

SAS® Software: Tools for Managing Your Company

Jerry J. Hosking, SAS Institute Inc., Cary, NC
 Suzanne S. Gordon, SAS Institute Inc., Cary,
 NC

ABSTRACT

SAS Institute prides itself on using its own technology to run its business. Several applications are discussed by the MIS programmers who support them. A variety of SAS software products, SAS/AF®, SAS/FSP®, SAS/CALC®, SAS/ENGLISH®, SAS/SHARE®, IMAGE Extensions to SAS/GRAPH®, and base SAS® software, plus Screen Control Language are showcased by sample applications in use at the Institute. Typical applications are

Application	Description/Product(s) Highlighted
Strategic Online Services	Tracking and resolution of customer problems and requests. Object-oriented programming techniques, SQL Procedure, SAS/AF, SAS/FSP, and SAS/SHARE software
Institute Directory	Data access and visual representation, both as pictures and in hierarchical structures. IMAGE Extensions to SAS/GRAPH and SAS/AF FRAMES
Account Management System	Information delivery concerning sales in progress, market issues, sales forecast, and tracking. SAS/AF and SAS/SHARE software, client/server environment
CPA Application	Financial planning and reporting. SAS/CALC software
Budget Information System	Financial management, analysis, and reporting. PROC REPORT
Human Resources Information System	Managing employee profiles, benefits, compensation analysis, government compliance, and applicant tracking. SAS/ENGLISH software and the SQL Procedure

Discussion includes an overview of functionality of the applications and specific descriptions of ways SAS software products are employed.

INTRODUCTION

The Management Information Systems (MIS) Department at SAS Institute uses exclusively the SAS® System to produce the applications necessary for the business operations of our company. While we sometimes take advantage of features of the operating system software for the platform under which the application runs, the actual tools used to develop and run the application are the wide variety of SAS System products that are also available to you.

Who Is MIS?

The Management Information Systems Department (MIS) in the Information Systems Division has a staff of approximately 35 made up of four applications programming groups and one information center group. The development and support of the applications used for business operations falls to the four

applications programming groups. Although not a part of the organizational structure of the MIS department in Cary, other MIS groups in regional and international offices sometimes share applications developed by each other. With more than 2,200 worldwide users and hundreds of distinct applications, MIS fully exercises all of the SAS software products that make up the SAS System.

What Applications Can Be Written Using the SAS System?

Like any business, SAS Institute needs to manage its business operations such as accounts management, customer services, product distribution, shipping and receiving, accounts payable and receivable, personnel, inventory, project management and tracking, financial planning, and information retrieval and reporting. Although this is not an exhaustive list of all the business operations for a company, it provides a framework to demonstrate the variety of functions within our company that the SAS System fills.

Because our company is a software research and development organization, our MIS staff has a large choice of host platforms from which to select for new development of applications. Currently, we have production business applications running on UNIX, MVS, VM, and VMS operating systems. Factors that determine which platform an application is developed for include

- ☐ access to hardware or equipment for the user group of the application
- ☐ balance of resource use across platforms
- ☐ expertise and choice of the application developer
- ☐ need for communication or integration with previously existing applications.

However, this last criterion has become decreasingly important as the development of cross-domain access has become available in the SAS System. Cross-domain access is illustrated later in the accounts management application.

While this paper cannot address every unique business function or detail all the operations involved in each function, we have selected several applications that support some of the business functions at SAS Institute. For each application, specific features that highlight use of individual SAS software products are described and, where appropriate, implementation details are provided.

PROBLEM/REQUEST/PROJECT TRACKING

We first focus on a problem/request/project tracking application that features the use of object-oriented programming techniques, as well as use of the SQL procedure. In addition, functionality that allows notification of status changes via automatic e-mail is described.

This application, Strategic Online Services (SOS), is used to track customer problems and requests for services provided by the Information Systems Division (ISD). Because ISD is responsible for providing the computing and telecommunications resources to SAS Institute, the customers it serves are actually SAS Institute employees. Some of the features of SOS include the following:

- data entry interface for problem and request recording
- information retrieval for reporting purposes and further action
- status notification to customers and staff responsible for resolution of the problem or request
- assignment of responsibility for problems or requests to ISD staff.

SOS is currently in production on MVS. A development and testing version is maintained on UNIX and when new features are developed, they are implemented on UNIX and then ported to MVS.

Object-Oriented Programming (OOP)

The basic concepts of object oriented programming fit well with the design of this application. In this case, the valid methods are determined for the object (a particular problem or request), and are defined by the class entry in the SAS/AF catalog.

A definite benefit of the OOP technology is the development and use of generic tools (classes) used by SOS and many of our production applications. This eliminates duplication of effort throughout our MIS department. A good example is the COMMENTS class. The comments class is used by SOS to access, enter, and store data in the SAS preview buffer. This is used in the comments for problems, requests, and answers within SOS. Just a few of the available generic tools classes and their methods are listed below.

COMMENTS	Manipulates list of all comments.	
Get	Returns attributes about objects in the comments class.	
Edit	Edits one comment.	
Load	Loads comments associated with a key value.	
MAIL	Manipulates system mail.	
Index	Returns a list of all pending messages for a particular user.	
Read	Returns the selected mail item message.	
Select	Returns the selected mail item number.	
Send	Sends a mail message.	
Qsend	Sends a quick one line message.	
COMMAND	Manipulates all command line activity.	
Get	Returns attributes about objects in the command class.	
Set	Assigns values to attributes about objects in the command class.	
Pull	Pulls the current command from the command line and stores	

it in itself.

Push	Pushes the command in itself to the command line.
Execute	Executes the EXECUTE method for the object specified by the CMDPROCS option.
CMDPROCS	Tracks available commands and how to execute them.
Execute	Executes commands, provided the command was found in the objects list of available commands.
Listall	Returns all available commands.

The basic content of the class entry is a list of methods and where the code that performs those methods can be located by the SAS System. Example 1 shows a sample class entry.

Example 1 Sample Class Entry

```

CLASS(
  _METHODS_=(
    CLOSE='N:PROBLEM' [C]
    ROUTE='N:PROBLEM' [C]
    UPDATE='N:PROBLEM' [C]
    EDIT='N:PROBLEM' [C]
    REOPEN='N:PROBLEM' [C]
    SAVE='O:PROBLEM' [C]
    CONVERT='N:PROBLEM' [C]
    ASSANS='N:PROBLEM' [C]
    FUMIGATE='N:PROBLEM' [C]
    FIND='N:PROBLEM' [C]
    OPEN='O:PROBLEM' [C]
    _INIT_='O:PROBLEM' [C]
    _TERM_='O:PROBLEM' [C]
    INTEREST='N:PROBLEM' [C]
    MAILINTS='N:PROBLEM' [C]
  ) [5] [L]
  _IV_=(
    DESC='problem object' [C]
  ) [3] [L]
  _TYPE_='OBJECT' [P]
  _MODULE_='SASAF' [P]
  _PARENT_='SOS.MASTER.INCIDENT.OBJECT' [C]
  _CURRENT_='PROBLEM' [P]
  _BASE_='SASHELP.FSP.OBJECT' [C]
  _PARENT_='SASHELP.FSP.OBJECT' [C]
) [1]

```

In addition, it is advantageous to create a resource entry in the catalog to identify all the classes for the application. Once used, this resource entry (basically a master list of classes) retains the classes and their methods in memory so as classes are called, they can be quickly accessed rather than necessitating a catalog search. A LOADRES function must be executed early in the application to take advantage of the resource entry. Use of a resource entry increases performance of the application significantly.

For further information on resource entries, see *SAS/AF Software: Frame Entry, Usage and Reference, First Edition*, Chapter 12,

The RESOURCE Entry.

Also consistent with object oriented programming goals, using OOP techniques within the SAS System provides easy ways to share code. Once a method is developed, defining it in the hierarchy for additional objects can be quickly achieved by modification of the class entry.

In our application design, it was determined that the technique most appropriate for communication between objects and methods is the use of SQL lists. Primarily, each method when called expects to receive an input list including the parameters it needs for the processing to occur. In turn, upon completion the method returns a list for the object that results in the action indicated by the user. With this approach, design of methods must be consistent and generic.

For more information on the use of methods, classes, and resources, see *SAS Technical Report P-216, SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07 and SAS/AF Software: FRAME Entry Usage and Reference, Version 6, First Edition.*

Data Retrieval with the SQL Procedure

Data retrieval in SOS is done primarily using the SQL procedure. The user interface for subsetting functionality is a scrollable list from which criteria can be selected that define the data to be retrieved. Once the criteria are selected, WHERE statements are constructed. In turn, code is submitted directly to PROC SQL with WHERE statements to achieve the subsetting indicated by the user. The two primary advantages of using PROC SQL for subsetting are

- its efficiency in indexed searching and, therefore, speed
- its flexibility for combining data sets to retrieve data from multiple sources simultaneously.

Example 2 shows sample PROC SQL code.

Example 2 Sample SQL Code for Problem Subsetting

```
submit sql;
  create view work.vprbsub as
  select distinct int.probno
  from sosda.problems prb,
  sosda.prbinter int
  where prb.probno = int.probno
endsubmit;

/* for each subset element */
do i = 1 to listlen(pcrit);
  crit = nameitem(pcrit,i);
  /* if they want OPEN, look for OPNCLS
  NOT equal to a C */
  if crit = 'OPEN' then do;
    submit sql;
      and ( prb.OPNCLS ne "C" )
    endsubmit;
  end;
  /* if they want CLOSE, look for OPNCLS
  equal to a C */
  else if crit = 'CLOSED' then do;
    submit sql;
      and ( prb.OPNCLS = "C" )
    endsubmit;
  end;
end; /* do to end of list */

/* if user wants data sorted */
```

```
if psort ne _blank_ then do;
  submit sql continue;
    order by psort;
  quit;
endsubmit;
end;
else do;
  submit sql continue;
  ;
  quit;
endsubmit;
end;
```

Electronic Mail Notification

A feature of this application that users have noted as particularly time saving is the generation of status notification messages and the distribution of these messages via e-mail. Of course when users indicate their intent to send a status notification message or the system detects the need to send automatic notification, the problem or request class is invoked. These classes create the message text, determine who to send the message to, then invoke the SEND method of the MAIL class to actually send the e-mail.

While the actual distribution of the e-mail messages is accomplished using operating system commands, noting the existence of this feature reminds us that additional capability can be achieved through the use of the SAS System to generate code that in fact will be executed outside of the SAS System. Example 3 shows sample code for a MAIL method and Example 4 shows sample code for the SEND method of the MAIL class.

Example 3 Sample Code for Mail Method of the Problem Class

```
mail:
  /* GET INFORMATION FROM OPTIONS OBJECT, */
  /* LISTS CREATED USING RC-MAKELIST() */
  probno = getnitemc(_self_, 'PROBNO');
  options = getniteml(getniteml(envlist('L'),
    '_JWPENV_'), 'OPTIONS');
  rc = insertc(optin, 'TEXTFILE');
  rc = insertc(optin, 'EMPNO');
  call send(options, 'GET', optin, optout);
  rc = setnitemc(optin, getnitemc(optout,
    'TEXTFILE'), 'GROUP');
  empno = getnitemc(optout, 'empno');

  /* BUILD FILL FILE NAME USING HOST TOOLS */
  mem = 'E' || empno;
  rc = setnitemc(optin, mem, 'MEMBER');
  rc = clearlist(optout);
  call method('misgen.host.scl',
    'BLDFNAME', optin, optout);
  textfile = getnitemc(optout, 'FILENAME');

  /* BUILD TEXT OF THE MAIL MESSAGE */
  values = getniteml(_self_, 'VALUES');
  rc = setnitemc(tlist, probno, 'PROBNO');
  rc = setniteml(tlist, long, 'TEXT');
  call display(getnitemc(_self_, 'MAIL'),
    tlist);

  /* BUILD ADDRESS LIST */
  link mailit;

  /* NOW MAIL IT USING THE MAIL CLASS */
  rc = setnitemc(mailin, address, 'TO');
  rc = setnitemc(mailin, repto, 'REPLYTO');
```

```
call send(mailobj,'SEND',mailin,mailout,
mailrc);
return;
```

Example 4 Sample Code for Send Method of the Mail Class

```
send: method optional = in out rc 8;
  hostf = getnitemc(in,'HOSTFILE');
  to = "(" || getnitemc(in,'TO') || ")";
  subject = getnitemc(in,'SUBJECT');
  replyto = getnitemc(in,'REPLYTO');
  rc = system("MAIL SEND " || to ||
    " SUBJECT('" || subject ||
    "') FROM('" || hostf || "."') RTO('" ||
    replyto || ")");
endmethod;
```

DATA ACCESS AND DISPLAY

The focus in this example is the Institute Directory (ID). This application provides data access and retrieval facilitated by SAS/AF FRAME entries as well as multi-media displays using IMAGE Extensions to SAS/GRAPH software. It is currently in production on the UNIX platform.

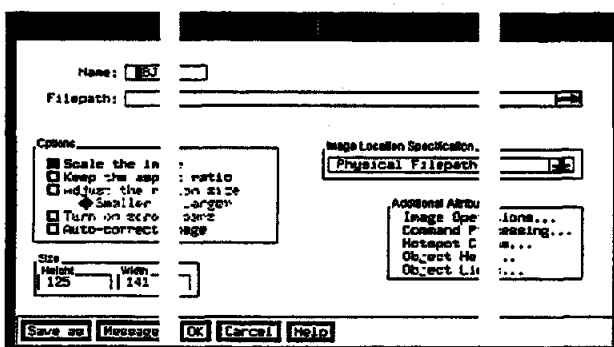
The purpose of the ID is to allow a broad range of users access to business information related to SAS Institute staff. This includes both information on individuals and the organization as a whole. The two features of this application to be addressed here are

- display of visual representations (images)
- hierarchical interpretation and presentation of data (organizational structure).

Other uses for the imaging technology include security applications, inventory tracking systems, purchasing systems, or even training software. The hierarchical presentations can easily be transferred into areas such as cost accounting systems or computer network management applications.

Image Extensions to SAS/GRAPH Software with SAS/AF FRAME Entries

The critical component of the SAS System for this application is the IMAGE widget available in Release 6.09 of the SAS System. The IMAGE widget facilitates the display of many of the standard formats for image files. At instantiation, the user is allowed to specify the location of the images, whether they are stored in an external location or within a SAS catalog. Other options settings are also available in the IMAGE attributes window shown in Display 1.



Display 1 Object Attribute Screen

The example specifies the physical file path to point to the directory where the IMAGE files are stored. Using the LNAMEMK function, we identify the exact image that we want to display in the IMAGE widget. The SCL variable with the same name as the widget is set equal to the LNAMEMK function, thus providing the widget with a packed string that contains path information along with other file information. Example 5 shows sample code for selecting an image file.

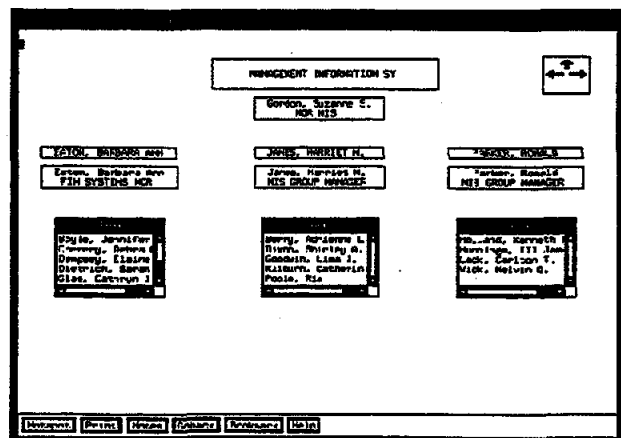
Example 5 Sample Code to Select Image to be Displayed

```
path = '/dept/hr/images/';
filerc = filename('dirpic',path);
picnum=empno;
EMPPIC = lnamemk(4,'dirpic',picnum||'.gif',
'format=gif');
```

In this case the employee number, EMPNO, relates directly with the external IMAGE file name. We are using the gif format, though several other formats are supported by the IMAGE Extensions to SAS/GRAPH software.

Hierarchical Data Display with SAS/EIS Software

Using SAS/EIS software, developers at European Headquarters of SAS Institute created an EIS application to display an organization chart for the company. The code for this application was passed along to MIS in Cary. We have been able to plug in our own data and incorporate the application into our ID. The application uses a combination of FRAME widgets such as control objects and list boxes along with SCL lists to present the employee data in a hierarchical manner. A sample display is shown in Display 2.



Display 2 Hierarchical Display for Employee Data

The organization chart is set up in such a way that when a user clicks on the name of a particular employee, the person's individual information is displayed immediately. This display includes the employee's picture using the IMAGE widget mentioned above.

This application is a great example of the flexibility and portability of SAS software. SAS/EIS software provides organizations with the necessary tools to monitor and manage their company structure. At the same time, applications using SAS/EIS are extremely portable. The only thing necessary to get the European application running at the Cary office was to reorganize the data a bit. No source code changes were necessary.

ACCOUNT MANAGEMENT APPLICATION

The Account Management System (AMS) is a sales accounts management application. Although AMS provides a variety of functions necessary for maintenance of sales data, this discussion features use of cross-domain access to data. Originally, the application was implemented on the VM operating system. To continue to balance computing resource use on different platforms and to take advantage of newer hardware technologies, we migrated AMS to the UNIX operating system. Rather than immediately port the entire application and data to the UNIX platform, we selected the cross-domain server to allow cross-platform access. This permits gradual adaptation to the use of the cross-domain server and permits moving groups of users to the new platform as new hardware is installed.

SAS/SHARE Software and Cross Domain Access

SAS/SHARE software permits simultaneous access and editing of the same SAS data set by multiple users. Control of data access and locking of records being updated is performed by the SAS/SHARE server. The cross domain server is a SAS/SHARE server that allows the same functionality, but between different hosts. At the time of this writing, the cross domain server is an experimental release.

The configuration of AMS after implementation of the cross-domain server includes access to data via three SAS/SHARE servers:

- Data sources needed only by users on VM are available under a SAS/SHARE server on VM.
- Other data sources needed only by users on UNIX are available under a SAS/SHARE server on UNIX.
- Still other data sources required by users on both VM and UNIX are available to all users under a SAS/SHARE (cross-domain) server under UNIX.

In later releases, the SAS/SHARE server under UNIX and the cross-domain server under UNIX will actually be the same server. The code changes required for use of a cross-domain server are minimal. Specifically, they are

- use of the COMAMID= option
- modification of LIBNAME statements to use host file names for the change in location of the data.

Example 6 shows sample code for the OPTIONS and LIBNAME statements.

Example 6 Sample Code for Cross Domain Server

```
options comamid=tcpx;
libname crossdat REMOTE 'path_name'
server=servhost.servname;
```

Performance Results of the Cross-Domain Server

Experience using both native and cross-domain share servers has shown good results in performance of the servers. Respective sizes for two of the data sets currently accessed via the cross-domain server are

- 45 variables making a total length per observation of 1,012 bytes with approximately 7,000 observations
- 24 variables making a total length per observation of 507 bytes with approximately 19,000 observations.

Speed comparisons for access to the data on VM via the UNIX cross-domain server is comparable to access to data via a native VM SAS/SHARE server. Therefore, VM users have suffered no degradation in response time by changing location and access to

the data to UNIX. While access to the data on UNIX via a native UNIX share server is much faster. Therefore, UNIX users have experienced a gain in response time by changing location and access to the data to UNIX.

For further information on SAS/SHARE software and specifically use of the cross domain server, see *UNIX Environments: SAS/SHARE Technical Report, Release 6.09, SAS Technical Report P-260, SAS/SHARE Software for the MVS Environment, Release 6.08, and SAS Technical Report P-261, SAS/SHARE Software for the CMS Environment, Release 6.08.*

ACCESS TO SAS DATA SETS FOR USE WITH SAS/CALC SOFTWARE

This discussion focuses on the CPA Application (CPA), which populates a SAS/CALC software spreadsheet with data in a SAS data set.

Staff of the Finance Division use CPA as a budgeting tool. These users have a moderate level of expertise with SAS/CALC software and are capable of creating their own spreadsheets and straightforward formulas and programs. For the technique of populating the spreadsheet with SAS data values, we examine the functions of the CPA Application that pertain to real estate. The SAS data set contains observations whose variables describe a piece of property. In addition, a secondary SAS data set contains further variables that provide data about the same observation, but are related to tax issues for the property. Although not addressed here in detail, the data entry tool for these data sets uses the FSEDIT procedure and extended tables in SAS/AF software.

SAS/CALC Software Advantages Applied to CPA Application

The primary advantage demonstrated by CPA is the elimination of duplicate data entry. The ability to pull in data from an existing SAS data set allows use of the data in a spreadsheet environment, yet continues to provide the flexibility of access to the data with a variety of other SAS software products.

Techniques for Populating a Spreadsheet with SAS Data Values

Depending on the expertise level of the users, the developer either designs the spreadsheet or uses a spreadsheet designed and provided by the user. The trick to allowing for data to be supplied from a data set in which new observations may have been added is to define boundaries for rows between which the new observations will be added. Then in your calculations, simply perform the calculation from the top row boundary to the bottom row boundary. The spreadsheet is defined in terms of columns and rows, pages, if desired, and the type of data to be in each cell (text, data/numeric, calculated, and so on). For our purposes, the spreadsheet is defined using columns and rows. For further information see *SAS/CALC Software: Usage and Reference Version 6, First Edition*, pp 36-45.

The major steps to fetching data values from a SAS data set are

1. Issue a command to clear all the cells in which automatic (data set supplied) data values will be filled. Be sure to exclude cells the user enters manually. See Example 7 for code for a sample clear command.

Example 7 Sample Code to Clear Spreadsheet Cells

```
/* Pages 182-184 in the SAS/CALC manual discuss
the CLEAR command and its options. This code
section also demonstrates how SCL functions
```

are used in SAS/CALC programs */

```
call execcmd('clear tp01.propname:bp01.proptax
/data;');
call execcmd('clear tp02.propname:bp02.proptax
/data;');
call execcmd('clear tp03.propname:bp03.proptax
/data;');
```

2. Run the command to bring data in. Skip prompting, but specify the data set name making variables columns and observations rows. The ID value is a variable created by combining two key fields. See Example 8 for sample code.

Example 8 Sample Code to Fetch Data and Specify Row and Columns

```
/* Refer to pages 240-260 in the SAS/CALC
manual for the discussion regarding
the FETCH command. The first part of the
discussion concerns Using Dialog Windows
which is followed by the actual FETCH
command syntax beginning on page 256. */
```

```
call execcmd('fetch noprompt data=property
var=col obs=row contype=add
id=propcode;');
```

3. Include code to first clear the program prior to the PAUSE and RECALC commands. Although the instruction to perform the CLEAR is documented (see comment below), it is a critical step and will cause errors if overlooked. The purpose of CLEAR is to prevent re-execution of code in the program buffer when the RECALC command is issued. See Example 9 for sample code.

Example 9 Sample Code to CLEAR, PAUSE, RECALC

```
/* Refer to the following SAS/CALC manual
pages for:
_command_ - pages 460-461
clear      - pages 182-184
recalc     - pages 312-313
pause     - page 519 */
```

```
_command_='clear /pgm;recalc;bottom;';
pause;
```

4. The value of the _COMMAND_ variable is the set of SAS/CALC software commands that are executed when PAUSE is encountered.

In this example, the final step is the display of the spreadsheet filled with data contained in temporary data sets that have been created to supply all the data necessary for the spreadsheet. The display is presented when the user enters the RUN command (with the program name) on the command line or uses an appropriate PMENU to run the program. See Display 3 for a sample of the final spreadsheet.

CALC: PROPERTY PROPSG.CALC (1)
Command menu

	PROPNAME	TAXACT	TAXVALUE	PROPTAX
	Property Name	Tax I.D. Number	Tax Value	Tax Bill
ROU502				
ROU503				
ROU547				
ROU504				
TP01				
P81001	Cary - Building 1	01-9411111	2,000.00	30.00
P81002	Cary - Building 2	01-9422222	10,000.00	100.00
P81003	Cary - Building 3	01-9433333	12,000.00	120.00
BP01				
ROU548				
S1	Total, By Property Location		30,000.00	300.00
TP02				
P82001	Irvine - Building 1	02-9411111	2,000.00	30.00
BP02				
ROU553				
S2	Total, By Property Location		2,000.00	50.00
TP03				
P83001	Aust n - Building 1	03-9411111	2,000.00	30.00
P83002	Aust n - Building 2	03-9422222	10,000.00	100.00
P83003	Aust n - Building 3	03-9433333	12,000.00	120.00
BP03				
ROU549				
S3	Total, By Property Location		30,000.00	300.00
ROU544				
CTOTAL	TOTALS FOR ALL PROPERTIES		62,000.00	650.00
ROU540				

54 10r 04 02-015

Display 3 Final Spreadsheet Created by Pulling Data Values from a SAS Data Set

DATA ACCESS, DISPLAY, AND REPORTING USING THE REPORT PROCEDURE

The Budget Information System (BIS) features the use of PROC REPORT in conjunction with use of WHERE statements and PMENUs.

BIS is used by division management as a budget planning, analysis, and reporting tool. Initialization of the application is the same for each user. However, the specific data accessed, and therefore displayed, is unique for each division and is automatically selected by code in the invocation of the application. The basic design of the application is the presentation of summary budget versus actual information that permits drill-down actions to gain access to more levels of detail, all the way to investigation of individual expenditures and payments. For example, within a division, the user begins by displaying summary figures for planned and actual expenditures for departments. By selecting departments or categories of expenditures (supplies, travel, furniture, and so on), breakdowns of the summary figures are displayed. See Display 4 for a sample summary display and Display 5 for drill-down results to specific budgeting categories.

Drill Down | Drill Up | End

Item	Description	Budget	Actual
		====	=====
		400	500
		=====	=====
Value selected = Division ISD			
ADM	Administration	100	100
HSD	Hardware Support	100	100
MIS	Management Info Systems	100	200 *
PUR	Purchasing	100	100

SAS MIN *Cr 17/006

Display 4 Sample Display of Summary Division Data

Drill Values Selected:

Division: ISD
Department: MIS
Account: 1303

Detail From Accounts Payable (AP) Information System:

Vendor: International Airways
PO Number: TR56960
Approved by: Suzanne Gordon
Item Description: Airfare to Dallas for Barbara Eaton

Invoice: 50884245
Invoice Date: 01FEB94
Paid Date: 15FEB94
Check number: 155890
Amount: 125.00

SAS *Cr 02/003

Display 6 Sample Display of Individual Expenditure

Drill Down | Drill Up | End

Item	Description	Budget	Actual
		=====	=====
		100	200
		=====	=====
Values selected = Division ISD Dept MIS			
1303	Travel	25	125 *
1404	Supplies	25	25
4908	Furniture	25	25
7399	Training	25	25

SAS *Cr 06/005

Display 5 Sample Display after Drill Down to Categories (Account Codes)

To expand the functions available to users, use of SAS/AF software within PROC REPORT is possible. For example, the individual expenditures data, shown in Display 6, is accessed by a call to FSEDIT from a SAS/AF program entry that is defined in the PROC REPORT program screen of the drill variable.

PROC REPORT Features

The interactive capabilities of PROC REPORT are particularly useful in presenting a tool that allows immediate, real-time reporting on budgeting actions taken (or proposed) by users. The report displays are easy to design, and because the user interface is a part of PROC REPORT, there is no development time actually spent on coding to create such functionality. It is not necessary to design SAS/AF program entries and accompanying SCL because the user is interacting directly with PROC REPORT. Also because PROC REPORT was designed for printing displayed data, capability for printing is good and again it is not necessary to develop printing routines.

Budget Information System Methods

The method used to display and then redisplay data based on selections of the user is an initial invocation of PROC REPORT followed by submission of additional PROC REPORT statements based on actions taken by the user. See Example 10 for sample PROC REPORT code to begin the display.

Example 10 Initial Invocation of PROC REPORT

```
/* AF SCL Code to invoke initial PROC REPORT */

INIT:
RETURN;

MAIN:
  SUBMIT IMMEDIATE;
    proc report data=budget.data
      (where=(division='ISD'))
      report=preport.report.findept
      profile=preport.report.norkeys;

  run;
  ENDSUBMIT;
RETURN;

TERM:
RETURN;
```

Variables in the PROC REPORT display are acted upon as they are defined in the COMPUTE statement. When the user selects a variable, by highlighting its location on the screen and clicking on

the drill-down action, PROC REPORT executes the code following the COMPUTE statement for that variable. The power to select the level of detail indicated by the user is possible through the use of WHERE statements.

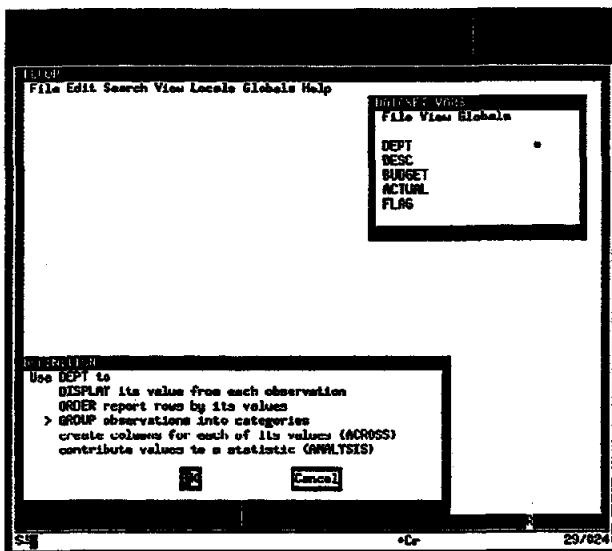
Budget Information System Development

The steps taken in development in the Budget Information System are

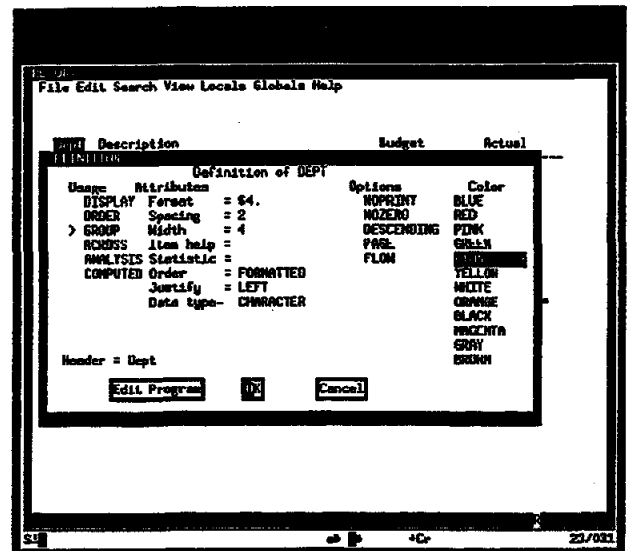
1. interactively executing PROC REPORT using the PROMPT option. See Example 11 for sample code.

Example 11 Using PROMPT Option with PROC REPORT

- ```
proc report data=budget.data prompt; run;
```
2. defining the characteristics of all variables to be used. This can be done through a series of prompting screens (initial screen shown in Display 7) or a single PROC REPORT define screen (Display 8).



Display 7 Defining Variable Budget via Prompt Method (Initial Screen 1)



Display 8 Defining Variable Budget by Selecting DEFINE in PROC REPORT

3. establishing what is to be displayed as a result of a drill-down action on a variable using the RLOAD statement. This RLOAD statement is entered by clicking on the Edit Program button on the PROC REPORT DEFINE window for the variable DEPT. See Example 12 for a sample RLOAD statement.

#### Example 12 Sample RLOAD Statement

```
call define(_col_, 'command',
 'rload preport.report.finacct;
 where also dept="' || trim(dept) || '"');
```

4. creating PMENUs to provide an easy mechanism to request drill down or up actions. Drill-down actions execute RLOAD statements previously defined and drill-up actions end the previous action by undoing a WHERE and refreshing the display. See Example 13 for sample code for creation of a PMENU.

#### Example 13 Sample Code for Creating PMENUs

```
/* Create PMENUs for PROC REPORT application. */

proc pmenu lib=preport.report;
 menu finmenu;
 item 'Drill Down | ' selection=drilldn;
 item 'Drill Up | ' selection=drillup;
 item 'End' selection=quit;

 selection drilldn 'EXECUTE';
 selection drillup 'END' WHERE UNDO; REFRESH';
 selection quit 'QUIT';

run;

quit;
run;
```

### DATA RETRIEVAL AND DISPLAY USING SAS/ENGLISH SOFTWARE AND THE SQL PROCEDURE

The Human Resources Information System (HRIS) demonstrates data retrieval using SAS/ENGLISH software and the Structured

**Query Language (SQL).**

This application enables the Human Resources staff to analyze and report information concerning the employees of SAS Institute. The design of HRIS employs a user interface that enables the user to retrieve information through natural language queries. Results can be displayed or printed.

**SAS/ENGLISH Software and SQL Requirements and Features**

Like PROC REPORT, the users interact directly with the procedure, so it is not necessary to spend development efforts on designing or controlling the user interface screens. These screens are the interface methods built into SAS/ENGLISH software.

**Human Resources Information System Development**

A major component of the development of an application that performs data retrieval using SAS/ENGLISH software is the definition of the knowledge base. The SAS/ENGLISH knowledge based defines objects (data sets), attributes (variables), and relationships between sets of objects.

The steps in developing the SAS/ENGLISH knowledge base are

1. prepare the data in normalized form
2. determine the object represented by each data set
3. establish relationships between objects and their attributes
4. define the vocabulary
5. define relationships among objects (if multiple objects are used).

**Prepare the Data in Normalized Form**

SAS/ENGLISH software requires that the data sets it uses be in normalized form. These data sets and their attributes are registered to SAS/ENGLISH software in a knowledge base. Some effort during development to restructure existing data may be necessary. However, the creation of normalized data sets can in fact save space due to a less repetitive method for storing data. The concept of data stored in normalized form is the creation of data such that multiple observations exist for multiple values of a variable for the same key field. For example, in a typical SAS data structure concerning employees and the types of insurance they hold, you would see a key field variable (employee number or name) along with a variable for each type of insurance available (health, disability, life) and value of those insurance variables to show whether that employee (key field for the observation) has that type insurance (such as 1 or Y=yes, 0 or N=no). In a normalized data structure, a separate observation with a key field variable (again employee name or number) and a single variable (called INSURE) with actual values that represent the type of insurance held (such as 1 or H=health, 2 or D=disability, and 3 or L=life). Then multiple observations may be present for the same key field variable to represent holding of multiple types of insurance. If no insurance is held by the person described by the key field variable, then no observations are present in the data set. Compare Table 1 with Tables 2 and 3 to contrast data for typical and normalized data structures.

| EMPNO | HEALTH | DISABLE | LIFE | SALARY | SEX | ... |
|-------|--------|---------|------|--------|-----|-----|
| 1     | Y      | N       | N    | 100000 | M   | ... |
| 2     | Y      | N       | Y    | 120000 | F   | ... |

| EMPNO | HEALTH | DISABLE | LIFE | SALARY | SEX | ... |
|-------|--------|---------|------|--------|-----|-----|
| 3     | N      | N       | N    | 110000 | F   | ... |
| 4     | Y      | Y       | Y    | 115000 | M   | ... |
| 5     | N      | Y       | Y    | 110000 | M   | ... |

**Table 1 Typical Data Structure**

| EMPNO | INSURE |
|-------|--------|
| 1     | H      |
| 2     | H      |
| 2     | L      |
| 4     | H      |
| 4     | D      |
| 4     | L      |
| 5     | D      |
| 5     | L      |

**Table 2 Normalized Data Structure (Insurance Plan Data Set)**

| EMPNO | SEX | SALARY | ... |
|-------|-----|--------|-----|
| 1     | M   | 100000 | ... |
| 2     | F   | 120000 | ... |
| 3     | F   | 110000 | ... |
| 4     | M   | 115000 | ... |
| 5     | M   | 110000 | ... |

**Table 3 Normalized Data Structure (Employee Data Set)**

In addition to the normalized data structure, data are stored in a larger number of smaller data sets that describe specific characteristics of the data rather than a smaller number of larger data sets that describe all characteristics of the data. This results in savings in size of data storage due to eliminating observations for key field variables that would not necessarily be present for each characteristic.

**Determine the Object Represented by Each Data Set**

The process of defining the object is the design phase of the project. Consider what the object (data set) you are creating will represent. Identify the specific thing that the attributes (variables) of the object will describe. Once identified, consider the information that will be used to describe that object. These become your attributes.

To continue to use the example described above, consider the

resulting objects. Rather than viewing the employee data set as a collection of personal data, develop the idea of an employee object in which sex, salary and other variables are attributes describing that object.

#### Establish Relationships between Objects and its Attributes

The first task in defining the relationships between objects and its attributes is establishing the type of attribute. Using SAS/ENGLISH knowledge base define panels, the developer responds to prompts to define relationships. SAS/ENGLISH software provides for several attribute types, and the particular values of the attribute will determine its type. We'll discuss three attributes for the employee object and the attribute type assigned to them:

1. empno (employee number)
2. sex
3. salary

*Employee number* is the unique identifier (key field) for the employee object. The attribute type for this is called an identifier. It is critical that an identifier attribute be a unique value. *Sex* is a classifier attribute. This type attribute simply describes a characteristic about the object. *Salary* is a measurement attribute. The distinction of a measurement attribute is that it is quantifiable and can be used for calculations.

To continue defining the relationships, describe the verbs that can apply to the attributes you have established. Again, use SAS/ENGLISH knowledge base define panels to register applicable verbs. See Figure 1 for a visual representation of the EMPLOYEE object and its verbs.

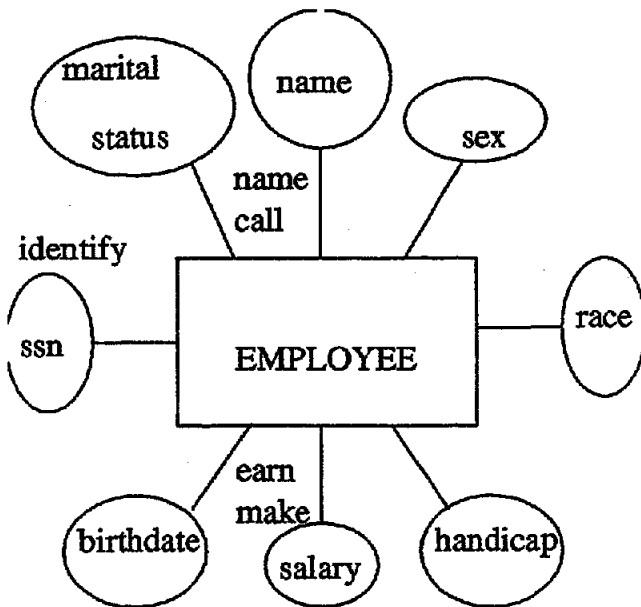


Figure 1 Verbs for Employee Object

#### Defining the Vocabulary

Continue to use the SAS/ENGLISH knowledge base define panels to name synonymous terms for the attributes and values of attributes that users might use in their queries. This can actually be supplied from real data values rather than entered by the developer. In addition, synonyms can be supplied for the object itself. See Figure 2 for a complete sample of the employee object, its attributes, verbs, and vocabulary.

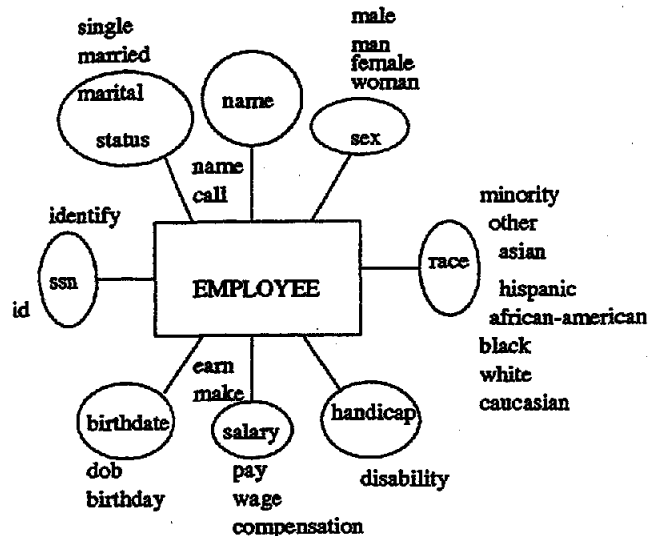


Figure 2 Complete Employee Object Representation

#### Defining Relationships between Objects

Thinking of relationships as defined by ownership or parent/child can explain how relationships between objects are defined. For example, an employee object is the parent of the insurance plan object. A critical factor in determining relationships between objects is use of the same identifier attribute in the objects. Similar to the process of defining relationships between an object and an attribute using a verb, relationships between objects can be described by defining verbs. All of this information is contained in the knowledge base and is supplied using SAS/ENGLISH knowledge-base panels. See Figure 3 for a representation of a relationship between objects.

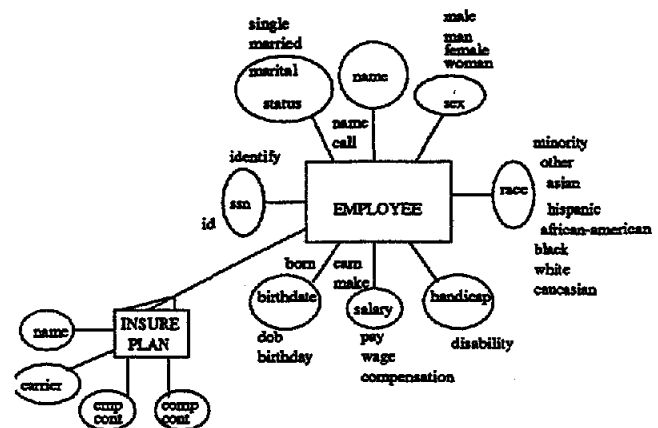
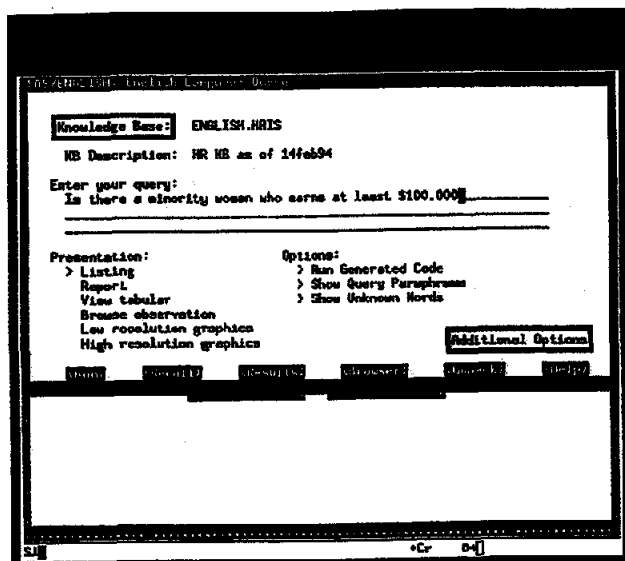


Figure 3 Relationship between Employee and Insurance Plan Objects

Execution of the application presents a SAS/ENGLISH knowledge base query panel on which the user enters the natural language query or statement that describes the subset to be presented. See Display 9 for a sample query.



**Display 9 HRIS Query**

SAS/ENGLISH software automatically translates the user's query into SQL commands.

## CONCLUSIONS

This paper has presented only a sampling of applications using a variety of techniques and SAS software products. These applications discussed here are all real applications used daily at SAS Institute. But these are only the tip of the iceberg. In addition to developing and supporting the applications necessary to perform the business functions of SAS Institute, the MIS Department continues to explore the range of capabilities provided by new and enhanced SAS System products. These new features are adopted for new or integrated into existing applications so that our users can benefit from the newest technology available. We operate our company with the SAS System. We hope we have given you ideas so that you can too.

This paper is a collaborative effort by staff of the Management Information Systems Department. Application programmers who contributed to this paper by sharing information about the applications they have developed and support are

- Teresa L. Arthur & Caroline C. Quinn - Strategic Online Services (SOS)
- Shirley A. Dixon - CPA Application (CPA)
- Elizabeth L. Hill - Institute Directory (ID)
- Laura A. Lehman - Human Resources Information System (HRIS)
- Deborah P. Morrison - Budget Information System (BIS)
- Ria Poole - Account Management System (AMS).

SUGI presenters from MIS concerned with the content of this paper are

- Suzanne S. Gordon - Manager, MIS
- Harriet M. Janes - MIS Group Manager
- Laura A. Lehman - Programmer Analyst
- Caroline C. Quinn - MIS Group Manager.

SAS, SAS/AF, SAS/CALC, SAS/ENGLISH, SAS/FSP, SAS/GRAPH, SAS/IMAGE, and SAS/SHARE are registered trademarks or trademarks of SAS Institute Inc. in the USA and

other countries. \* indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.