

Taking Charge of the SAS Environment for Adaptive Production Reporting

Dwight Eggers¹

Abstract

A SAS[®] application developer needs to be aware of the elements of the environment in which his process will run and know how to manipulate these in a disciplined manner. This is especially important when attempting to implement an application that is transportable to other platforms or operating systems. It is also important for achieving modular code that is reusable by other applications.

This paper presents a technique by which the scope of portions of the SAS environment can be limited. It is described within the context of a case history for a specific production reporting application. The compiled macros, custom formats, and macro variables are manipulated with an environment stack so that the scope of these elements can be easily limited or restored during the life of the process. This mechanism is controlled within the program flow by the means of a script that drives the process.

Introduction

Report generation from operational databases is one of the most common SAS applications. The language provides extensive support for this through such features as SAS/Access, SAS/Graph, PROC REPORT, and ODS. With this rich tool set that SAS provides, the report application developer is left with the task of creating processes that are appropriate for his data and user base.

One of the challenges for the application developer is to develop production processes, run routinely over an extended period, that are adaptable enough that new instances of the report can be introduced with ease. A greater challenge is being able to customize the display set between individual reports.

This presentation describes a case history of a production reporting application that has

considerable flexibility in the data content and display set for each of the numerous reports being generated in a single run of the process. This is done in a way that optimizes the maintainability of the code by the use of an external script to drive the process and a mechanism to control the scope of the SAS environment. Before describing the application and how this flexibility is achieved, the SAS environment is reviewed.

SAS Environment

Many things control the environment in which a SAS process is executed. Some of these are determined by the site – operating system, file system, network resources – completely outside of the control of the SAS process. Beyond these, the application developer has complete control. To achieve a reliable, maintainable, and transportable application, implemented in code that is reusable across applications, then it is critical that the developer be aware of the environment and manipulate it in a disciplined manner.

The term “SAS environment” is meant to include everything that controls or is needed to execute a SAS step, excluding existing SAS data sets. These include SAS configuration and option settings, libref and fileref settings, the compiled macro catalog, global macro variables, custom formats, and the external parameters available to the SAS process from the system global settings.

The most common model for defining the SAS environment is a layered architecture where different portions of the environment are introduced or overwritten at the appropriate stage early in the process. The top layers are common to all operating systems and the SAS language processor: 1) system global variable settings; 2) parameter settings included with the invocation of the SAS language processor; 3) SAS config file settings; and 4) the code within

¹ The author can be contacted at Dwight.Eggers@verizon.net

the autoexec file. Thus the SAS environment is nearly complete before any application code is started. If no other action is taken, this environment will remain static over the life of the process.

Fortunately – or unfortunately, depending on your point of view – the SAS language provides the programmer with considerable control over the environment throughout the life of the process. The drawback of this flexibility is that the scope of most elements of the SAS environment, at least for Base SAS, is global to the entire process. Once an environment element is introduced or changed, this persists for the remainder of the process unless a corresponding action is taken to restore the previous environment.

The SAS users' community, through SUGI papers and SAS-L discussions, focuses a lot of attention on ways to manipulate the SAS environment. It is outside of the scope of this presentation, and too large of a task, to do a comprehensive review of these contributions.

One common element of the SAS environment is the application directory structure for code and data, as manipulated with the LIBNAME and FILENAME statements. Since this aspect of environment is virtually ignored in this presentation, one representative reference is cited (Carpenter & Smith, 2002).

The remainder of this presentation describes one reporting application. A major feature of this application is a mechanism to control the scope of portions of the SAS environment.

Application and Data

The Software Metrics group collects and reports statistics on corporate-wide software development activities at a major telecommunications equipment manufacturer². These reports are used for project, product, and process management. These reporting activities are supported primarily with SAS and Sybase.

² The company that sponsored this effort no longer employs the author. He was asked to exclude references to the company by name from this paper.

For this presentation, the basic data are software trouble reports (TRs) that are logged during development testing, system integration, and deployment. This software is for telephony applications on mobile and wireline telephone systems deployed worldwide. The TRs cover both code and documents. All product development and customer support operations throughout the company are linked to this global operational database which is known as MHS for Modification Handling System. This is a major tool consisting of about 700,000 TRs, accumulated over the 15-year life of the database, and used by about 6,000 software designers and support personnel. The overwhelming majority of these TRs are reported and resolved internally, long before system deployment. A copy of the MHS database is replicated to the Metrics Sybase environment so that the reporting activity does not impact the performance of the operational tool.

The MHS database consists of numerous tables covering all aspects of the TR, including a complete history of the TR flow from initial registration to final resolution. Some of these attributes are static, such as the software product ID and the nature of the problem, and others are dynamic, such as the current status of the TR. The only TR identifier that is mentioned in this discussion is known as the market reference. It is a unique free-form identifier of multiple segments, according to project or market naming conventions, together with an index number

There are two basic statuses of TRs -- open or closed -- though each has numerous incremental steps in the TR workflow. Open TRs are those that are receiving the current effort toward resolution. The closed TRs provide a history of the project and the quality of the product.

An extensive series of reports are created for the large variety of stakeholders in the project, product, and organization management structures. Throughout this presentation, the organizations that order the report are referred to as customers, though these are internal to the corporation.

The reports are distributed as web documents on the corporate intranet. There are typically about 350 unique TR reports active. A few are permanent, but most have a life span of a few months over the active period of a design project. The reports are updated twice daily so the reports do not lag the operational database by more than a few hours. There are other views of the data available within the operational tool and a SAS/Intrnet application, but these twice-daily snapshots of the operational database satisfy most needs.

Some of the roles of the customers, that determine their varied needs from the reports, include the following:

- project and organizational management -- track the activities within their scope of responsibility
- test leader -- track open TRs to resolution in order to coordinate testing activities
- quality coordinators -- monitor the nature of faults and where these are found within the development cycle

The Metrics group also generates reports against other operational databases for other aspects of the software development life cycle. The Metrics group has been performing the TR reporting activity for about 10 years.

Report Content

There are two things that characterize an individual report: 1) data content, and 2) display set. The data content is the most basic. It is referred to as the scope of the report. This is a specification, defined by the customer, of the criteria used to extract TRs from the MHS database. The display set, also referred to as the report template, is the set of graphical and tabular presentations of the data that pass the report scope criteria. The basic report template consists of about a dozen PROC GPLOT, GCHART, and REPORT displays.

Normally the display set is fixed -- the vast majority of reports use a standard template. This was a fairly rigid limitation of the first 5 years of this TR reporting activity. Occasionally a customer would demand something unique, and this would be accommodated with some ad hoc

logic and additional coding within the SAS application.

The TR reporting application in its current form evolved in two steps. Initially all of the data scoping was performed in Sybase using sql code in unix shell scripts -- one script per report -- and this subset of data was accumulated into a single flat file to pass to the SAS reporting process. As the number of reports grew, these individual Sybase scripts became costly and difficult to maintain. The coordination between Sybase and SAS processes was also very cumbersome. With the introduction of SAS/Access and increased competence of PROC SQL within the Metrics group, it was decided that the data scoping activity needed to be incorporated within the SAS process.

The second step was a radical redesign to support greater flexibility in the report template. The customers were becoming more diverse and creative in their presentation requirements, and the initial approach of ad hoc modifications to the display steps was becoming too difficult to develop and maintain.

A secondary requirement during this development effort was to improve the web presentation of the report by more fully utilizing the capabilities of html for such things as frames, drill-down using imbedded links, and downloadable versions of the report content in CSV format.

The remainder of this presentation describes how a solution to these requirements was implemented in a SAS production report process. The solution described here is unique to the above set of data and report requirements, but the same principles are appropriate for many reporting applications.

There are three major elements of the solution:

- Drive the SAS application with a script.
- Structure the report template as a hierarchy of macros.
- Maintain an environment stack where macros and other elements of the environment can be introduced/removed easily for individual projects or reports.

There is a two-level hierarchy of project/report in the SAS TR reporting process. Each run of the process cycles through the set of active projects. A project corresponds to an assignment from a customer and normally corresponds to an overall design project. Each project consists of multiple instances of reports, corresponding to the various subprojects or other subsets of the data as needed by the project. The directory structures used by the report process, for both input and output, correspond to this project/report hierarchy.

Report Script

The report set is defined to the TR report process by a script contained in a flat file that is read into a SAS data set at the beginning of the cycle for each project. This script identifies each report and the criteria used to select the TRs to include in the report. The script also supports modifications to the report template, but this aspect of the implementation will be deferred until the discussion of the Environment Stack.

The data selection criteria in the script are converted to PROC SQL code during the processing of each report. This code is added to the input stack from a DATA_NULL_step using the CALL EXECUTE facility (Whitlock, 1997).

The syntax of the report script is a shorthand notation that provides enough richness to cover the varied requirements of the TR report process. Portions of the SQL step that are common to all reports, or can be derived, are constructed by the program step. The main advantage of this shorthand notation is that it simplifies the maintenance of the report script.

There are three basic classifications of data selection criteria:

- TRs written on a particular set of products
- other attributes of the TRs -- particularly the market reference ID that identifies the project and other subordinate classifications within the project
- a union or intersection of the preceding two criteria

If a customer specifies that a report be scoped to an individual set of products -- for example, all

products that are new or being reopen for change within a design project -- then this product set is identified with the report order and stored in an auxiliary Sybase table. This table is known to the report process through a SAS/Access view, METRICS.TR_PRODUCTS that supports the TR reporting process.

There are 11 tables from MHS that contribute to the TR report. To simplify the report process, there is a separate Metrics process that maintains the required subset of TR attributes in a single Sybase table. This denormalized table is available to the SAS process through a SAS/Access view descriptor, TRBROWSE.MHS.

The easiest way to explain the data scoping portion of the report script is with a series of examples. Consider the following segment of a script that defines four reports:

```
.report: APOLLO-EUS-FT
.scope: MHS
.msco: market_ref like "APOLLO-
FT%" and site="EUS"
.ereport

.report: APOLLO-EUS-DESIGN
.scope: PRODUCT
.psco: project="APOLLO" and
country="EUS"
.ereport

.report: APOLLO-EUS-union
.scope: MHS or PRODUCT
.psco: project="APOLLO" and
country="EUS"
.msco: market_ref like "APOLLO-
FT%" and site="EUS"
.ereport

.report: APOLLO-EUS-join
.scope: MHS and PRODUCT
.psco: project="APOLLO" and
country="EUS"
.msco: market_ref like
"APOLLO-FT%" and site="EUS"
.ereport
```

The syntax consists of tag words that start the line, preceded by a dot. A colon follows most tag words. The text following the colon to the next tag word represents the content of the statement. An individual report begins with a **report** statement that identifies the title of the report. The **scope** tag word identifies which data sets are subset in extracting the TR set for the report. There is an optional **ereport** tag word, without content, that delineates the end of

the script segment for a report. A full definition of the script language is outside of the scope of this presentation, but by comparing the script with the resulting code it is fairly straightforward to recognize the syntax. A few additional comments are made following these expansions.

The first report, titled "APOLLO-EUS-FT", is for all TRs written within the APOLLO function test activity at the EUS site, regardless of which organization has design-responsibility for the products on which these TRs are written. The target audience for this report is the function test management at EUS. When this segment is encountered in the process flow, it expands to the following SQL step (note that the content from the script is highlighted in bold font):

```
PROC SQL;
CREATE DATASET ReportSet as
SELECT *
FROM TRBROWSE.MHS
WHERE
market_ref like
"APOLLO-FT%" and site="EUS"
;
```

The second report, titled "APOLLO-EUS-DESIGN", is for all TRs written on products within the responsibility of the EUS design center for the APOLLO project, as specified when the report was ordered. The target audience for this report is the design management for this project at EUS. When this script is encountered in the process flow, it creates the following SQL step

```
PROC SQL;
CREATE DATASET ReportSet as
SELECT A.*
FROM TRBROWSE.MHS as A,
METRICS.TR_PRODUCT as B
WHERE
B.project="APOLLO" and
B.country="EUS" and
A.ARTICLE_NUM=B.ARTICLE_NUM and
A.R_STATE=B.R_STATE
;
```

The third report, titled "APOLLO-EUS-union" is a union of the TRs that meet either of the criteria for the first two reports. It provides a view of all TRs that are related to either design or testing activities at EUS. When this segment of the

script is encountered in the process flow, it creates the following SQL step

```
PROC SQL;
CREATE DATASET ReportSet as
SELECT A.*
FROM TRBROWSE.MHS as A,
METRICS.TR_PRODUCT as B
WHERE
  (A.market_ref like
"APOLLO-FT%" and
A.site="EUS"
  )
OR
  (B.project="APOLLO" and
B.country like "EUS" and
A.ARTICLE_NUM=B.ARTICLE_NUM
and A.R_STATE=B.R_STATE
  )
;
```

The forth report, titled "APOLLO-EUS-join" is the intersection of the TRs that meet both criteria of the first two reports. It identifies all TRs that are part of the EUS design subproject and were identified in EUS function testing. The resulting SQL step is identical to the above, except that the "OR" between the major segments of the where clause is replaced by an "AND" connector.

There can be multiple "mscope" or "pscope" statements. When these are encountered, there is an implicit "OR" connector between each subsetting condition.

There are additional options for data scoping that are possible in the TR report process scripting language, but these examples identify most of the basic features and provide a sense of the assorted combinations of data subsets that are available. The entire script is parsed into a SAS data set at the start of each project cycle and then the SQL step is build for each individual report as the process cycles through the report set for a project.

The richness of the scripting language is determined by the complexity of the application. As the data selection criteria are more varied, this adds to the complexity of the script parser and to the step that builds the SQL code.

The data selection criterion is a basic and required element of each individual report, but there are other elements to the script language

that will become apparent in the remainder of this presentation.

Hierarchy of Macros

As was indicated in the introduction, the primary objective of the redesign of the process was to support flexibility and maintainability of both the scope of the data that is included in each instance of the report, and easy customization of the display set in each report. The above discussion of the reporting script has described how this flexibility is achieved in defining the data that goes into each report. We now turn to the customizing of the display set.

The approach to customization that was followed in the early years of this reporting process was to add logic to the code so that some segments are activated or skipped depending of conditionals passed in macro variables or linked to the project name. This worked fine initially, but as the count of reports grew and the customers got more creative about the kinds of presentations they wished to have, this approach of making incremental changes using ad hoc logic became too cumbersome. Each change added new complexity to the code. It was recognized that it would be much simpler to be able to swap out or introduce completely new code segments to the process.

The report template represents a hierarchical document of chapters and subchapters down to as many levels as needed. At the lowest level of this tree structure, the leaves represent an individual display. The overlying levels represent the structure of the report -- the chapter and subchapter headings.

The code can be structured in the same way as the report template, with each element of the report structure implemented as a separate macro. To illustrate this, consider a report of three chapters. The macro to support the top level structure looks like the following:

```
%macro report;
  %summary_display;
  %chapter1;
  %chapter2;
  %chapter3;
%mend report;
```

If the first chapter contained two subchapters, then the chapter1 macro would look like the following:

```
%macro chapter1;
  %subchapter1_1;
  %subchapter1_2;
%mend chapter1;
```

If subchapter 1.1 contain two displays, then this macro would look like the following:

```
%macro subchapter1_1;
  %display1_1_1;
  %display1_1_2;
%mend subchapter1_1;
```

The display macros at the bottom of this structure contain the actual SAS/Graph or PROC REPORT calls to generate the output for the report.

The above discussion is highly abstracted and somewhat over simplified to introduce the concept of the macro hierarchy. Figure 1, on the next page, is the table of contents page for an actual report as it would appear on the web.

The title of the report appears at the top. This is followed by a summary table of the overall totals for major categories of interest. The rows correspond to different groups by status. The columns differentiate TRs into All TRs and those that have been classified as software faults. These are further differentiated by activity for the entire duration of the project and the last 7 days.

Following the summary table are the titles of report chapters and their displays. Each line is linked to another web page that contains tables or graphs of greater detail. The chapter level is linked to a summary table for this aspect of the data. The first chapter, Report Scope, is a description of the data selection criteria that define this report in the driver script.

You may recognize that the above discussion of a hierarchy of macros does not correspond exactly with the actual report. There is no mechanism to generate this summary page with links to subordinate displays. Another subtle feature of the actual report is that at times the data set created by the selection parameters is empty, or the underlying display may represent a data set that is empty. For these cases, it is

desirable that the full outline of the report be generated, but that the link not be activated.

It is difficult to strictly align the report structure with a macro hierarchy, as initially implied in this section, but the general concept is valid.

Figure 1: Sample table-of-contents page

V11_INDUS-MSC-OVERALL

Activity	TR Count		Fault Count	
	Overall	Last 7 Days	Overall	Last 7 Days
Registered	1,365	2	837	0
Answered/Open	12	5	8	2
Closed	1,269	8	826	7
Canceled	60			
Unanswered	24			

Report Scope

Summary Statistics

Cumulative

Open/Registered by Week

Current Status by Week Registered

Distribution of Fault Codes

Distribution of Answer Codes

TR Listings

Unanswered

Answered/Open

Closed

Canceled

TR Answering Performance

Design Answer Overall Goal Assessment

Design Answer Weekly Goal Assessment

TR Closure Performance

RE-to-Closure Overall Goal Assessment

RE-to-Closure Weekly Goal Assessment

Environment Stack

The motivation for coding of the report content as a hierarchy of macros is to be able to modify elements of the display set for individual projects or reports.

Macros that compile at run time, either from inline code or through the AUTOCALL facility,

are stored in the catalog WORK.SASMACR. If we wish to change elements of the display set, this can be done by introducing alternative versions of the macros of the standard display set into the run stream. It is fairly obvious how this could be achieved with an extension to the report script. For example, if we wanted to change out the first display in chapter 1.1 of the structure described above, we could define this in the script as

```
.report: SPECIAL
  .macro: alternat1_1_1.sas
  .scope: ...
  ...
  .ereport
```

where the content of the script **macro** statement is the name of a file that defines another version of macro **display1_1_1**. The report process would also have to be modified with a step that would generate an **%INCLUDE** statement, again with the **CALL EXECUTE** facility, for this designated file.

The drawback of this simplistic approach, however, is that once this alternative version of the macro is compiled, then the original version is overwritten in the current SASMACR catalog. We need to restore it by some means when the **SPECIAL** report is completed. Otherwise all of the subsequent reports would be generated with this new display.

One way to overcome this is to create a copy of the SASMACR catalog before this new version of the **display1_1_1** macro is compiled. Then after the processing of the **SPECIAL** report is completed, the original version of the macro environment is restore by deleting the SASMACR catalog, which has been modified for this one report, and then rename the save copy back to SASMACR.

A similar approach can be used for other components of the SAS environment for this report process. For this application, it is desirable to also be able to substitute simple custom formats and global macro variables. The formats can be handled in the same way as the macros, since these are also represented in the SAS environment as a catalog, **WORK.FORMATS**.

The global macro variables present a different challenge, since these are stored in an internal symbol table that can not be handled in the same way as a catalog. They can be copied into a data set, however, by using the SASHELP.VMACRO view at the beginning of the cycle. Then this global macro environment is restored at the end of the cycle by comparing the value of each macro variable in the current SASHELP.VMACRO view with the earlier saved data set. If differences are encountered, then %LET statements are inserted into the run stream, again using the CALL EXECUTE facility, to restore the original settings to these macro variables.

This mechanism for saving and restoring portions of the SAS environment is referred to as an environment stack. The save/restore functionality is also referred to as push/pop operations on the environment stack. For this application the stack can have up to three levels:

- 1) The **base level** represents the production version of the report template, which remains static for the life of the TR report process.
- 2) The **project level** contains the base level environment overwritten by customization that is to span the scope of the entire project.
- 3) The **report level** contains the project level environment overwritten by customization that is unique to an individual report.

An example of a report script that uses these features will help to explain these processes. Consider the following expansion to the earlier script example:

```
.project: APOLLO
  .global: csv_output=YES
  .format: invaline $fault
           'B'--<'D'='YES'
           other='NO'
  .macro: alternatel_1_1.sas
.report: APOLLO-EUS-FT
...
.ereport
.report: APOLLO-EUS-DESIGN
  .macro: special1_1_1.sas
  .scope: PRODUCT
...
.ereport
```

The **project** tag word has been introduced to identify a level of the directory structure for input

and output. There is a separate directory structure for each set of customized macros. The following three tag words -- **global**, **format**, and **macro** -- identify changes to these portions of the SAS environment at the project level. The first report, APOLLO-EUS-FT, would be generated with the display set defined by this project level environment. The second report, APOLLO-EUS-DESIGN, would create a new report-level environment by compiling a further customization of the 1.1.1 display. When the processing of this report was completed, then the project-level environment would be restored by popping it from the stack. The remainder of the APOLLO reports would be processed using the restored project-level environment. Then when all reports of the APOLLO project had been completed, the original base-level environment would be restored.

This operation of managing the environment stack is performed by a pair of support macros, **configur** and **reconfigur**. At the beginning of the project or report cycle, %**configur** examines the data sets created from parsing the script to identify changes to the environment. If found, then the portions of the existing environment which are being modified are pushed onto the stack and the appropriate %LET, %INCLUDE, or PROC FORMAT statements are inserted into the run stream. At the conclusion of the report or project cycle, %**reconfigur** restores the old environment from the stack. There is enough logic in these macros to only save and restore environment when necessary.

The base-level report needs to be explicitly defined in a separate script at the very beginning of the process and introduced to the environment using %**configur**. It is still possible to use the AUTOCALL facility, but it is important that the portions of the macro environment that are subject to change by this stack mechanism not rely on the AUTOCALL facility.

Even though this stack mechanism provides new power to control the SAS environment, the usual pitfalls still exist. If a global macro variable is reassigned within a macro, and this global variable portion of the stack has not been activated for this report, then that setting becomes part of the base level environment and persists for the remainder of the process.

Other portions of the environment – options, goptions, libref declarations, etc. – are not preserved by this mechanism. If the custom macro needs to reset any of these, then this macro should explicitly save/restore this setting on entry/exit. For example, if the size of the graph is explicitly declared in the new display macro, then we can assure that the GOPTION environment is not corrupted by the following technique. At the beginning of the macro, include the segment

```
%* save the current size;
%local oldsize;
%let oldsize=%str(hsize=)
    %sysfunc(getoption(hsize))
    %str(vsize=)
    %sysfunc(getoption(vsize));
```

At the conclusion of the macro, the graph size portion of the old environment can be restored using the statement

```
goptions &oldsize;
```

The initial segment created a local variable that is a complete graphic option setting. The concluding statement restores these settings. It is good practice to explicitly declare this variable as local. GOPTIONS, and a very few OPTIONS, seem to be the only situations that warrant a change in the environment as such a low level of the process.

Discussion

This report process has been in use for about five years. It has proven to be very robust. The process code has remained virtually untouched following the initial year of debugging and iterative refinement of the features. Well in excess of 2,000 uniquely defined reports have been run by this current version of the TR report process.

This kind of an environment model, together with the driver script, provides a radical change in the management of a production reporting process. Editing the driver script, completely independent of the SAS code for the process, controls the active set of reports. When a customized display is requested, the basic code remains untouched. The new display usually represents a modified version of an existing display macro. The code for this new display is a copy of this design base, which after modification is placed in the appropriate code subdirectory organized

by project or product line. The environment portion of the driver script also needs to be modified to introduce this new display. The display macros remain relatively clean of macro logic since each unique display has its own macro, even though numerous macros may have the same macro name.

This ease of maintenance of the SAS process is not to suggest that the TR reporting process is self-sufficient. It still requires a sustained effort to maintain the order-entry and web presentation ends of the overall operation. The Sybase side of the operation also needs to be monitored regularly to assure the integrity of the basic data. These operations, though, were also required before the SAS process was modified. Overall, this redesign of the TR report process has substantially reduced the unit costs of individual reports, and nearly all of the SAS programming effort has been directed to creating value-added new displays for the customers and modifying other report processes to conform to this model.

Manipulation of the environment stack involves some additional overhead to the execution time of the process, but this is a very small part of the total execution time. In practice, it is rare for the display set to be customized at the level of an individual report. Customization of the environment is quite common at the project level, but there is typically no more than a 15 to 20 projects active in a run, each of which may have about 10 to 40 reports. With this usage scenario, the push/pop operations of the environment stack are very infrequent and the additional execution time is insignificant.

Even in the extreme case of every report having a unique display template, the maintenance of the environment stack would remain small compared with the data extraction operations against the database and the data presentation steps.

Conclusion

All SAS development that expects to be run on a routine basis needs a keen awareness of the SAS environment and how this is coupled to the requirements of the application. Changes to the environment within the process should adhere to rigorous coding standards. A layered architecture provides a well-proven model by

which to structure the application and manipulate the environment.

The requirements for some applications are sufficiently complex, such as the adaptive reporting application described in this presentation, that it is beneficial to take extraordinary measures to control the SAS environment. The concept of an environment stack allows you to limit the scope of select portions of the environment so that it is easy to restore to an earlier configuration after changes are no longer needed.

This model does not provide control over the entire environment. Each component of the environment needs to be manipulated individually. This technique does not eliminate the need for strict discipline on how environment changes are made in the process flow.

References

Carpenter, Arthur L & Richard O. Smith (2002). Library and File Management: Building a Dynamic Application. *Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference*. SUGI, Orlando, FL. Paper 21-27.

Whitlock, H. Ian (1997). CALL EXECUTE: How and Why. *Proceedings of the Twenty-Second Annual SAS Users Group International Conference*. SUGI, San Diego, CA. Pp. 410-414.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.