Running Multiple SAS/AF® Applications in a Single SAS® Session

Mark L. Schneider, SAS Institute Inc., Cary, NC

ABSTRACT

For years internal users of Institute tools have been struggling with the startup time attributable to an interactive SAS® session. Since more and more SAS applications are being developed by departments like MIS, Development Support, QA, and Professional Services, invoking the various SAS sessions is not only time-consuming, but a growing resource crunch as well. The MULTISAS project attempts to address this problem by providing a framework and set of standards for multiple SAS/AF® applications to coexist within a single SAS session. Pertinent issues include user profiles, temporary data sets, global macro variables and macro routines, librefs/filerefs, formats, global SCL lists, options, window names, and interapplication communication. By sharing a single SAS session, applications require less memory and launch in a fraction of the time than they would in a more stand-alone form.

INTRODUCTION

At SAS Institute, we believe in leveraging our investment in the SAS System by developing powerful administrative, testing, and analysis tools based on SAS/AF. In the past, software developers from departments around the Institute programmed stand-alone solutions which were more monolithic than distributed in design. Of course, such full-featured tools exact a toll on system resources, and our users (other SAS employees) started to make a number of complaints:

1. Each tool took too long to bring up and down.
2. Only a limited number of applications could be brought up simultaneously without tying up all the system resources (usually memory).
3. There was no compatibility, let alone integration, of the tools.

Faced with these valid concerns, tool developers from MIS, QA, Development Support, Professional Services, and SAS/AF Development recognized the need to pursue a single-session approach to tools based on SAS/AF. We felt that a single invocation of SAS at the beginning of the day would be tolerable from our users' perspective, especially when coupled with a dramatic decrease in tool launch time once the session was active. Also, a single session would reduce the amount of memory required to support multiple applications. Strictly from a developer's aspect, there was a certain appeal to an integrated suite of tools, allowing for easier data sharing and correlation. The project was dubbed ''MULTISAS,'' for its attempt to incorporate multiple SAS applications into a single SAS session.

The result of our continuing efforts is a set of standards and conventions by which application developers can create MULTISAS compliant tools. This is in every way a living document, since we are discovering new challenges and clever workarounds as new tools are developed or converted to adhere to the standards. A list of the technical obstacles uncovered so far and our corresponding solutions follows.

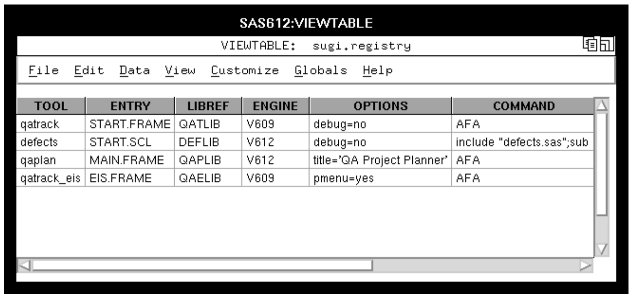
TOOL INVOCATION

The single most important feature of SAS allowing for MULTISAS functionality is the ''AFAPPLICATION'' command (or AFA for short). Introduced in Release 6.07, it starts a SAS/AF application in its own window, allowing execution of several SAS/AF applications at the same time. In fact, applications can either be called from a central launchpad program or from each other.

Although the AFAPPLICATION command allows a certain amount of flexibility within SAS/AF, there are technical problems with invocation methodologies employing code outside SAS/AF. Operating system-specific shell scripts (i.e., REXX, .BAT files, Korn shell, DCL, CLIST, etc.), AUTOEXECs, and -INITSTMT options previously submitted prior to starting a tool's first FRAME, SCL, or PROGRAM catalog entry cannot be allowed. Such code would interfere with a SAS/AF tool's ability to call another tool.

In addressing this problem, a basic philosophy was established that a MULTISAS tool needs to have a single catalog entry point which is dependent on no other code, SAS or non SAS, having been executed. All necessary tool initialization will be performed by this catalog entry. However, in order to access the entry point, a single libref must already be assigned which points to the containing library.

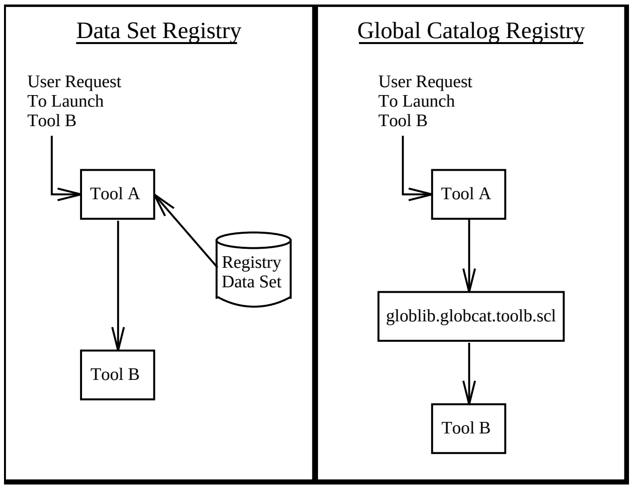
One way to handle this is by creating a SAS data set containing the MULTISAS registry of entry points (see Figure 1). The tool launcher (either another tool or a central launchpad) can then read the observation corresponding to the requested tool, submit a LIBNAME statement based on the LIBREF, ENGINE, PATH, and LIBOPTS variables (not all appear in the Figure), and execute the tool's startup command based on the COMMAND, ENTRY, and OPTIONS variables.



TOOL	ENTRY	LIBREF	ENGINE	OPTIONS	COMMAND
qatrack	START.FRAME	QATLIB	V609	debug=no	AFA
defects	START.SCL	DEFLIB	V612	debug=no	include "defects.sas";sub
qaplan	MAIN.FRAME	QAPLIB	V612	title='QA Project Planner'	AFA
qatrack_eis	EIS.FRAME	QAELIB	V609	pmenu=yes	AFA

Figure 1: Tool Entry Point Registry

An alternate approach is to maintain all entry points in a single site-wide catalog. The catalog eliminates the need for storing libref and startup command information in a separate data set since any required LIBNAME statements and startup commands can be submitted by the entry points themselves. Figure 2 shows a graphical depiction of how these two approaches compare with each other. Note that in the case of the global catalog registry, the *globlib* libref points to the site-wide tool library and is predefined for all applications.



Data Set Registry
User Request To Launch Tool B
Tool A
Registry Data Set
Tool B
Global Catalog Registry
User Request To Launch Tool B
Tool A
globlib.globcat.toolb.scl
Tool B

Figure 2: Comparison of Two Entry Point Approaches

The major advantage to the data set model is the ease with which the registry can be changed. Modifications to the global catalog model have the potential to disrupt tools in use at the time. The advantage of the catalog model is the simplified invocation method since the initial tool-specific LIBNAME statement is assigned by the tool's entry in the global catalog rather than by the calling tool (as in the data set model). Since the Institute's tools are supported by a variety of developers in several different departments, the data set model was chosen for its ease of modification.

One issue still under debate is the use of the INCLUDE and %INCLUDE statements, particularly as part of a tool's registered startup command. While there are no technical complications involved, some feel that including source code from flat files outside of SAS is unnecessarily indirect. Not only does it decrease application portability, but it makes maintenance more difficult by spreading code across both flat files and catalog entries.

On the other hand, flexibility of the standards is very important, especially given the large number of applications to be converted. The easier it is to comply with the MULTISAS rules, the faster applications will be available for our users. Generally speaking, the fewer restrictions placed on developers, the better. Therefore, any of the following commands would be allowed as entries in the MULTISAS data set registry:

```
INCLUDE "toolinit.sas"; SUBMIT

COPY "libref.catalog.toolinit.source"; SUBMIT

AFA C=libref.catalog.toolinit.scl
```

Note that the latter two commands would require registering a LIBNAME statement in addition to the commands themselves. The INCLUDE example could make use of such a preregistered LIBNAME statement as well, but this is generally frowned upon since such a libref could be assigned within the included source itself.

BASE SAS SOFTWARE ISSUES

Although much of the code behind SAS/AF applications is written in SCL, there are numerous occasions where DATA step code is warranted. Some examples include:

1. Generating reports using PROCs
2. Including previously written routines
3. Isolating frequently used pieces of code
4. Submitting code on behalf of the user (for example, user-customized reports)

Unfortunately, several common DATA step programming practices cause problems when placed in a MULTISAS environment. In addition, many features of the base SAS System (librefs, formats, and options) complicate the issue. When possible, the standards try to provide a technical solution. As you'll see, many of the challenges relate to potential naming collisions. In these cases, a naming convention is prescribed, employing a tool-specific, three character prefix.

SUBMIT Blocks

By far the most common way of submitting DATA step code from a SAS/AF application is through the use of SUBMIT blocks. In consideration of other applications within the same SAS session, tools must refrain from leaving submitted code in the PREVIEW buffer. This implies the use of a SUBMIT *when* statement in the last SUBMIT block prior to the next RETURN statement to execute, where *when* can be CONTINUE, IMMEDIATE, PRIMARY, or TERMINATE. Note that this is always true for SQL SUBMIT blocks since they require the CONTINUE argument.

Macro Facility

SAS macro variables are stored in a symbol table that is shared across all tasks within the same SAS session. Macro routines are typically stored in the WORK.SASMACR catalog, although they may also be stored as compiled macros in the SASMACR catalog of a user-defined permanent library (via the SASMSTORE= option). The global nature of these storage mechanisms allows for potential naming conflicts.

As a result, MULTISAS compliant applications must prefix all macro routines and global macro variables with their tool-specific, three character prefix. Whenever possible, local macro variables should be used. Since the scope of local macro variables is restricted to the parent macro routine, they are relieved of the three character prefix requirement. This would include any macro variables defined explicitly via the %LOCAL statement, or implicitly by being passed as parameters to a macro routine. Conversely, any macro variable defined outside of a macro routine (i.e., through %LET or SYMPUT statements) or via the %GLOBAL statement must use the three character prefix.

One exception to the naming standard for macro routines relates to those which are defined in open SCL code. Since these routines are evaluated at SCL compile time, no corresponding entries would appear in a user's SASMACR catalog at run time. Thus, developers have the freedom to name such macros whatever they would like. The following code shows an example of valid SCL/macro usage. Note that the XXXPRINT macro in the SUBMIT block must use the tool's three character prefix since it is defined in DATA step code, and thus is not evaluated until run time.

```
INIT:
   %macro initvar(variable);
      &variable = 0;
      put "I reinitialized &variable";
   %mend initvar;

   SCL statements

   submit continue;
      %macro xxxprint(dataset);
         proc print data=&dataset;
         run;
      %mend xxxprint;

      %xxxprint(data1);
      %xxxprint(data2);
      %xxxprint(data3);
   endsubmit;
return;
```

Librefs and Filerefs

Application-defined librefs and filerefs share the same global name space. As a result, all such constructs must include the tool's three character prefix. There is only one exception to this rule. The FILENAME function in SCL offers a way to request a system-generated fileref. If a character variable whose value is blank is passed as the fileref argument, FILENAME will generate and return a unique fileref for you. Here's an example of this technique:

```
unique_fileref = '';
rc = filename(unique_fileref, 'physical-filename');
```

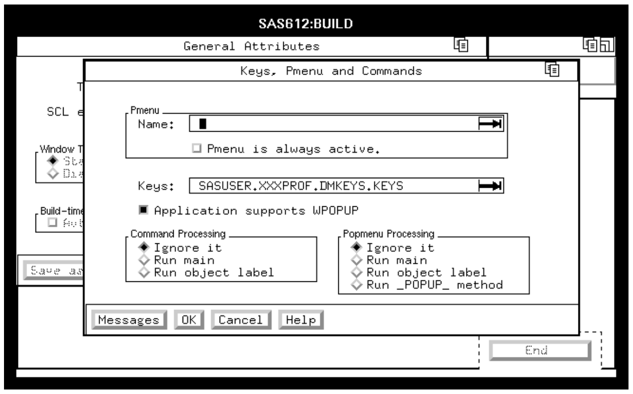
Upon execution, *unique_fileref* will contain the fileref which points to *physical-filename*. Note that this value will not use the tool's three character prefix, nor will it appear in the FILENAME window.

User Profiles

Because Version 6 of the SAS System does not currently support the concept of multiple user profiles within a single SAS session, all applications must share the same SASUSER library. Since all members of this library will be visible to all applications, care must be taken to uniquely identify all catalogs, data sets, etc. All application-generated members of the SASUSER library must use the three character prefix convention.

Furthermore, since SAS saves the PROFILE catalog in the SASUSER library, associated entries containing function key definitions, window attributes, and fonts can no longer be considered as application-owned. As much of this information as possible should be moved to application-specific catalogs/entries. For example, a FRAME developer could place common key definitions in a KEYS entry in a tool's source catalog. The general attributes of a FRAME could then point to this static entry. The downside to this is that users cannot make changes to key settings which persist from one session to the next.

A more user-customizable approach involves first creating a tool-specific user profile catalog at run time. The catalog name would need to employ the three character prefix, but could then exist permanently within the normal SASUSER library. FRAMEs would point to this dynamically created catalog for their key definitions (see Figure 3). Since the KEYS entry would persist from one SAS session to the next, tool-specific key definitions would survive.



Pmenu
Name:
☐ Pmenu is always active.

Keys: SASUSER.XXXPROF.DMKEYS.KEYS

■ Application supports WPOPUP

Command Processing
◆ Ignore it
◇ Run main
◇ Run object label

Popmenu Processing
◆ Ignore it
◇ Run main
◇ Run object label
◇ Run _POPUP_ method

Figure 3: Pointing to Tool-specific Key Definitions

Another problem, although less frequent, is with applications using the WSAVE command to preserve window attributes like color, size, and position of DMS windows. SAS ordinarily writes this information to WSAVE members of the SASUSER.PROFILE catalog. However, since WSAVE is limited to DMS windows (Program Editor, Log, Output, etc.) as opposed to other procedure windows like SAS/AF and SAS/FSP®, this problem is limited in scope. In fact, as long as a user does not mind sharing the same color, size, and position of DMS windows across all MULTISAS applications, this is not a problem at all.

Temporary Data Sets

Since only one WORK library is maintained for a SAS session, temporary data sets created by different MULTISAS applications must share the same name space. Forbidding use of the WORK library would solve the problem, but that is a bit drastic. Again, careful use of the three character prefix for all members written to the WORK library will address the potential for collisions. Another option is to create an application-specific XXXWORK library (following the naming convention) to hold temporary data sets. For performance reasons, this library should physically reside on the user's local workstation as opposed to on a fileserver, whenever possible. Like WORK, this library is temporary in nature. It is created when the application (not SAS) is started and deleted when the application is exited.

An alternative to using the three character prefix to differentiate temporary data sets is to use the SAS System's automatic naming convention feature. If you do not specify a data set name or the _NULL_ keyword in your DATA statement, the SAS System will create a uniquely named data set in the form DATA*n*, with *n* being incremented for each successive data set.

Formats/Informats

The FMTSEARCH= system option (Release 6.07 and above) governs the order in which format catalogs are searched. SAS always searches WORK.FORMATS first and then LIBRARY.FORMATS, unless either is explicitly listed in the FMTSEARCH list. Any catalog in any libref can appear in the FMTSEARCH list. Obviously, temporary (those in WORK.FORMATS) and permanent (those in all other catalogs) format and informat names have the potential to collide across these various catalogs. The MULTISAS project prescribes three standards to address the complications with temporary and permanent formats.

First, all application-specific temporary format and informat names must be prefixed with the tool's three character code. This is required since the SAS System always places such formats and informats in the common WORK.FORMATS catalog.

Second, all application-specific permanent formats and informats must either be prefixed with the three character code or reside in a temporarily loaded catalog. In the former case, the application can simply append the format catalog name to the FMTSEARCH list using SCL calls to OPTGETC (to retrieve the current list) and OPTSETC (to append the application's catalog to the list). In the latter case, temporarily loading the catalog implies appending it to the FMTSEARCH list, doing format processing, and removing it, all before control is returned to the user. For example:

```
old_fmtsearch = optgetc('FMTSEARCH');
submit continue;
   option fmtsearch=(mylib.formats);
   ...
endsubmit;
rc = optsetc('FMTSEARCH', old_fmtsearch);
```

Third, the LIBRARY.FORMATS catalog is reserved as a repository for formats and informats shared by all or many MULTISAS compliant applications. The special LIBRARY libref points to a single approved location accessible to all participating tools. Pointing the LIBRARY libref to any other location is expressly forbidden.

Options

SAS system options are in effect for all applications invoked from a SAS session. Although a candidate for change in Version 7, only one such set of options can currently exist in a given session. Once a SAS/AF application gets control, default values of options may have been set internally by SAS, through a configuration file, or at SAS invocation. A change to any of these options will be reflected by all applications. As a result, required application-specific option values must be set and reset (via OPTGETC/OPTSETC and OPTGETN/OPTSETN) before returning control to the user.

Application developers are also encouraged to use data set option equivalents, if available, in place of their system option counterparts. There currently exist eight such equivalents. They are BUFNO=, BUFSIZE=, COMPRESS=, FIRSTOBS=, LABEL, OBS=, REPLACE, and REUSE=.

Although not technically options, TITLE and FOOTNOTE lines should be treated in the same way as system options. Developers must take care in clearing any titles or footnotes set prior to returning control to the user. This is done by simply submitting the TITLE or FOOTNOTE statement without arguments. Similar care is needed for SYMBOL, PATTERN, LEGEND, and AXIS definitions, although they are most easily reset using a GOPTIONS RESET= statement, since executing a SYMBOL, PATTERN, LEGEND, or AXIS statement without arguments will only clear SYMBOL1, PATTERN1, LEGEND1, or AXIS1, respectively.

The restrictions on the use of SAS options are admittedly strict. Much of the effort involved in converting legacy code to MULTISAS standards involves the isolation or removal of option settings. One area of particular concern is the current widespread use of graphics options set via the GOPTIONS statement. SAS/GRAPH® intensive applications are forced to set and reset graphics options frequently just to avoid affecting each other's output.

SAS/AF ISSUES

Fortunately for the MULTISAS project, most features of the SAS/AF product allow for peaceful coexistence of multiple applications brought up using AFAPPLICATION commands. Each application maintains a separate address space for variables, lists, etc. However, there are a few precautionary measures that an application developer must take in order to prevent affecting the behavior of other applications.

SCL Environment Lists

SAS/AF supports the concept of shared data environments. This is basically the programming equivalent of global variables. With either local or global data environments, there is no need to pass variables via either CALL DISPLAY arguments or macro variables. Local data environments are application-specific and therefore offer no complications to the MULTISAS project. Conversely, the global data environment is accessible across all applications within the same SAS session. The ENVLIST function is used to retrieve either the local or global environment list ID.

In order to prevent collisions in the global environment list, applications must insert one and only one sublist, which is named the same as the application itself. The contents of each sublist are application-defined. Not only does this provide an isolated data environment for each tool, but it also provides a convenient way to determine which applications are currently active within a given SAS session. Furthermore, applications can use each other's sublists as a mechanism to pass data back and forth. An application must delete its corresponding global sublist before exiting.

Window Names

Sharing the same window name across two applications can cause trouble if one of the applications tries to execute a command based on the shared name. For example, ''NEXT *window name*; END'' attempts to close down another window within the application. However, it might result in accidentally shutting down part or all of an entirely different application in the same session.

There are two ways that the MULTISAS standards suggest to circumvent this problem. The first is to simply use a tool's three character prefix for all its window names. The second involves renaming windows just prior to pushing commands to them. The following shows SCL which retrieves a window's current title, temporarily re-titles the window using a unique string (for example, using the three character prefix or a datetime stamp), executes the HELP command, and then restores the title to its original value.

```
call send(frameid, '_GET_PROPERTIES_', proplist);
title = getnitemc(proplist, 'NAME', 1, 1, '');
rc = dellist(proplist);

call send(frameid, '_SET_TITLE_', '<unique title>');
call send(frameid, '_REFRESH_');

call execcmdi('NEXT "<unique title>"; HELP');

call send(frameid, '_SET_TITLE_', title);
call send(frameid, '_REFRESH_');
```

TOOL EXITS

Submitting an ENDSAS statement or executing a BYE command will immediately terminate the surrounding SAS session, regardless of which application makes the request. In order to prevent applications from prematurely terminating others, the MULTISAS standards prescribe a specific shutdown protocol. As mentioned above, an exiting application must remove its corresponding global sublist from the global environment list. It then checks the global environment list to see if there are any other applications still running within the SAS session. If so, it simply exits with no further action. If not, it executes a BYE command to immediately terminate SAS.

DIFFERENT RELEASES OF SAS

Existing SAS/AF applications developed at the Institute use a number of different SAS releases, including 6.12, 6.11, 6.09, and even 5.18. Obviously there are compatibility concerns with running applications from various releases within a single SAS session, and thus a single SAS release. SAS does offer downward compatibility for many entry types, but catalog-based applications sometimes require CPORT/CIMPORTing as well as recompiling.

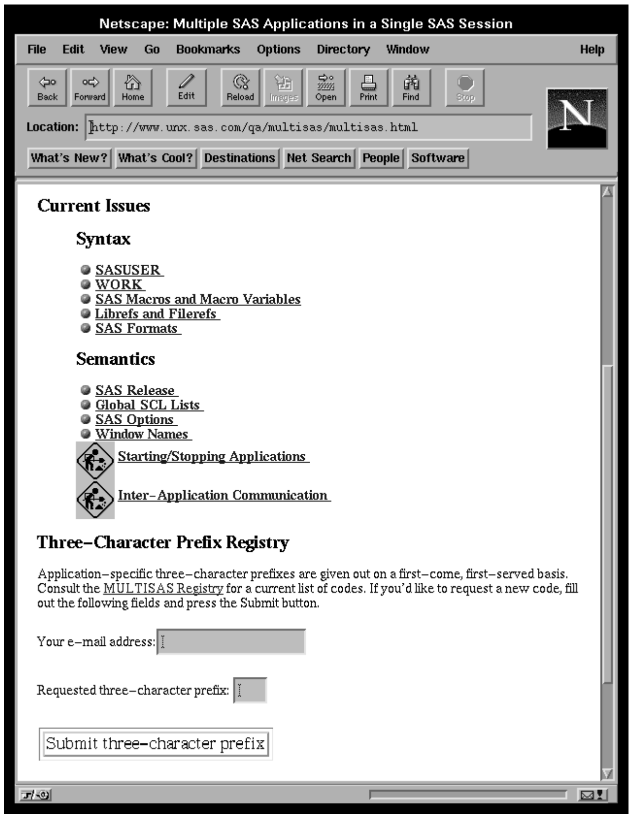
As a result, we decided to choose Release 6.11 as the lowest level supported in the MULTISAS scheme. Any applications developed at 6.10 and below would need to be converted to a supported release. This allows us to create generic ‘‘launcher’’ applications which serve as umbrella tools for all applications built under the same release as the launcher. It also allows FRAMEs at this release to share a common BUILD.RESOURCE with reusable customized class definitions.

ADMINISTRATION

The MULTISAS project brought on several logistical considerations, including:

1. Soliciting participation from various departments
2. Providing an open forum for questions, suggestions, and general discussion of proposed standards
3. Administering the registry of compliant tools and associated three character codes

Fortunately, most of these concerns were addressed by the creation of a Web page on our company intranet. Not only does the page document the current set of standards, but also provides an interactive means by which tool developers can join the MULTISAS project mailing list, register applications, and search past discussions on various topics. Figure 4 shows a portion of the MULTISAS Web page.


Netscape: Multiple SAS Applications in a Single SAS Session
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Location: http://www.unx.sas.com/qa/multisas/multisas.html
What's New? What's Cool? Destinations Net Search People Software

Current Issues

Syntax

- SASUSER
- WORK
- SAS Macros and Macro Variables
- Librefs and Filerefs
- SAS Formats

Semantics

- SAS Release
- Global SCL Lists
- SAS Options
- Window Names

Starting/Stopping Applications

Inter–Application Communication

Three–Character Prefix Registry

Application–specific three–character prefixes are given out on a first–come, first–served basis. Consult the MULTISAS Registry for a current list of codes. If you'd like to request a new code, fill out the following fields and press the Submit button.

Your e–mail address:

Requested three–character prefix:

Submit three–character prefix

Figure 4: Intranet Web Page Excerpt

PROBLEMS/PITFALLS

There's no doubt that creating MULTISAS compliant applications adds another level of complexity to the job of an application developer. Care must be taken not to introduce naming collisions which at best would cause applications to behave unexpectedly and at worst could cause data integrity problems.

SAS/AF programmers must also keep in mind the impact of context switching between applications within the same session. All activity with the potential for producing conflicts (SUBMIT blocks, setting options, passing commands to windows, etc.) must take place before control is returned to the user. Leaving any of this code partially executed or otherwise incomplete could have deleterious effects on other applications if the user chooses to switch to another SAS/AF task in the interim.

In practice, it has proven much more difficult to make existing applications MULTISAS compliant than it has to develop new ones in the MULTISAS mindset. For example, judiciously naming all constructs using a new tool's three character prefix is a fairly simple task. On the other hand, retrofitting such a naming convention on an existing tool's global constructs (temporary data set names, macro variables and routines, filerefs and librefs, formats and window names) can be daunting.

One area that proved particularly easy to miss was source code being called indirectly through %INCLUDE statements. Many application developers prefer to take advantage of configuration management tools offered by the operating system outside of SAS. This usually implies saving code to external flat files, editing them via operating system commands, and including them into SCL entries at compile time. Obviously, simply scanning SCL for MULTISAS compliance is by no means sufficient. All included source code must be checked as well.

One effective technique to trap some of the aforementioned pitfalls at run time is to continually monitor librefs in the LIBNAME window and temporary data sets in the DIR window for the WORK libref. Any entries which do not follow the MULTISAS naming conventions can be corrected as found.

PAYOFF

The most visible benefit to users of MULTISAS compliant applications is a drastic reduction in startup time. Since the SAS System is not initialized for each program, the only detectable delay is in program-specific initialization code. If kept to a minimum, applications launch almost instantaneously.

To demonstrate the reduction in startup time, the following configuration was used:

1. HP 9000/715 workstation running HP-UX
2. SAS Release 6.12
3. Simple SAS/AF application with no INIT code
4. All necessary SAS executables already cached in the local filesystem
5. No other active applications beyond standard operating system daemons

Launching the SAS/AF application in the traditional manner required between 28 and 30 seconds of real time, 17 of which correspond to SAS Display Manager System initialization. Launching the same SAS/AF application from another SAS/AF application in MULTISAS fashion required just slightly more than a second.

Perhaps an even more compelling payoff for MULTISAS compliance is in memory consumption. Multiple SAS sessions means multiple SAS images in memory. Running several SAS/AF applications in the traditional method can increase the demand on system memory considerably, usually causing page faults to virtual memory. This seriously hampers response time for all applications on the system.

Using the same configuration as for the startup time measurements, each SAS/AF application running under its own SAS session took up approximately 6948K of memory. However, when running this SAS/AF application multiple times under the same SAS session, only the first invocation required 6948K. Subsequent calls required between 1796K and 1856K of additional memory. Although the numbers will most certainly vary from system to system, it is apparent that MULTISAS execution offers a dramatic decrease in memory consumption for multiple applications.

One further area of potential for MULTISAS compliance is in the increased integration and interoperability among applications. Data mining which crosses program boundaries is simplified when all programs are active in the same SAS session. Data analysts and report writers no longer need to be concerned with setting up librefs and other metadata constructs. The applications themselves have already taken care of this. Analysts can simply bring up SAS/ASSIST®, the SQL Query Window, etc., and correlate the data as needed.

CONCLUSION

Although this paper addresses many areas of concern for multiple SAS/AF applications coexisting in the same SAS session, it is by no means an exhaustive list. As we convert more applications to conform to the MULTISAS standards, we continue to encounter new challenges. Still, we have yet to uncover any obstacle serious enough to counteract the obvious advantages in application startup time and memory consumption.

Future work in this area is planned to include:

1. Writing automated tools to check code for MULTISAS compliance.
2. Registering icons for each MULTISAS application to be included in a company-wide launchpad/toolbar
3. Creating a ''launcher'' icon class which uses the data set registry (see Tool Invocation section) to bring up a selection list of available tools and run the selected one. Such an icon could then be instantiated on any MULTISAS compliant FRAME entry.

ACKNOWLEDGMENTS

The author wishes to thank Meg Arnette and Howard Willey for their technical review.

AUTHOR CONTACT INFORMATION

Mark L. Schneider
SAS Campus Drive
Cary, NC 27513
phone: (919)677-8000 x6170
email: sasmus@unx.sas.com