A DEVELOPMENT ENVIRONMENT FOR APPLICATIONS BASED ON SAS® SOFTWARE

Danielle Davis of SAS Consulting Services Inc.

ABSTRACT

Data structures, design methodologies, and programming tools are the foundation components of an application development environment. This paper describes these components using a SAS software perspective. Specifically, the paper covers design methodologies, management of application components, preparation and maintenance of parallel test and production environments, and reusable code libraries. A sample SAS application that uses the described techniques concludes the paper.

INTRODUCTION

Most application developers would agree that the final product should be easy to maintain and expand. This paper presents key aspects of application development that provide the foundation for this desirable goal with emphasis on using SAS software for rapid application prototyping. The paper discusses, from a SAS perspective: iterative development, table-driven architecture, and reusable code libraries. A discussion on how to manage the application including packaging, structuring, and naming conventions follows.

Establishing a framework for application development has several benefits:

- Application developers can concentrate on the functionality and presentation of the application itself.
- Enhancements and changes to applications are easy to make.
- A team of developers working on a larger project can integrate their efforts easily.
- Application design time is shorter.
- Application development time is shorter.
- Startup time is less for new people on all projects once the framework is put into place.

APPLICATION DEVELOPMENT METHODOLOGY

The growth in technology and available software tools require an examination of the methodology used in application development. Contrast the traditional development approach with the rapid prototyping approach.

Traditional Development Approach

There are usually four aspects to traditional application development:

1. requirements gathering
2. requirements analysis
3. system design
4. system implementation

Some of the disadvantages of this approach include:

Long Development Cycle

The traditional approach often requires long development cycles. Much time and effort is often spent on writing functional and design specification documents. The length of time from when the requirements are defined until the first line of code is written can be significant; months, sometimes even years. During this time, business needs can change, and changes require rewriting the specification documents, thus lengthening the cycle.

An additional potential disadvantage to a long development cycle is lost investment. The benefits the application may have provided to the business may not be realized in the longer time frame.

Inflexibility

Once the specifications have been written, there is usually strong resistance to making changes. This can often discourage the inclusion of otherwise desirable newly realized features. Even worse, such inflexibility can lead to the acceptance of undesirable features.

Poor Programmer Utilization

The programmers on staff are often in limbo until the specifications have been approved. This is a waste of an expensive resource. A less tangible but important side- effect is diminished programmer morale.

NOTE: The disadvantages of the traditional approach apply regardless of the programming language used for implementation. Whether you use COBOL or the SAS language, utilizing the same traditional approach to development will incur the same disadvantages.

Rapid Prototyping Approach

The advances in technology since the mid-1980's allow a more flexible and streamlined approach to application development. Specifically, a proven strategy is to employ a rapid prototyping approach that utilizes iterative development and table-driven architecture as its cornerstones. The key advantage of rapid prototyping is that it allows the user to quickly get an idea of the general "look and feel" of the application and, more specifically, see the specific functionality. The results are frequent opportunities for review and feedback. Flexibility is inherent to the process.

Iterative Development

The initial phase of an iterative development effort is geared toward ensuring that the requirements for the system are well defined and documented. As an example, the document might include a description of the application's data structures. The tangible result of this phase is a summary document that highlights the issues and functions and briefly describes the approach.

Unlike the traditional approach, the iterative development approach emphasizes the use of prototypes. The idea is to provide increased functionality with each iteration and involve the user in the definition, development, and testing of the application.

A first iteration of the application can be available for the user for review within several weeks of beginning work. This provides the user and the developer with early assurance that they are on the right track. If changes need to be made, they can be addressed early in the development process. Each iteration can include new

features or capabilities and at the same time can incorporate comments or corrections from previous iterations. However, the frequency of delivery of iterations should be carefully balanced with the functionality to be included.

During the development process initial prototypes are expanded to become *functional iterations* which are prototypes that are self-contained units of the application. They can be tested independently and are often associated with a distinct group at the user's site. Large applications usually require at least two functional iterations.

The iterative development approach is extremely successful because it promotes a close partnership between the developer and the user. The design effort becomes a dynamic process that can readily incorporate unforeseen needs and take advantage of products and facilities that were not available at the beginning.

Table-Driven Architecture

This section describes a system architecture by which various components of an application are maintained via parameter files; in this case implemented as SAS data sets. The technique of using parameter files in this fashion is called table-driven architecture. An application using table-driven architecture allows for more flexibility and lower maintenance than one in which the developer writes code to perform the same functions. Examples of parameter files could include:

- Specifying the parameters that 'drive' a report.
- Specifying the list of items in a menu.
- Specifying user access privileges.

A table-driven application performs under the direction of values in a parameter file. This allows changes to be implemented by altering values in parameter files instead of rewriting or adding code.

The table-driven approach is adaptable to a wide variety of data processing tasks involving varying degrees of complexity. Table-driven features can be used at all levels of an application, from low-level aspects such as determining which values in a selection list are character or numeric to more visible features such as specifying which items should appear in a menu. A system incorporating this design principle can significantly facilitate the user's ability to maintain, customize, and enhance an application.

Rapid prototyping that incorporates iterative development and table-driven architecture is a very powerful methodology. For example, instead of writing a document describing a menu interface, use SAS/AF® software and a parameter data set that defines the menu items to quickly build the menuing application. This can often be done in a matter of hours. If the user wants to change the menu definition, change it immediately by modifying the parameter data set in which the menu definition is specified; or walk the user through changing it. Choosing the second alternative can aid in turning over the application to the user.

It is important to emphasize the synergy of iterative development with table-driven architecture that is packaged and quickly delivered to users via prototypes. Changes to the design of the application are inherent to the life cycle of the project. Put another way, design changes are often simply coding changes; they are one and the same. Users are part of this dynamic process since they can see and respond to the changes as they occur. Much of the development effort is done in parallel. Contrast this outcome with the traditional approach to system design that tends to work in sequence with static elements with little attendant flexibility.

Reusable Code Library or Toolbox

A reusable code library is a collection of software routines or modules that can be used in the development of new programs. At SAS Consulting Services, we have a reusable code library that we refer to as our Toolbox.

Examples of elements in our Toolbox include:

- SAS/AF PROGRAM and FRAME entries
- pull-down menus
- FORMS
- SAS sample code
- utility macros.

There are many benefits to having a Toolbox; the benefits closely parallel those associated with "object-oriented" programming which has seen a recent general acceptance in the industry. Some of the benefits are:

Reusability	Toolbox entries can be used repeatedly across different applications. This reduces the requirement for application-specific code.
Reliability	Toolbox objects have already been debugged and tested. Their use, therefore, enhances the reliability of applications.
Standards	Using Toolbox entries provides an automatic way of enforcing coding standards within and across applications.
Productivity	Using Toolbox realizes greater productivity by reducing coding time.
Modularity	Toolbox entries are modular. Their use, therefore, facilitates modular structuring of applications.
Modeling	Model Toolbox entries are adaptable to specific applications.

Note that a single SAS data library can serve as the repository for all Toolbox code. The different elements can be packaged to allow easy retrieval of all related components. For example, each Toolbox SAS/AF member is stored in its own SAS catalog. All program entries, frame entries, pmenus, source entries, etc., that support the tool are in the same catalog.

Development, Test, and Production Environments

The life cycle of an application should be supported by three distinct and parallel application environments: development, test, and production. Each of these environments has all the components of the application; the difference is in the way they are used. As the name implies, ongoing development occurs in the development environment. Application testing, including unit and system testing conducted in a formal and structured way, occurs in the testing environment. Note that this is not the same as the ad hoc testing that programmers do when they are developing. The most stable release resides in the production environment. This will usually be the version that was last delivered to the user.

The main advantage of having these distinct environments is that it allows three key activities to be conducted in parallel, yet without fear of any one environment being accidentally "tainted" by another. Additionally, and just as important, each environment constructively affects the other. The development environment, for example, can revise existing functionality that must be newly tested in the test environment. The test environment will affect the development environment if bugs are found. The production environment influences the test environment as users find new ways of executing the application that require testing. Bugs found in production are fixed in the development environment.

MANAGING THE COMPONENTS OF THE APPLICATION

It is important to organize the components of the application in a modular and rational fashion. Issues such as naming conventions for the application components and the location where the components are stored must be resolved. Using a logical naming convention makes it easier to identify, store, and retrieve the application components. Work on the application proceeds with greater efficiency and reliability which, in turn, contributes to quicker and more accurate application development.

An application usually consists of several software components. Some of the more obvious ones include:

- parameter files
- macros
- application source code
- application data
- user-defined formats
- utility source code (for example, DATA steps that create the data sets of the application, indexing programs, etc.).

Components of interactive applications include:

- SAS/AF PROGRAM, FRAME, SCL entries
- pull-down menus
- screen definitions (FSEDIT)
- key definitions.

Using an effective naming convention and storing the components in an organized fashion is important to the project's success. There are no hard and fast rules to do this. However, the next section outlines a scheme that has proven itself over time at SAS Consulting Services, beginning with a generic description of the scheme. A sample application to illustrate how the scheme may be implemented follows.

Storing and Naming the Application Components

Store the application components in a hierarchy:

First level	the project identifier
Second level	a unique identifier for the application in case there is more than one application for the project.
Third level	indicates the version: Development, Test, or Production.
Fourth level	indicates the particular component of the application. This is the level in which all the elements of each component are grouped under an appropriate name.

Note that the method in which the hierarchy is specified depends on the file structure of the operating system of choice. Directory-based systems such as the DOS or Unix operating systems parallel the four levels above with four (sub)directory levels. In a mainframe environment such as MVS, the four levels are implemented as nodes of the operating system filename.

Fourth Level Groupings

Identify the application components and group them at the fourth level. The following are the more common groupings:

SAS	This is where the application's SAS programs are stored and are available for execution. Note that the programs could also be stored in a SAS catalog as SOURCE entries (see SYSTEM below).
MACRO	This is where SAS macros are stored when they are to be invoked as an AUTOCALL library.
RAWDATA	Raw data files. These are the non-SAS files the application needs.
FMTLIB	SAS format library. This is where the user-defined formats and informats are stored. The original PROC FORMAT code is also stored here as SOURCE entries in a SAS catalog.
PARMDL	This is a SAS data library that contains the application's parameter data sets, for example, those that support the table-driven architecture. This library also contains a DATASRC catalog with entries corresponding to each parameter data set. These entries define the variables, lengths formats, labels, etc., of each data set.
SASDL	This is a SAS data library that contains the application's data used for analysis and reporting.
SYSTEM	This is a SAS data library that contains the following SAS catalogs:

SOURCE	contains the application's SAS programs stored as SOURCE entries. This would also include miscellaneous "one time" code such as programs used to create SAS views. It is useful to store the programs in a catalog as this makes them more portable, even across operating systems.
SASMACR	contains compiled macros. The associated macro source code entries are also stored here as SOURCE entries. This facilitates retrieving the macros should changes be required.
WINDOWS	contains all SAS/AF programs, FRAME and SCL entries (for interactive applications).
PMENUS	contains the compiled pmenus. The associated pmenu source code is also stored here as SOURCE entries. Note that it may sometimes be more desirable to store the compiled pmenus in the same catalog as the AF programs that use them. This would allow the pmenu to be referenced in the GATTR window with a two-level name (for interactive applications).
SCREENS	contains the custom FSEDIT screens for data entry (for interactive applications).

A SAMPLE APPLICATION

The following application illustrates how the development methodology can be used. The sample application is based on a fictitious company, GIZMO Manufacturing. This system is referenced in the paper, "Building Data-Driven Object-Oriented Applications: Selected Techniques", presented at the Eighteenth Annual SAS Users Group International Conference.

The application contains the following elements:

- an application data file containing inventory data.
- an interactive SAS/AF menu system to enable regular users to run reports on demand and a system administrator to maintain the parameter files.
- several parameter files and FSEDIT screens.
- source code to create and update the parameter files on an ad-hoc basis.

- the macro that produces all of the application's reports.

Three data libraries support the application.

The library referenced by the libref **WIDGET** contains the inventory file stored as a SAS data set. It is stored separately from the parameter data files since those files are application specific, while the inventory data are used in a variety of applications at GIZMO Manufacturing. Storing the inventory data set in its own data library simplifies data access for all systems that need to access it.

The library referenced by the libref **PARMDL** contains the application parameter files that the system administrator is allowed to access and update. Specifically:

- a data set that contains the list of menu choices.
- a data set where each observation defines one of the available reports.
- a data set which defines how each report line in each report is calculated.
- a data set which defines the report stub for each report line.
- a data set containing the list of valid warehouse codes and their names.
- a data set containing the list of valid product codes and their descriptions.
- a data set that contains this list of parameter files.

The PARMDL library also contains a catalog called DATASRC. This catalog contains a SOURCE entry for each data file in this library. The entry in the catalog is the program used to initially define the variables, lengths, format, labels, etc. in the data set. For simplicity, the entry name is usually the same as the name of the SAS data set the source code creates. For example, the entry PARMDL.DATASRC.RPTLIST.SOURCE contains the source code used to create and maintain the PARMDL.RPTLIST data set. As each data set in a SAS data library must have a unique name, the corresponding catalog entry names are also therefore unique. Such a naming convention makes it easy to identify the source code that corresponds to a data set.

The **SYSTEM** library contains everything else that the application requires.

- WINDOWS contains the SAS/AF software entries including PROGRAM, FRAME, and SCL.
- SCREENS contains all of the FSEDIT screens for all of the parameter files. A naming convention of SYSTEM.SCREENS.<dsname>.SCREEN is used where <dsname> is the name of the data set. This convention forces consistency of usage and easy identification of the appropriate screen entry for each data set. For security and access reasons, all of the screens are stored in the SYSTEM library. You could also consider having a SCREENS catalog in each data library, using the same naming convention (similar to what was described earlier for the DATASRC catalog). Additionally, HELP and PMENU entries associated with each screen would be in this catalog.
- SASMACR contains both the source code and the compiled macros used by the application to produce its output reports. For example, the source code for the %REPORT macro is contained in SYSTEM.SASMACR.REPORT.SOURCE and the compiled executable macro is stored in SYSTEM.SASMACR.REPORT.MACRO. System options are used to define SYSTEM.SASMACR as the location to find stored compiled macros. Such a naming convention makes it easy to identify the source code that corresponds to each macro. See SAS Technical Report P-222 *Changes and Enhancements to Base SAS Software Release 6.07* for more information on using stored compiled macros.
- PMENUS contains both the source code and the compiled PMENU definitions (as created by PROC PMENU) using a naming convention similar to that described above. Again, such a naming convention makes it easy to identify what source code corresponds to what pmenu definition. One could also consider storing the PMENUs in another catalog (e.g., WINDOWS).
- SOURCE contains all other SAS programs used during the development of the application that, perhaps, cannot be safely discarded, but at the same time are not part of the application itself. For example, TESTPGMS.SOURCE contains programs that are used to test and analyze the data sets used to produce the inventory reports.

An applications developer could consider any number of alternative schemes for structuring and organizing applications. Using separate catalogs for the major components of an application combined with choosing a naming convention to facilitate the logical linking of entries makes managing and understanding the application development environment more straightforward.

CONCLUSION

The dynamic growth in technology and the ever-increasing capabilities of SAS software enable users to conceive and develop larger and more diverse applications than ever before.

Business needs are constantly growing and changing. To quickly respond to these needs, an application development methodology using flexible software techniques is a necessity.

The development methodology and environment described in this article have proven successful for SAS Consulting Services. It works across applications, increases development productivity, and provides flexibility in the applications delivered to customers. It has served as a model for users to continue to develop applications using SAS software as an effective tool in satisfying their business needs.

REFERENCES

Staff of SAS Consulting Services Inc., Rockville, MD (1993), "Building Data-Driven Object-Oriented Applications: Selected Techniques". *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, 18, 1331-1338.

SAS Institute Inc. (1991), SAS Technical Report P-222 *Changes and Enhancements to Base SAS Software, Release 6.07*, Cary, NC: SAS Institute Inc.

SAS, SAS/AF, and SAS Consulting Services are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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