to use, this method facilitates use of their selections.

Macrovar.SCL - This very important method will take variables from an SCL List and assign their values to macro variables of the same name. This method works on both local and remote platforms, a very important need with macro programming.

(In the Appendix we have shown some of these and other methods)

SCL Methods are a very valuable component of any development area. Without them, each developer will be writing code, that the developer in the next office is also writing. With SAS this can make supporting applications a nightmare.

We store these methods in a central location, and so whenever a developer needs to assign a library, he/she calls the **LIBREFS.SCL** method. In this way, when a problem arises or some new feature is required, it is addressed in the method and all applications benefit.

In summing up our methodology, the core of our strategy was to let the objective for using the data determine what procedures or programming steps were used to accomplish that task. Thus the user did not have to know the syntactical differences between SQL and Proc Summary, but did need, for example, to know that they wanted to merge(join) two datasets and compute means and sums on the numeric variables. The user lets the system know what objective he/she wants to accomplish and the system figures out how best to accomplish it.

The first step in the process is for the user to let the Adhoc system know where their data is.

Location of the Data

Our data existed in multiple formats in various mainframe and PC systems, plus external data that we received from outside sources. Some of our key data sources were:

1. IMS Claims file on the MVS mainframe read into SAS datasets through an extract process.
2. DB2 tables on the MVS mainframe with health plan information, hospital and physician information, and normative tables with summaries of health data information for each of the past years.
3. Mainframe SAS datasets created through existing SAS applications
4. Datasets on any of our 16 LAN server drives, 1 or more PC drives, and LFS drives which are LAN drives partitioned on our MVS mainframe using IBM Lan File Services.

If you look at Exhibit 1 you will find the application we use to determine location of data and to get users selections of libraries and datasets. In the upper right hand corner is a radio box titled Location of Data. Once the user selects the location of their data our application will react accordingly to fill the library and dataset list boxes.

Important note: We decided to use consistent terminology, so as not to confuse new users. Therefore SQL terminology is not used. Tables are datasets, columns are variables. If someone likes the SQL

terminology, then there is the Query Window.

Even though we may not use SQL terminology and try to use terms consistent with SAS, we do use SQL for certain requests. If someone needs to merge datasets, or needs use of functions(month, zipstate, day) that SQL handles easily, then we will use it to perform the user's request.

So in order to fill these list boxes with the libraries and datasets for a given location we have employed the following:

1. For SAS datasets:
 - a. Local - Use the dictionary.tables views sashelp.vslib and sashelp.vsmember to find out the available libraries and datasets.
 - b. Remote - We download the sashelp.vslib and sashelp.vsmember views to the work library and fill the list boxes with the available libraries on the MVS mainframe.
2. For DB2 and other external Databases that allow SQL Passthru. Submit a passthru query to that system to obtain the available tables and their field names. In our case, we had the advantage of our IS area using mirror tables to describe the MVS DB2 tables to our LAN. So we loaded these mirror tables into our list boxes.

Objective for Using the Data

Once a user has selected his/her data location the user now needs to determine what he/she wants to do with the data.

Some Example Requests:

1. The Network Director needs a count of how many Hospitals, Physicians, and Clinics there are BY State.

2. An account would like to know the volume of the Hospital charges they are incurring by Provider State. They need this report sent in a Lotus spreadsheet overnight.

In truth these are very simple requests in the Health Care Arena, but they will suffice to demonstrate our point.

For example 1, Using base SAS if your dataset contained no numeric variables you could accomplish this two ways. You could not use Proc Summary or Tabulate because they require a numeric variable. Here is a place where SQL comes in real handy. If your dataset Medical.Claims only had two variables: PROVTYPE (type of provider hosp,phys,etc...) and PROV_ST (provider state) you could accomplish this very simply with the following SQL:

```
proc sql;
  create table prov_cnt as
  select provtype, count(prov_st) as count
  from medical.claims
  group by provtype;
quit;
```

Your other option would be to do a data step .

```
data claims;
  retain;
  set medical.claims;
  by provtype;

  if first.provtype
    then count=0;
    else count+1;

  if last.provtype
```

then output;
run;

Or you could do a data step and add a numeric variable like X with a value of 1. Then you could use Proc Tabulate or Summary. However, if you have a million observations this may not be a possible option.

In this case, the SQL offers the quickest solution with the least opportunity for confusion. Even if its not always the fastest, sometimes SQL can be a project saver.

Let's try Example 2 using our application:

An account would like to know the volume of the Hospital charges they are incurring by Provider State. They need this report sent in a Lotus spreadsheet overnight. See Exhibit 2 for a graphical picture of this process.

The user would choose the **Datasets** listbox and select **medical.claims**. Upon this selection, he/she would be prompted with a popup list called **Dataset Function List** (see Exhibit 2) and they would choose **Create Dataset with Report**.

The user then selects the required variables for this request, **Prov_St** and **Chg_amt**, and they are sent to the **Selection List** listbox (see Exhibit 2). The user then selects each of these variables one at a time to determine what he/she would like to do with them. Upon selection of a variable a popup list called **Variable Function List** (see Exhibit 2) is displayed. For **Prov_st** the user would select **Group By**. For **Chg_amt** the user would select **Select Statistic** and then **Sum**.

At this point the request is nearly complete. The only step left is to tell the system where they would like the output file stored and what

name to give it. Once this is done, The user then chooses **Action** from the **Options List** and then **RUN**. When the program is done a new dataset will appear in the **Datasets** window. The user then selects **Export** from the **Options List** and exports the dataset to a Lotus spreadsheet.

At anytime during this request process the user can Choose **Preview** and view their request dynamically. This preview will not be the preview window with SAS code, but rather, a description of their request in logical terms. Something like the following:

You have chosen to:
create work.prov_cnt
from Medical.Claims
grouping by PROV_ST
summing CHG_AMT

The main advantage of this methodology is that we can analyze data whether it's in MVS, OS2, or DB2 and its all done using the same application. As for the SCL programming, the whole function is to create the necessary Source code that will need to be submitted to location where the data is stored.

Export to External Software

The final section involves methods for exporting to external software like Lotus, Excel, Paradox, Dbase, etc...

We used two methods. For creating text files (flatfiles and delimited) we used the Data Step, for Lotus and other software we used SAS Access to PC File Formats.

Here is some sample code for writing a Lotus .wk1 file:

```
%let dsname=&lib..&dsn;
%let filenm= &path%\%trim(%substr(&dsname,
%index(&dsname,.)+1)).wk1;
%put &filenm;

options xsync;

data _null_;
  call system("if exist &filenm erase &filenm");
run;

proc dbload dbms=wk1
  data=&dsname;
  label;
  putnames=Y;
  limit=8191;
  list all;
  load;
  path = "&filenm";
RUN;
quit;
```

Problems Along the Way

- In using Proc Means & Summary, we ran into significant problems with the Class statement with files with observations greater than 1 million. The Class statement consistently caused virtual memory crashes in MVS. Thus for remote submissions we use Proc Summary with a By statement to summarize data.

- Proc Tabulate and Proc Freq were not used in our system because most often we require datasets to be created. Also they have limits as to how many unique occurrences of a By variable they can handle. This number is around 32,000. Since our datasets usually contain millions of observations, this was not an option.

- Management of the MVS mainframe libraries. Many of the users were intimidated by the use of ISPF in managing libraries. Making sure that there was enough space available for large million observation datasets moved us to the use of temporary datasets.

- Each SAS procedure has different ways of specifying statistics and what to call the variables that are produced. Some procedures allow you to specify specific variables for analysis (proc summary), some do the statistic on all the numeric variables listed (proc univariate). Therefore the renaming strategy for each procedure proved challenging.

References

SAS Language Manual

SAS Companion to OS2 Environment

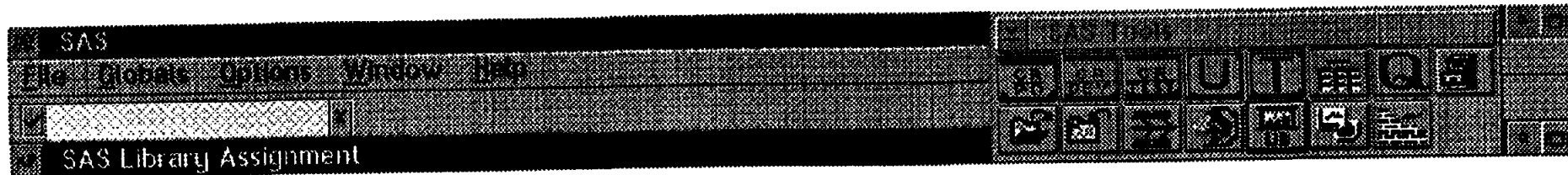
SAS/Access Interface to PC File Formats

SCL Reference 2nd Edition

SAS, SAS/AF, SAS/CONNECT are registered trademarks or trademarks of SAS Institute, Inc in the USA and other countries. IBM, OS/2, LAN Distance, and LAN File Services are registered trademarks or trademarks of International Business Machines Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Thanks to John McCall at the Atlanta Institute for his encouragement and excellent teaching method.



SAS Library & Dataset Selection

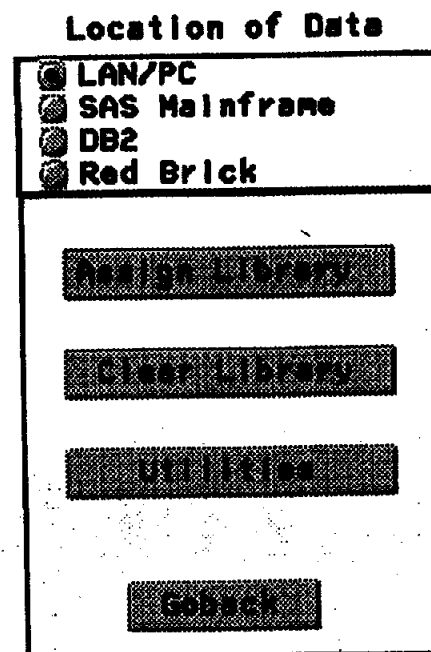
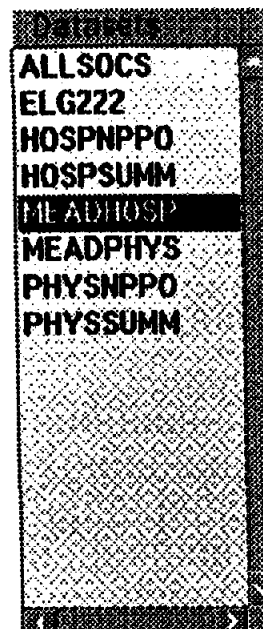
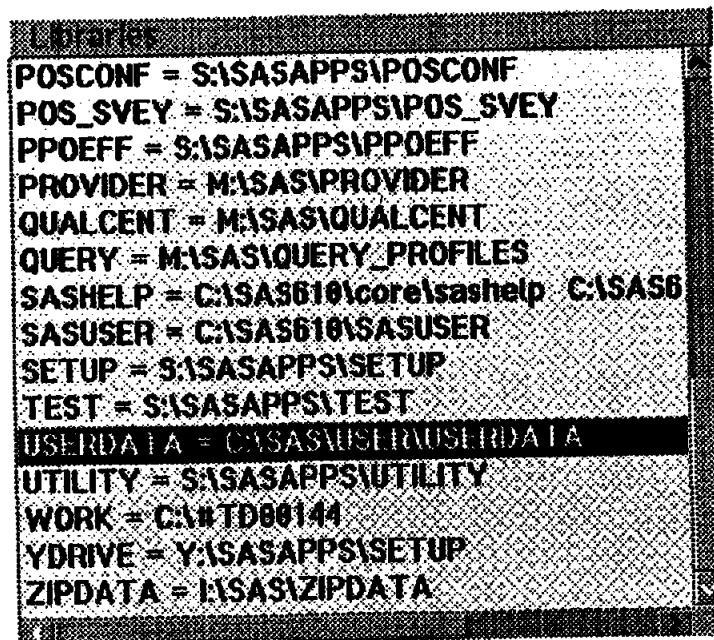
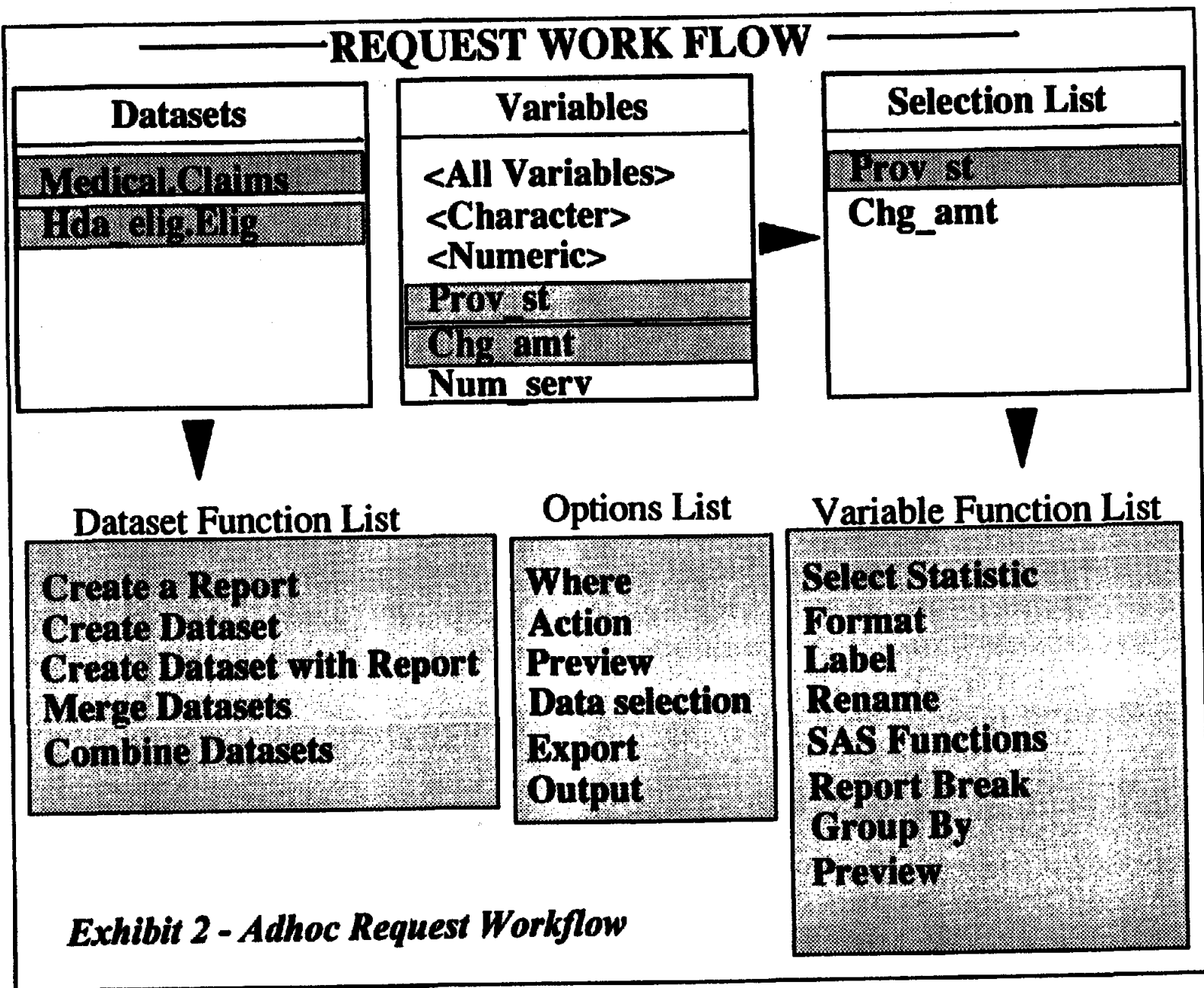


Exhibit 1 - Location of Data



Some Sample Methods referred to in this Paper.

```

/*****
/* LIBREFS method is called from any application that needs a libref assigned */
/* This method works on multiple platforms. LOC is the location of the data */
*****/

```

LIBREFS:

```

method libname $8 path $80 readwrit $3
  loc $3 remote 8 rls $1 libok $1;

link init;
libok='N';
loc=upcase(loc);
rls=upcase(rls);
readwrit=upcase(readwrit);
rlshold='N';

if (remote=1 and loc='SAS') then
do;

/*****
/* Assign Mainframe Library */
*****/

if rls='Y' then
do;
  rlshold='Y';
  rls='N';
end;

call method('librefs.scl','pathchk',libname,path,
  readwrit,loc, remote, rls, libok);
put templib=;

if templib='Y'
then vartxt='libname '||libname||' '||path;
else vartxt='libname '||libname||' '||path||' '|| DISP='||readwrit;

put vartxt=;

submit continue remote;

  %vartxt;
  %let lib_mf=%syslibrc;

  data work.macros;
    lib_mf = input(symget('lib_mf'),8.);
  run;

  proc download data=work.macros out=work.macros;
  run;
endsubmit;

submit continue;
  data _null_;
    set work.macros;
    call symput('lib_mf', lib_mf);
  run;
endsubmit;

lib_mf=symgetn('lib_mf');

call method('librefs.scl','arrmsg',lib_mf,libok);
put 'MF Library RC ' lib_mf= libok=;

/*****
/* Assign Remote Library Services Library */
*****/

if rlshold='Y' then rls='Y';

if rls='Y' and libok='Y' then
do;
  call method('librefs.scl','pathchk',libname,path,
    readwrit,loc, remote, rls, libok);

submit continue;

```

Applications Development

```

libname alibname "spach server=irem_sess;

%let lib_rls=anyalibrc;
%put lib_rls= alib_rls;
%do;
    lib_rls=symgetn('lib_rls');
    call method('librefs.scl','errmsg',lib_rls,libok);
    put 'RIS Library RC ' lib_rls libok;
%end;

%do;
    else if (remote=0 and loc='SAS') then
    do;
        libok='Y';
        put 'Remote Session is inactive. Locations ' loc;
    end;
%do;
    %*****
    /* Assign LAM/PC Local Library */
    %*****
    else if loc='LAM' or loc='PC' then
    do;
        path=compress(path,' ');
        path=compress(path,' ');
        /* no real need to call pathchk for local libnames */
        lib_lam=libname(libname, path);
        call method('librefs.scl','errmsg',lib_lam,libok);
        put 'LAM/PC Library RC ' lib_lam libok;
    end;
%do;
    %do;
        endmethod;
    %do;
        RETURN;
    %do;
        %*****
        /* Method for checking Error Messages */
        %*****
%K00000:
method liberr % libok %1;

%*****
/* error messages */
/* -70006 or _smolib means library doesn't exist */
/* +-70006 means libref refers to same physical libref as */
/* another library */
/* 630061 means this is the same library that an */
/* is currently running in. */
%*****
if ( liberr = %sysrc(_smolib) or
(liberr > 0 and liberr ^= 70006 and liberr ^= 630061) )
then libok='N';
else libok='Y';
endmethod;

RETURN;

%*****
/* Method for assigning Macro Variables from an SCL list */
/* to multiple platforms. The variable loc is the location */
%*****
MACROVAR:
method listid % loc %1;

%*****
libname=libname(listid);

```

```

macrovarnameitem(listid,1);
if type=type(listid,1);
%do;
    if type='N' then
        macrovar=getitem(listid,1);
        if loc='SAS' then
            submit continue remote;
            data _null_;
                call symputn('&macrovar',&macrovar);
            run;
        endsubmit;
    else
        call symputn(macrovar,&macrovar);
    end;
%do;
    if loc='SAS' then
        submit continue remote;
        data _null_;
            call symputn('&macrovar',&macrovar);
        run;
        endsubmit;
    else
        call symput(macrovar,&macrovar);
    end;
%do;
    endmethod;
%do;
    RETURN;
%do;

```