1.0 INTRODUCTION:
A classic SAS® application environment has several component entities, as for example: SAS programs; SAS-Datasets and or Data Views in one or more Data Libraries; Catalogs for User Defined Formats; and possibly SAS/AF® application modules. In addition to these “objects”, there are the processes that alter or use these objects, such as the running of software either in real-time or in background mode (“batch”). Making sense of, and gaining control over, a complex application can be a challenge.

The body of this paper suggests some ideas -- both in the way of special application management software and in the way of procedural standards -- that can assist in the management of a medium sized application area. The ideas presented reflect an approach used in The Hartford’s Reserving Area and is based on experience over the past four years managing this moderate-sized, but growing, SAS application environment. The application area is in support of the property-casualty business reserving function at The Hartford. A staff of three maintains the applications.

Although the context of the ideas happens to be in a text-based (non-Windows) interface with SAS on a COMPAQ (formerly DEC) Alpha mid-tier, many (if not most) of the concepts would seem to be relevant to -- and transferable to -- functions and procedures serving to manage applications in WINDOWS environments.

Topics covered will include:
- Program nomenclature, structure and commenting standards;
- An automatically self-documenting “program manager”;
- File labeling standards;
- Format Catalog update standards;
- Reliance on SAS Macro; and
- Use of SAS-AF to manage production, access data and assist in managing the application.

For those not acquainted with the Alpha environment, VMS uses a directory-based file architecture, and the Alpha functions as both a mid-tier and also a LAN server by way of a product called Pathworks®. Non-proprietary files, like text files and SAS programs, are thereby directory accessible both from the PC Windows environment and also from the DEC’s VMS perspective.

2.0 WHERE WE CAME FROM:
As of early 1996, when the staff of the area increased from two to three persons, the Reserving Systems consisted mostly of FORTRAN and PL-1 programs, although a couple of SAS prototype replacements for the 3-GL software had been implemented. Programs were built and changed in LSE (“Language Sensitive Editor”, the DEC’s loose analogy to CMS XEDIT or TSO FSE). Although the DEC LSE had facilities for real-time debugging of programs (analogous to the old IBM PL-1 debugging compiler), the tendency was to test most programs in batch runs using com-files (“Command Files”). These last, written in DCL, are analogous to mainframe JCL, CLISTS or REXX.

Most of the Reserving production was -- and is -- on a monthly basis. In the legacy environment, programs read external data sources and set up files in the Reserving area. Most of the Reserving files were Fortran Style direct access (or “Regional 1”, from the PL-1 perspective). They were horrid to make sense of since they lacked explicit record keys. Against these, other programs ran models to analyze the data, producing hardcopy style reports to disk. Still others did various kinds of reporting directly on the source data itself.

Ad hoc reporting was usually handled in the following manner. A customer would ask Reserving Automation Support personnel for a particular report. This latter individual would then submit the appropriate job, which then wrote the report out to a common area on DASD. The user could then access it via PATHWORKS and either print it or (later) parse the information into Excel® or Lotus®. Generally, users spent a lot of time manually entering data into spreadsheets.

The FORTRAN and PL-1 programs were parameter driven (in the sense that they read a parameter file). But, for run purposes, the line of least resistance was to set up individual com-files for each of the more common run variations of the reporting programs wherein the parameters for the run were entered in stream in the com-file DCL. It was up to the systems person who set the com-files up to keep track of them.

System activation and management was thus heavily a matter of manual processes. Any accessible documentation was totally separate from the actual application software (programs), the com-files, and the data files. A “Production Document” was essential.

3.0 WHERE WE WENT:
Over the next three years, the Reserving Systems Support area replaced the entire body of what constituted the application. All the FORTRAN and PL-1 pro-
grams became SAS programs, and the data and model results were migrated to SAS Datasets in the process. Using traditional Version 6.0 SAS/AF, a new and totally comprehensive application driver called RESMENU was set up to initiate all the various processes in Reserving applications. This structure permits Automation staff to run production -- which updates data and model results -- and all customers to activate the reporting functions. In their SAS reincarnation, the Reserving "models" were modularized into stages to calculate results, and the reporting components of the model functions were isolated into separate report generating SAS programs.

Thus the manner in which the reporting functions were carried out radically changed. The new SAS/AF application driver RESMENU presents the users with a menu structure that the user can migrate through to select an area of interest and then run reports within the area. The inflexible legacy-era reports have given way to new versions with more variations. The end-user employs RESMENU's SAS/AF PROGRAM-entropy screens to capture parameters -- such as lists of lines of business -- and pass them into the executed SAS program.

In addition to creating reports to the Output Window (for example, using PROC REPORT or TABULATE), the current report programs produce (as an option) files that can be directly uploaded into spreadsheets without parsing. At first these special files for spreadsheets were blank separated "file-import-numbers" Lotus-style 'prn' files; these could be quickly read directly into Lotus. But, as Excel displaced Lotus as the tool of choice, the SAS report programs have shifted to generating "csv" files instead. Manual data entry has been totally eliminated.

The rewritten application environment does retain the com-files for long-running production functions triggered by Reserving Automation Support personnel; otherwise com-file usage has largely been dispensed with. Since the turnaround time for most reports is short, the current reporting functions execute on-line.

In the last stage of the application rewrite, there was a change over to far richer data sources. These offered "drill down" opportunities previously unavailable in the legacy data sources used by the original system. This opportunity has come at the cost both of a more complex production cycle and of much, much larger and more numerous data files. For access purposes, indexing was introduced to reduce report run times.
4.0 RESMENU: WHAT IT IS

We begin our Housekeeping tour with what we as Systems Support see and what our customers see as the mechanism for getting things done. On the previous page is a display of the main menu of RESMENU, showing the users’ access path to one of the end-user applications, called “Flexible Calendar Reports”.

You will note that the SAS/AF we are using is not Windows based. This was a circumstance forced on the developers (i.e. on the author and those who worked with him). A Windows interface was not an option until this year. Within the limitations of this text-based environment, the developers have endeavored to make the system as much point-and-click as possible.

The menus, such as the two shown on the previous page, are actually “extended list” SAS/AF Program entries. The user selects what he/she wants by putting the cursor on the selection and hitting <enter>. The selection, either the next menu down or the application selected, then appears. (When the current AF application is brought up under Windows/Motif, double-clicking with the mouse on the selection effects the same action.) The report selection screens make extensive use of choice groups and pop-lists for such things as line of business selections. Three of the choice groups are outlined on the interface screen exhibit for “Flexible Calendar Reports” on the previous page. “Hot-buttons” activate the pop-up lists, and arrows on the exhibit highlight two of these: one for a list of sub-lines desired (“Lin:”) and one for a list of variates or financial statistics (“Item List”). The hot buttons actually constitute a hidden choice group on the screen.

The pop-up selection lists are also “extended lists”. Here, as with the menus, the user simply points with the cursor to select the item(s) on the list he/she wants. Selecting a highlighted item unselects it.

The emphasis is to minimize the amount of typed input from the user and to make it as easy as possible -- within the limitations of the environment -- for the user to easily select what he/she wants. To achieve this objective, cosmetic gimmicks are important. The “hot buttons” activating the pop-up lists blink. The “fill in” areas for text input are displayed in reverse video. Judicious use is made of the limited availability of colour. The text-based terminal emulator we currently use is PowerTerm. It permits different types of text -- like highlighted, blinking and reverse-video -- to be assigned different colours. For example, on RESMENU the “hot buttons” on most screens blink in red.

The VMS and network environments we use have changed during this past year or two. As I write, Windows/Motif is now generally available to all users on our mid-tier, and the Intranet is the direction of user data access. Clearly it is to our interest to migrate out of RESMENU’s current legacy non-windows SAS/AF interface and toward a WEB based interface. This we will eventually do. The tools available in the GUI environment are richer for purposes of setting up an interface like RESMENU, and the use of the Intranet potentially eliminates the requirement that a RESMENU user have an ID on the COMPAQ Alpha mid-tier. This having been said, however, the migration to a GUI interface does not mean that the need to use intelligent screen design takes a holiday.

5.0 A PICTURE OF THE APPLICATION TOOL SET:

Above is one of those silly schematics that attempt to communicate significant information using ten cents worth of effort. The picture identifies our application’s components; these are at the same time both the basic building blocks of the application and also the source of the management problem for the staff. As developers, our intent is first to make the application as immune to maintenance as possible and then, secondly, to render the application as self-documenting as possible.

The arrows on the “ten-cent” diagram represent connections or references that underlie documentation considerations. To this end the significance of the direction of the arrow is important: the entity at the arrow-head is a useful piece of knowledge for the entity at the butt-head end. How does this work? Consider the arrow from the SAS programs to the SAS Datasets. It says that knowing what program made the dataset is a useful piece of information.

An example might help. Take a look at the execution
screen for the “Flexible Calendar Reports”, reproduced here. The executing (submitted) program name appears on the screen, SB0520RP, as is the name of the AF entry presenting the screen, also SB0520RP. This is clearly not for the end-users’ benefit; he/she could give a happy ding-dang what SAS program is providing the good stuff. BUT it is important for the developers and maintainers of the set-up.

Now the executing (submitted) SAS Program can be most assuredly unearthed inside the SCL-code of the AF-Program presenting the screen shown. And, yes, this SCL code is accessible through a CATALOG or a BUILD command in the classic SAS/AF. However the

ploy used removes the necessity to go to this trouble. Just bring up the menu and you know right away what the SAS program is, if maintenance is required. Implementing the gimmick is just a matter of noting the executing SAS program on the screen when it is built.

There are other pieces of documentation on the screen, for example the program library, which has the SAS Filename “RESRVPGM”. In Reserving Systems, programs happen to occupy unique directories (rather than source code catalogs), and this SAS Filename points to the directory. “RESRVSRC” is a SAS Libname for the data library where the source data for the reports is stored. The current production date cycle of the data is also shown. This last item is clearly of interest both to the system customers and to the maintainers of the system.

The screens thus have a second purpose beyond allowing the users to do things. They help the maintainers maintain things.

6.0 RESMENU’S OTHER PURPOSE: RUNNING PRODUCTION:

On the following page is something not all application areas can -- or might want to -- buy into. But for us it fits like a glove. Why limit the use of your application driver to making life easy for the end-user? Use it to make your own life easy.

Many traditional 3-GL environments rely heavily on so-called “production documents”. (The Reserving area had one and still does.) These specify what jobs are to be run and how the various jobs are to be submitted. In the mainframe environment -- and also on VMS -- jobs can also be set up using an automatic scheduler to run, say, a certain job on a certain day of the month. Job dependencies can also be set up so that job X won’t run until job Y completes its business successfully. What is presented on the next page is a more informal alternative, but an alternative that is still a structured approach to the same problem.

The diagram on the following page is another access sequence, this time beginning with the sub-menu that pops up when the “SB-Update” option on the RES-MENU main menu is selected. “SB-Update” is the production that rebuilds the data files (data mart) used by our customers and the Reserving models.

The first menu shows the main elements of the production sequence -- in the order in which they can be executed without destroying job dependencies. The first “RUN PRODUCTION” option updates any mapping tables that we need to generate our line of business identification structures. Access to data generated in other areas requires that these mappings be up-to-date. The bulk of the data -- the so-called loss
development triangles -- is built with the third “RUN PRODUCTION” option, Option 5.0 “Monthly XB2020-based Loss Jobs”. The NADB and Large Loss options create data adjustments that the models use to remove anomalies that would otherwise hurt the model projections. The final option inputs some miscellaneous financial information, such as premium income, lying outside the scope of the loss data.

If you select option 5.0 you get the second menu. This one lays out the job sequence for re-creation of the loss triangle data mart, in order of dependency. The first step on the list extracts data from our ORACLE source files. The second begins the process of rebuilding the Reserving Triangle Data Mart.

When Option 2.0 on this second menu is activated, the person running production is now presented with the third screen down. We should explain that we are not currently using a formal warehouse tool (like SAS/MDB). For this reason and for ease of update and access, the Reserving data is partitioned into physical datasets, aligned by groupings of lines of business. We refer to these as the “Production Legs”. These legs can be recreated individually, or all legs in the list can be launched all together (first option). One can choose to run the recreation on-line (which one might do, if one is testing out a program change) or one can use a batch submission. If the latter, one can choose the batch queue.

If we select the Homeowners’ run, we will finally arrive at the screen shown on the next page. This is what we call a “Go-Gogo” screen: it captures no parameters but permits the user to see what he/she is launching prior to releasing the job. It also gives the individual a chance to change their mind, in case (for example) they meant to choose “batch” and chose “on-line” in error.

Up to this point, the screens repeat a lot of the information previously noted: the name of the AF-programs whose screens we are passing through and the names of the various libraries for programs, tables, actual data and com-files. The “Go-Gogo” screen, however, has some different kinds information, shaded for clarity on the next page:

- A clear notation as to the leg (line of business group) being run.
- A clear notation as to whether the process will be batch or on-line.

- The name of the SAS program which will be executed.
- The name of the Com-file which will be launched if the process is run in batch. This will invoke SAS and Execute the program.
- The names of the VMS Log (created by the COM-File and the SAS-Log file, created by the Batch SAS execution. Both of these will be on the activa-
A TYPICAL RESMENU PRODUCTION “GO-NOGO” SCREEN

+SB-UPDATE - A3-3-SB0005EX

************************************************************************************

XB2020 DATA DEVELOPMENT
ACCIDENT MONTH TRIANGLES -- LEGS 0 THRU 9

************************************************************************************

Prod-Date: 2000-05  SAS-libname: RESRVSRC  Pgm-library: RESRVPGM
Base-Year: 1974  Work-Lib...: RESRVWRK  =========<  Submit Job or Cancel >=====

Production Leg: 0  Homeowners' Multi-Peril and Pers Fire...
Run-Mode...: Batch Submission  Debug-Run: NO  Program-Used: SB0005PC
Batch Queue: RES$BATCH  SASLOG.File.: XB2020K0
Comfile.: RESRVCOM:SB0005EX
VMS Log.: SB05EX0  Parm-Str: /PAR=(*0,200005,*CREATE*)

Hit <pf3> to Execute -----------------------------  Hit <pf4> to Cancel

******************************************************************************

A TYPICAL RESMENU PRODUCTION “GO-NOGO” SCREEN

Tor’s home drive, with a “.log” extension, accessible in both the LAN and VMS perspectives of the COMPAQ Alpha.

- A depiction of the parameter string fed to the VMS com-file.

7.0 WHAT MADE THIS? TYING DATA AND CATALOG ENTRIES TO SOFTWARE:

One issue in an area managing a large number of files is the problem of keeping track of the files. Reserving is a firm believer in the use of SAS Dataset Labels on SAS Datasets and descriptive labels on catalog entries like SAS Formats. Reserving uses many user-defined SAS Formats of the Value replacement kind; some provide titles or descriptions for codes, but the heaviest use of this type of Format is as a mapping, as an alternative to DATA-Step MERGE’s and PROC SQL joins.

Clearly SAS Dataset-naming conventions can help in the management process. In the little fragment of a library DIR-listing below, you can see how the stem of the “Large Loss Adjustment” files indicate, by way of mnemonic, the nature of the data on the file. Using a common stem as the beginning of the dataset name also tends to force files having a common purpose to cluster together a simple alphabetic DIR listing.

Now look at the SAS Dataset Labels. The creating program has put its name on the first eight positions of the file label and has also nicely deposited the production cycle date on the label. Thus the data set label information ties the data back to the software that creates the data. (This notation of the program name is an alternative to file naming standards employed in some main-frame computer shops, wherein the file name always carries the first six characters of the generating program name. Thus, files produced by a program named SB7215PC might be called SB7215C1, SB7215C2 and so on.)

A similar thing can be done to SAS Format Catalogs. See the CAT-Listing display at the top of the next page. Note that the sample format descriptions here also have the creating program’s name. Although, in the Reserving RESMENU application, production cycles are not pertinent to the SAS Formats, another consideration is. Many of the user-defined SAS Formats are generated from tables in the form of SAS Datasets, using a DATA-Step and PROC FORMAT with the CNTLIN option. Clearly it is important to know what that source table is, and that too is documented, where

<table>
<thead>
<tr>
<th>Libref: RESCATFX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: ALL</td>
</tr>
<tr>
<td>SAS File</td>
</tr>
<tr>
<td>LRGLMF05</td>
</tr>
<tr>
<td>LRGLSS05</td>
</tr>
<tr>
<td>LRGLSSRM</td>
</tr>
<tr>
<td>LRGLSSRY</td>
</tr>
<tr>
<td>XB2020C0</td>
</tr>
<tr>
<td>XB2020M0</td>
</tr>
</tbody>
</table>
Putting SAS Labels on datasets is a no-brainer. Both DATA-Steps and Procedures, like PROC MEANS, allow LABEL= options on the statement producing the output file. Putting the production date on as part of the label requires only that you have a macro variable hanging around carrying the date and that you use double quotes embracing the label value. The macro variable can be a GLOBAL variable generated in the autoexec (sasexec), or it can be a parameter to the executing program, if the SAS program is in the form of a SAS Macro. And ALL of RESMENUS's SAS Programs are Macros -- no exception. We talk about this in the next section.

Putting labels on the Format Catalog entries is a little less obvious. You can enter such information manually in the CATALOG window. However, when the SAS Format is regenerated to the Format Catalog, this label will be destroyed, requiring you to re-enter the label. There is an alternative device documented in a Coders' Corner paper by Jack Shoemaker in the 1998 NESUG Proceedings. As the last step in the SAS program that regenerates the SAS Format, add a PROC CATALOG step with a series of MODIFY statements, as per

```sas
PROC CATALOG CATALOG=fmtlib..FORMATS;
MODIFY CRMCSUF.FORMAT
  (DESCRIPTION="SB0044FV-Table G: Claim Resrv M.C. Descr 11/29/99");
MODIFY CRMNAME.FORMAT
  (DESCRIPTION="SB0044FV-Table G: C.R.M.C. Description 12/01/99");
MODIFY YRMON.FORMAT
  (DESCRIPTION="SB0099FM: pic'YYYY-MM' Numeric Format 11/29/99");
MODIFY CSCRMC.FORMAT
  (DESCRIPTION="SB0045FV-Table H: ClmSym Suffix Over 12/01/99");
MODIFY CSYNAM.FORMAT
  (DESCRIPTION="SB0046FV-Table K: Claim Symbol Descrip 11/29/99");
RUN;
QUIT;
```

### 8.0 USE OF MACROS - 1: PARAMETER DRIVEN-

**SAS CODE:**

This is not a paper on SAS Macros, but macros are a critical part of the way in which Reserving keeps house.

As alluded to in the preceding section, all of our programs are in the form of SAS Macros. The most immediate reason for programs to be in such a form is that there are certain driving parameters that will vary from use to use. Clearly production date is one.

At the top of the next page is the %MACRO statement of SB0005PC, the program submitted by the “Go-Nogo” SAS/AF screen depicted a few pages back. This is a key program in the sequence that builds our loss triangle “data mart” by filling out the line of business consolidations in the accident month level loss triangle data. The macro invocation involves quite a few keyword parameters -- what is their purpose?

The production date PDATE is sort of obvious, and yet why did we make it a parameter rather than a global macro variable reference? The reason is that in test mode we can easily back-run without resetting our “globals” (i.e. global macro variables, like production date). The FILEID parameter specifies the production leg being run against. The Base Year (BYR) parameter specifies the first accident year in our data. Test files might differ from production data in this regard. The SAS Libnames of the data libraries, from which data comes and to which the created files go, are also parameters -- see LIBNAME, WORKLIB and OUTLIB. Now we could have hard-coded SAS Libnames in the program SB0005PC and simply redefined them at usage time. But passing the names in as parameters allows for us to have multiple libraries accessible by the same program without redefining the libname’s. It also permits busting up data libraries without impacting the accessing software.

The BYVARS, IDVARS and SUMVAR1-SUMVAR4 look suspiciously like SAS PROC-speak. Each is a list of SAS Variable names. What these parameters permit is the alteration to the scope and sequence of the class and analysis variables used by the program. Most of what SB0005PC does, on the way to building the line of business consolidations, is to extract data, rename it, sort it and then sum it across records with like sort-break “by-variable” values. There are many repetitions of this sequence in SB0005PC. In early versions of this program, a major focus of maintenance involved changes in the class variable list and changes in the variates to be included in the summation process.

The solution path was to feed in the currently used lists as macro variable parameters, as shown here. When the list is expanded or the application is used against data lacking one or another of the class variables or variates, the program can be applied without manual cloning.
9.0 USE OF MACROS - 2: REUSABLE CODE
(“UTILITY MODULES”):

The kind of processing repetitions cited in the discussion of SB0005PC spawned a small number of Macros to perform these repeated functions in other programs. One such module was set up to sort and summarise a SAS Dataset. Sometimes the source file would be too large to sort as a single PROC SORT. Then it would be necessary to breakup the file into pieces, sort the pieces, and then remerge the sorted components. This series of computations produced a module we call SB0010MS. In SB0010MS, the input and output datasets, the class variable list and the analytical variable list are all parameters. SB0010MS first inquires the source file (here parameter INFILE) as to how large it is. This is a straightforward use of %SYSFUNC:

```sas
%let dsid=%sysfunc(OPEN(&infile.));
%let numrecs =
   %sysfunc(ATRN(&dsid.,NOBS));
%let rc=%sysfunc(CLOSE(&dsid));
```

SORTSIZE, the number of observations which can be sorted in a single execution of PROC SORT, is a parameter to SB0010MS. If the NUMRECS value exceeds the SORTSIZE by an amount over a certain tolerance, then SB0010MS executes a DATA-step wherein the source is busted into components with no more than SORTSIZE number of records. Clearly the number of these breakout files on the DATA statement can be determined from NUMRECS and SORTSIZE. There follows a macro %DO-loop that iterates through the PROC SORT’s of the components. A second DATA step then re-merges the sorted files. On the other hand, if NUMRECS is less than or equal to a tolerance level above SORTSIZE, the source file is simply sorted.

SB0010MS is ubiquitous in the RESMENU software, but it is not the only utility. SBTIMER, used in our longer-running programs, notes how long job execution-step groups take and writes this information to a little permanent file. This saves an archeological dig for such stats off the lengthy SAS Logs. Finally there are a pair of utility macros which handle certain glops of the consolidation process. They in turn invoke SB0010MS for the individual sort-and-summarise steps.

10.0 USE OF MACROS - 3: COMMENTS AND SAS-LOG DOCUMENTATION:

Many of the programs executed, as part of data retrieval for users or as part of our production, have quite a few steps and many parameters. An issue is making the SAS Log readable when verifying that things went correctly or determining (usually after a system change) why and where things went awry.

When a non-macro SAS program is executed in the SAS Display Manager, the program code and any comments appear with run messages. Thus the run messages directly tie back to the code and comments relating to the function being executed. On the other hand, when a macro is run, the “/* ---- */” type of comments won’t appear under any circumstance. Although activating MPRINT will cause the asterisk--semicolon (“* ----- ;”) form of comment and the SAS Code to appear with the run messages.

For purposes of production, it is quite unnecessary -- and even sometimes downright counterproductive -- to routinely run large programs with MPRINT always on. For routine checking, MPRINT generally gives much too much information. And through version 6.12, this barrage does not come in the most readable form - there are, after all, no cosmetic indentations. Still, it might be nice to quickly see what the program execution steps mean.

Now RESMENU programs are all SAS Macros. To address this issue, what might be regarded as “comments” within the program are set up as %PUT statements rather than comments of any of the types supported by SAS. These %PUT messages announce the
name of the program being executed, display parameter values the program is using, and, ahead of each job-step in the program, announce a description of the step.

As a simple check on production program runs and on-line access reporting software executions, that degree of information on the SAS-Log has proven sufficient to get started. A problem might be routine, as for example a bad parameter value from the SAS/AF application driver. MPRINT doesn't buy anything in that case. If it is not simple, then the problem program can be executed on test data in the Display Manager using MPRINT and what-have-you in the way of debugging tools.

At the top of this page is a SAS program shell showing how the use of the %PUT's function both as comments and also as run-messages to the SAS-Log. Note, consistent with the example, that the macro name is invariably also the name of the SAS program itself.

11.0 MANAGING THE SAS-SOFTWARE -- THE CONCEPT OF THE PROGRAM MANAGER:

One of the nice things about the Windows versions of SAS (6.08 through 6.12) was the interface with the Windows File Manager. When you hit the FILE-OPEN sequence with the mouse, you were presented with a nice list of your programs, and you could move easily around the directories to get other lists. To open a program you double-clicked on it in the presented list and - Voila! - there it was in the SAS Program Window.

Outside of VM/CMS, the text-based environments were not quite so kind. Using source code catalogs would present you a nice list, but you still had to use commands to include the source and save it back. One thing I personally did not care for were the commands involving four-level names.

Having been spoiled in Windows 3.1, I sought to replicate some of the benefits of that environment, and wound up building a "Program Manager" SAS/AF mini application, which is tied into RESMENU. (Go back to the second page; you will see RESMENU has an option called "Application Management".) What I am about to show you is accessible there in RESMENU and as a stand-alone, function-key-activated application.

As we noted earlier, all RESMENU programs are simply stored as files in VMS directories. The programs have a naming convention: a two or three position AA-code
("application area"), a four- or three-position sequence number, and a two-position program type. For example, the "SB" in the names SB0005PC and SB0184RP is the application area code for "data production update". "PC" is means "data processing module" whereas "RP" means "report generator". Another AA-Code we happen to use is "SUT" for generic utilities. Judicious naming of programs keeps programs having related functions clustered in alphabetic lists.

All the directories where SAS Programs are stored have SAS Filenames assigned to them. Earlier we saw the filename RESRVPGM on the SB0520RP screen as an example of this. Using this SAS Filename the following commands will fetch program SB0005PC into the display manager Program window and save it back to the directory:

```
INCLUDE RESRVPGM(SB0005PC)
FILE RESRVPGM(SB0005PC)
```

What the "Program Manager" SAS/AF application does is first present you with a list of program libraries. When you select one of these libraries and then hit the function key triggering the END command key, you get a list of the SAS Programs in the selected library. When you select one of these programs on the list and hit the END command key, the Program Manager issues the INCLUD command against the selected program, and it pops up in the Program window. (Just like Windows ….)

Note that the program listing presented by this “Program Manager” has both descriptions and a last save-date. Unlike the description information on the Source Code Catalog listing, however, the descriptions are not manually entered, but are derived directly from the second line of the SAS source code. See, for example, the entry for SB0184RP and compare to the previous page.

As new programs are added or existing ones are modified, the listing presented by the "program manager" needs updating. To refresh a listing for a library, hit the "hot button" called "Refresh Program List" on the first screen. If you do, it will execute a utility program which first generates a VMS directory listing of the ‘.sas’ files in the program directory and then, using this program listing, opens each SAS program file to pick off the revised description. In the process, it also picks off other information not displayed here, such as program dependencies — i.e. the macros (like SB0010MS) invoked by the program. This other information can be displayed on a report triggered by the other "hot-button" labeled "Print Library Listing".

ACKNOWLEDGMENTS:

SAS is a Registered Trademark of the SAS Institute, Inc. SAS/AF is a copyrighted product of the Institute.

Lotus is a Registered Trademark of Lotus Development Corporation, a subsidiary of I.B.M.

Excel is a Registered Trademark of Microsoft, Inc.

Pathworks is a Registered Trademark of COMPAQ, Inc.

12.0 LOS ENDES:

What you have seen is a tour through some ideas that we have found useful in tying together an application. Although having some separate descriptive documentation has its merits, there are a lot of things that can be done to make an application environment give you information without having to come by it the hard way. Tying data and processing components to one another using simple conventions helps a lot. Failure to do so is not easily compensated for by traditional documentation, which, in any case requires conscientious attention to keep it up-to-date. Housekeeping using the ideas here has the advantage that the information is less likely to fall out of date. It is part of normal maintenance and development.