Abstract
The SAS/AF® product in Version 8 of the SAS System® is a robust realization of an object oriented development environment. This product will allow developers to easily create sophisticated in-house applications.

MAJARO InfoSystems is the developer of a SAS-based clinical information system called ClinAccess that was originally developed using Version 6. As such, we are acutely affected by the version change of the SAS System.

We have been working for some time now on upgrading our ClinAccess product to Version 8 of the SAS System. This paper will attempt to detail the process of applications development in SAS Version 8 as gleaned from our upgrade experience.

Introduction
Working in an industry that is highly regulated by a large federal entity presents unique situations that require unique solutions. Add this to the multiplying expectations of managers and users alike and we have a healthy amount of work as clinical software developers.

As SAS developers in this industry, we need to learn to work smarter, because we all know that working harder has not gotten us where we need to be.

Why SAS is the Answer
With the FDA’s selection of SAS System data files as the standard for receiving electronically submitted data, the use of SAS becomes more critical to clinical information management. Considering that the FDA plans to accept most submissions electronically by 2002 and FDA reviewers will be equipped with the SAS System Viewer, keeping your clinical information in SAS makes sense.

Typically, analysis for a submission is done in SAS. In many companies reporting is also effectively accomplished using the SAS System. So why is the data not always gathered and stored in SAS?

Previous answers to that question have relied on “expert” opinion that SAS is not a “true” database. While it is correct to say that SAS is not optimized to be an online transaction processing (OLTP) database for handling millions of hits per minute, clinical data management does not present such a situation.

All other features that you expect from a database are available in Version 8 of the SAS System. These include:
- Generation data tables
- Password protection of data tables
- Encryption of data in tables
- Row-level locking for concurrent data entry
- Indexing
- Integrity Constraints
- Declarative Referential Integrity Constraints
- Audit trail
- 32K variable lengths for comments

As you can see, the SAS System certainly meets the requirements of any database system. SAS provides a clinical database solution.

Other reasons that have been suggested for not using SAS as a Clinical Information System have to do with the available clinical software. In-house systems have been deemed too expensive to develop and maintain. Thus, many companies have migrated to available off-the-shelf products. Unfortunately, most of these store your data in non-SAS data structures.

This has placed many companies in the unseemly and costly situation of having to maintain both a database software system and their SAS-based analysis and reporting applications, moving data between the different software platforms as needed.

An examination of the conditions that exist in this industry as well as the available software would lead to the conclusion that our environment requires a SAS-based solution for Clinical Information Management.

MAJARO InfoSystems has offered such a solution since 1987. Our product, ClinAccess, is an off-the-shelf SAS-based solution for clinical trial data entry, management, and review. ClinAccess keeps clinical data in SAS from start to finish. This paper will discuss features required of a Clinical Information System and how they are implemented in ClinAccess.
**System Scope**

Keep in mind that we are wearing a software developer’s cap now. Our methodologies are held to a much more strict standard that that of an ad-hoc SAS programmer. We’re looking at the big picture here.

The first step in software development is to determine the scope and life span of your application. You must have a concrete grasp of the goals you are trying to achieve with your software. These should be obtainable, measurable goals that ultimately help speed a drug to market while maintaining an auditable trail for regulatory purposes.

A specific goal may be to develop a system for managing report programs for a clinical trial, or a system for performing data entry from CRFs.

Other considerations at this time in the software development life cycle would be whether and how your application will be integrated into existing applications or whether the application is intended for further development.

Your specific goals will determine which blend of SAS System products you will need to build your application:

<table>
<thead>
<tr>
<th>Custom built applications:</th>
<th>SAS/AF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client/server application.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Custom built applications:</th>
<th>SAS/WebAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thin-client web-based application.</td>
<td>SAS/IntrNet</td>
</tr>
</tbody>
</table>

| Access to data in other database structures. | SAS/ACCESS |
| Concurrency access to data. | SAS/SHARE |
| Submission of SAS commands on a remote host. | SAS/CONNECT |

The ClinAccess product has been in continual development since 1987, so the scope of this software is large, including forms tracking, data entry, clarification management, data review and querying, reporting, generation of statistics and graphics, and thesaurus coding.

With an intended permanent life span, great care must be taken by ClinAccess developers to ensure that new technologies can be seamlessly implemented into the software over time. Recent new technologies implemented into ClinAccess include CRF imaging and web-based information delivery.

**Introduction to Frame Technology**

SAS/AF is the SAS System product that allows developers to create interactive applications using an object-oriented approach to development. A screen that is built to have a GUI interface is called a Frame.

<table>
<thead>
<tr>
<th>Structure of AF Applications</th>
<th>SAS/AF Frames are displayed as windows in an interactive session. These windows are populated with objects (buttons, entry fields, etc.) required for system execution. These objects, called components, are controlled by specific attributes and perform specific actions that are defined by methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Catalog</td>
<td>The AF application can invoke SCL code or other windows or Frames as required to perform the necessary actions requested by the user.</td>
</tr>
<tr>
<td>Application Execution</td>
<td>During the build process, the developer compiles the SCL used in the application. This compiled code is used during execution of the application.</td>
</tr>
</tbody>
</table>

**System Specifications**

Once the system scope has been explicitly defined, functional specifications need to be written. These are the blueprints for your system functionality. All desired functionality should be explicitly detailed in these specifications.

Statements like “easy to use” do not suffice as specifications!

Functional specifications often include screen mock-ups to start assist getting the GUI developed.

It is important to get end-user assistance in developing the functional specifications and screen mock-ups. End user acceptance and ease of use can make-or-break the most elegant application. The end users know the tasks they must complete. The software developer does not.
After the functional specifications have been approved, the software developer begins the process of writing Design Specifications. This occurs before one single line of source code is written!

There are two extremely important reasons to take the time up front to develop your specifications:

1) **FDA Requirements**
The FDA will look for these documents during audits. While the FDA has so far been somewhat lenient of imperfect software development and validation documentation, strict enforcement of these requirements are forthcoming.

2) **Accelerate Development**
This is a hard one to get your managers to swallow - especially when they want to see tangible development activities on a tight schedule. However, by taking the time up front to turn the functional specifications into design specifications, you work out any logic difficulties before you invest time in code that does not fit the system requirements.

**Standards**
There is one more topic I want to cover before we start writing any code. I know how much we as programmers dislike following standards. However, I cannot stress the importance of following standards in permitting you to develop more easily maintainable code.

These standards should include things like naming conventions, program header blocks, comments, coding style, user messages, etc. At MAJARO we also use standard colors in our program code. For example, comments are in white, blue for program links, magenta for macro code, green for messages, etc.

While it may not seem like a big issue if some developers code do blocks like

```
if x = 5 then do
    date = today(); time = time();
end;
```

while others code

```
if x = 5 then
    do;
    date = today();
    time = time();
end;
```

However, the cumulative effect of thousands of do blocks each coded differently in your system is a hindrance when maintaining code. Develop coding conventions to be followed by your application and train developers to use these coding conventions. You get code that can be quickly read by all your developers.

**System Architecture**
All applications require a basic structure; where the application executable code resides, how and where the data is stored, structure and purpose of metadata, etc. This information can be referred to as the System Architecture.

In a clinical information system, this architecture should include the following:

**Input and Output Clinical Data**
Good system architecture does not mix input and output data. For example, in a data entry application, there would be one location for storing data after first entry and another location for data after it has been second entered and verified.

This separation of data leads to the definition of data storage locations, where each data storage location is identified by the type of data that it contains. For example, in ClinAccess, there are multiple directories defined within each study. In addition to the separate data entry directories mentioned above, there are directories for audit trail data, discrepancy data, data entry screens, output data, metadata, and generated reports.

**Hierarchy of Metadata**
Clinical information systems are ideally suited for a hierarchical structure to metadata. This hierarchy could follow a path such as

```
Study → Drug → Site → Global
```

This would allow global standards to be established and used for all studies, with the ability to override these standards for a specific site, drug, and/or study.

For example, consider a codelist (format) for the variable GENDER. There may be a global standard at your company for coding 0 for Female and 1 for Male. But if a study is perhaps imported from elsewhere and the data contains F for Female and M for Male, you would define a study-specific codelist for GENDER for this study. The hierarchy ensures that the study-level codelist is used for this study.
This hierarchy could be used to store a multitude of information in addition to codelists, such as report title/footnote templates, user preferences, data dictionaries, etc.

**SAS Files: Executable Files**  
Your clinical system should reside in a secure area. Only proper users should have access to the locations of your system executable code.

**Application Parameters**  
There are a number of ways to pass parameters within your application. These include passing parameters directly when invoking a Frame, using run-time macro variable values, and metadata tables containing system specifications.

Probably the best way to pass parameters in a SAS/AF application is by using the Local Environment List. This is a special list that is automatically maintained by the SAS System while you are in an AF application. For example, you could store a User ID in the local environment list for use throughout your system.

Place the value of the SCL variable `userid` into the local list with a name of USERID:

```scl
rc = setnitemc(envlist('L'),userid,'USERID');
```

Read the userid from the local list:

```scl
userid = getnitemc(envlist('L'),'USERID');
```

**System Interface**  
The System Interface is how an application is presented to the end user for performing tasks. GUI interfaces are familiar to most users today and are easy to develop using SAS/AF.

**Navigation**  
A SAS/AF Frame permits users to navigate the application and issue commands via all the common GUI standards:
- Pull-down menus
- Pop-up menus
- Toolbars
- Push Buttons
- Function Keys

As a developer, it gets tiresome to write applications that have multiple ways to execute the same command, but this functionality is taken for granted by users in today’s environment and must be present in any application.

Wizards are another common way for systems to navigate a user through a particular task. This approach is very effective and easy to develop in SAS/AF.

**Messages to the User**  
I cannot stress the importance of providing messages to the user during the execution of your SAS/AF application. I spend huge amounts of my time as a developer checking return codes in SCL and coding the delivery of system messages to the application user.

There are a number of methods that can be employed to deliver messages to the user:

1) `_msg_`  
The `_msg_` is a message line that appears at the bottom of the SAS window. Placing messages in this line is very easy:

```scl
_msgs_ = 'Appears on bottom of the screen';
```

Unfortunately, this message line is often overlooked by users. Thus, messages of importance should not be placed here.

2) Message label  
Use a Text Label component on the Frame to display messages. This is a better solution because the message appears in the Frame window and is less likely to be overlooked.

This SCL statement uses dot notation to assign a value to the Label attribute of the Text Label component:

```scl
msg.label = 'Message in a Text Label.';
```

3) MessageBox function  
This SCL function creates a host message box that will display a passed SCL list, displaying each item in the list as a separate line in the message box. Various options are available for specifying the message box icon, title, and the buttons that are to appear in the message box. The returned value is the button pressed in the message box:

```scl
listid = makelist();
rc = insertc(listid,'My message');
button = messagebox(listid);
```

Despite requiring more lines of code because you must build the list to be passed into the function, this is the best solution for displaying important messages to the user.
4) **Messages to the SAS Log**

When extensive messages or diagnostics are needed, I often write these to the SAS Log. You can write as much as you want to the SAS Log and even use the Log to record a user’s movement and actions while using your application.

```sas
put 'This message goes to the Log' ;
```

It is quite helpful to have error messages return a code that a developer can use to refer back to a specific place in the application code.

## Classes

Each object on a Frame is based on a class definition, which is stored as a CLASS entry in a catalog. A RESOURCE entry contains a set of classes. The default Resources are:

1. Components
2. Version 6 objects

Generally, SAS supplied class entries are in `SASHELP.CLASSES`, while resource entries are in `SASHELP.FSP`. However, there are many other catalogs in the `SASHELP` library that contain these entry types. Classes may be added and/or dropped and other resource entries may be added to the Components window. Simply right-click in the Components window to display a popmenu of actions. This popmenu is also how the class dictionary online documentation is accessed.

It is important to know the classes available in order to be able to easily produce the desired functionality of your application.

Don’t be intimidated by the following list of classes supplied with SAS/AF. The component created by the class is exactly what you would expect from the class name, so it is quite easy to become familiar with the available set of components. You will find that your applications will mostly use just a few of these classes to accomplish most of your tasks.

### 1) Executing commands

a) Desktop Icon
b) Push Button

### 2) Selecting items

a) Check Box

![Check Box Example](image)

b) Radio Box

![Radio Box Example](image)

c) Combo Box

This control produces a drop-down list of acceptable choices for entry into the field.

![Combo Box Example](image)

d) Spin Box

This control allows the user to scroll through either a range of numeric values or a set of character values.

![Spin Box Example](image)

e) List Box

![List Box Example](image)

f) Library Selector *

Control to select assigned SAS libraries.

g) Member Selector *

Control to select SAS files.

h) Catalog Entry Selector *

Control to select entries from a catalog.

These useful controls (f-h) can be used to select various types of SAS files. A Catalog Entry Selector is illustrated on the next column:
i) Dual Selector *
    This is a selection tool with two lists – one list of available selections, the other list of selected items.

3) Data collection
   a) Text Entry
      Allows simple text entry into a field.
   b) Text Pad
      This notepad with text wrapping capabilities is for collecting large amounts of text.
   c) Data Table *
      A Data Table displays an entire data table in a spreadsheet layout, with columns and rows.
   d) Data Form *
      The Data Form is an object that is used to display a row in a table for edit or browsing.

4) Annotations
   a) Container Box
      This is used to draw a box on the Frame.
   b) Graphic Text
      Large, colorful text using SAS/GRAPH fonts.
   c) Text Label
      Simple, black text using host fonts.

5) Graphics
   Use of these controls requires SAS/GRAPH.
   a) Graph Output
      This displays a generated SAS/GRAPH output.
   b) Chart
   c) Histogram
   d) Pie
   e) Scatter
   f) Critical Success Factor
   g) Map

6) Models
   Model Components provide a way to access some type of data. The data to be accessed may be raw data, which you may permit to be modified, or metadata, data that describes other data.

<table>
<thead>
<tr>
<th>Version 8 Default Model Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Catalog Entry List Model</td>
</tr>
<tr>
<td>Catalog List Model</td>
</tr>
<tr>
<td>Color List Model</td>
</tr>
<tr>
<td>Data Set List Model</td>
</tr>
<tr>
<td>External File List Model</td>
</tr>
<tr>
<td>Library List Model</td>
</tr>
<tr>
<td>LIST Entry Model</td>
</tr>
<tr>
<td>Range Model</td>
</tr>
<tr>
<td>SAS File List Model</td>
</tr>
<tr>
<td>SLIST Entry List Model</td>
</tr>
<tr>
<td>Variable List Model</td>
</tr>
<tr>
<td>Variable Values List Model</td>
</tr>
</tbody>
</table>

7) Browsing Files
   a) Catalog Entry Viewer *
   b) External File Viewer *
   c) Video Player *

* Experimental, unsupported classes that do not appear in the default resources. To access these classes, you must add either the resource SAShelp.fsp.experimentalafcomponents.resource to the Components window, or add the desired classes to the Components window. These components may become available in Version 8.1 or Version 9.

© These components will be available in Version 8.1. Until then, you must use the Version 6 object.

* There are no Version 8 components for these objects so the Version 6 objects must be used.
Custom Classes
The Class Editor allows developers to easily create and maintain your own custom SAS/AF classes. Start by developing a simple class, such as creating an OK button from the Push Button class.

A more complex custom class that is implemented in ClinAccess is the StudyTable class. This defines a component that is used to select a Clinical Study and/or Data Table within the study. This custom composite class is illustrated below:

This custom class is instantiated in most of the Frames in ClinAccess to select either a study or table with which to work. This StudyTable class has a number of custom attributes and overridden methods. These are used to control the appearance and actions of the object in the Frame, such as controlling which studies are available for selection, checking user authorizations, performing file allocations, and placing metadata into the Local Environment List.

Required Features
There are some features common to all clinical systems.

Protections and Authorizations
Clinical data is sensitive information that should be treated as a valuable asset as well as a regulated commodity. Not only can improper management of clinical data cause the invalidation of an entire clinical trial, but it may also result in strict remedies, including serious financial penalties, from the FDA.

To avoid such costly outcomes, clinical information systems must be password protected and only allow users with proper authorizations to modify clinical data. Other user permissions should be required to perform other functions in the system.

A clinical information system should verify that a user has correctly logged into the system before performing any actions. This would prevent an unauthorized user from hacking into the system.

Each authorized user must also have permission to perform specific actions. A default user in ClinAccess may only review clinical data. Special permissions are required to

- perform data entry
- verify or correct data
- perform database administrator (DBA) functions such as study set-up
- perform system administration functions such as defining global metadata

Of course the location of the user accounts and permissions are well protected too!

Who, What, When, How
Another aspect of proper management of clinical data is the tracking of actions taken by users in the system.

The FDA has requested that entry and revision of clinical data be logged to indicate both the user performing the action as well as the date and time of the action.

The following data entry screen in ClinAccess illustrates a number of features available with a GUI based data entry screen:

The features to note are these:

- A Data Form object is used for data entry.
- A Data Table object is used to display multiple rows from a document in a spreadsheet format. In this case, there are many lab test results on the CRF, each test resulting in a row in the lab table. This allows us to enter data into a table structure that is best for analysis and reporting, while maintaining an easy interface for data entry.
- Columns may be hidden and unhidden on the spreadsheet view of the document.
- The GUI point-and-click interface has a user-accepted look-and-feel as well as expected functionality.
Audit Trail & Integrity Constraints

With Version 8 of the SAS System, you can request that an audit trail be maintained for a data table. This audit trail automatically records who, what, and when and can be instructed to collect:

- Copy of a new record
- Copy of a deleted record
- Before and/or after copies of a modified record
- Failure to delete a record
- Failure to add a record
- Failure to update a record

The failure information tracked in the SAS audit trail is not available anywhere else in the SAS System. These failures may be the result of insufficient authorization. For example, if another user has the record locked.

Add and Update failures can occur when a record is rejected because it does not meet the Integrity Constraints defined with the table. Integrity Constraints, also new with SAS Version 8, allow you to define rules that a record must meet before it can be added to a data table. For example, you can specify an Integrity Constraint that requires the PatientID variable to be non-null.

A simple example of using PROC DATASETS to initiate an audit trail and assign integrity constraints to a data table called WORK.DEMOG is illustrated below:

```sas
PROC DATASETS LIB=WORK;
  /* Initiate audit trail */
  AUDIT DEMOG;
  INITIATE;
  /* Integrity constraint to require PTID */
  MODIFY DEMOG;
  IC CREATE
    PT_REQ= NOT NULL(PITID);
  /* Integrity constraint for PTID range */
  IC CREATE
    PT_VAL= CHECK(0<PITID<101);
QUIT;
```

Rather than writing SAS code, a SAS/AF system, like ClinAccess, can use a GUI interface to gather this information from the user and generate the necessary PROC code.

Permanent User Library

Users of clinical data management systems need a place to permanently store their user preferences as well as queries, data, reports, and other programs used during the review and analysis of a study.

Database Administrator (DBA) Functionality

A Database Administrator is typically a senior user who is given expanded permissions to create and manage studies and tables.

Example of a DBA Task: New Study Setup

One of the functions of the DBA is to define new studies and tables. The following section walks through the screens that are used to perform this action in ClinAccess.

First step in the process is to define the new study to ClinAccess:

![Define New Study Screen]

Note that this screen collects data about the study to be defined as well as the location to be used to store the study data.

The user may also easily select an existing study to copy, in which case the structure of the new study is based on the structure of a selected existing study. The DBA would then select which tables from the existing study to define in the new study.

After a study has been created, the tables that contain the study’s clinical data are defined. This task uses a table definition Wizard to walk the user through the steps of creating a table.
The first step simply identifies the table and some basic parameters of the table. This screen uses a custom StudySelector component that allows users to select from among the studies defined to ClinAccess. The user also has the ability to specify an existing table to copy.

One of the table parameters in the above screen is a table Structure. The available Structure values to select are:

- One record per Patient
- One record per Visit
- Multiple records per Visit
- One record per Event
- Multiple Events

This information helps ClinAccess determine how best to display and manage the table for the user. For example, the previously shown Labs data entry screen is the type of screen used to display data from a table with a “Multiple records per Visit” table structure.

The next step of defining a new table is to identify the columns in the table. This Frame uses a dual selector component to select variables to be included in the table from a list of available variables. Note that details about a variable are displayed when a variable is selected in either selection list in the dual selector.

After specifying the columns in our table, we are prompted to identify the key and header variables in the table. A default set of key and header variables is selected by ClinAccess, based on the table structure and column selected. Users may override these default selections.

The final step in the table definition Wizard is the specification of Integrity Constraints to be applied to the columns.

The specifications on this screen are used to create the Integrity Constraints. Integrity Constraints are new with SAS Version 8. These are rules that are stored with the table definition that define acceptable data for entry into the table. If a record fails any specified Integrity Constraint it cannot be committed to the table. There is absolutely no way to get data into a table that does not meet the table Integrity Constraints.
Declarative Referential Integrity Constraints are not defined on this screen because these types of integrity constraints refer to values stored in other tables. An “Advanced” feature is available for defining these Declarative Referential Integrity Constraints.

Notice that to create all the Frames used in this DAB example, there are only 9 classes are used:
- Graphic Text
- Container Box
- Text Label
- Text Entry
- Combo Box
- Push Button
- Study Selection custom class
- Radio Box
- Dual Selector

As previously mentioned, most work can be accomplished using only a handful of the available classes. Become well acquainted with the components that you use most often in your environment.

**Conclusion**
This is an exciting time to be a clinical software developer with the SAS System. In the many years that I have been developing clinical information systems, this is the most enthusiastic I have ever been about a SAS-based solution.

Having worked in Version 8 for some time now, I find it very difficult to go back to the Version 6 SAS/AF build environment. The Version 8 environment is vastly superior in ease of use as well as the quality and appearance of your created frames.

**References**


Visit our web site for copies of these papers.

**Acknowledgements**
I must extend thanks to Dr. James Schwenke, Dr. Tom Hoffman, John Brega, Deb Gilmore, Dr. Martin Rosenberg, and all the other clinical information geniuses that I have worked with over these many years. I can see the light and it is The Clinical Information System……at last!

**About the Author**
Steve Wilson is the Director of Clinical Applications Development at MAJARO InfoSystems, Inc., a consulting and software development company for the pharmaceutical and biotech industries. MAJARO develops and markets ClinAccess™, an integrated clinical trials system developed in SAS/AF software. Visit us on the web at www.majaro.com.

Steve has been developing applications in SAS for over 15 years and has given numerous presentations at SUGI as well as regional and local conferences.
Steve can be contacted at: Swilson@majaro.com

ClinAccess is a registered trademark of MAJARO InfoSystems, Inc., Santa Clara, CA, USA. All screens shown are copyright (c) 1999 by MAJARO InfoSystems, Inc. all rights reserved.