Paper 21-27

Library and File Management: Building a Dynamic Application

Arthur L. Carpenter, Data Explorations
Richard O. Smith, Data Explorations

ABSTRACT

In order for a SAS® application to be dynamic, it must be able to automatically find the programs that make it work and the data that it is to work against. It must know where to write its output and where to store any new data sets that it creates. More importantly it must do these things without intervention on the part of the user. Even more importantly the application must not confuse the components of one project or task with those of another. The structure and organization of the libraries and directories is central to this ability.

Dynamic applications are written to be portable across projects and a key component of the application must necessarily be the structure and location of appropriate files and libraries. While many different styles of organization are possible, a high degree of organization is required. Of course you will need to know what type and how much structure is required for your projects.

This paper discusses the issues surrounding the organization of the libraries and directories associated with a dynamic application.

KEYWORDS

dynamic application, directory structure, AUTOCALL libraries, compiled stored macros, autoexec

INTRODUCTION

Applications that will be used across projects, systems, and even platforms must be written to accommodate the variety of situations that will be encountered. As a developer of the application, you will need to take into consideration a number of aspects that will need to be coordinated within the framework of the application. One of your critical issues will be to build into the application the ability to locate and use, not only the programs that make up the application, but also the data that are specific to a particular project or task within the application.
The overall structure of the directories, naming conventions used within the application, and the location of libraries and files all play pivotal roles in determining the success and maintainability of your application. You need to be able to create a logical directory structure that will be reusable for each project. Since many of the programs and macros used within the application will require dynamic project specific information, naming conventions must be established and strictly adhered to. The location of data, both project specific and data general to the overall application, must be specified and consistent. These locations usually depend heavily on the directory structure.

The examples in this paper are based on applications developed by the authors. We want to show you how we solved various problems, however these solutions are not dictates or even necessarily the best solution for your own application. We instead present these solutions because they have worked for us.

Although the examples in this paper are all based on applications written for Directory based operating systems, the concepts apply equally well to file based systems such as MVS. Also the choices of path and directory structures presented here are specific to the presented application and are not intended to be dictates for your applications.

USING THE AUTOEXEC.SAS

By default whenever the SAS System is started, it searches for a program called AUTOEXEC.SAS in the !SASROOT directory, and when present, this program is automatically executed. The AUTOEXEC.SAS is an ordinary SAS program; it can contain DATA steps, macro calls, and macro variable definitions just as can any other program. Typically AUTOEXEC.SAS is used to define the environment of the current SAS session.

Actually you can customize this process by renaming the program and/or by placing it in some other directory. Usually the program name of AUTOEXEC.SAS is not changed but its location is often changed to a project or task specific location. SAS must be pointed to the new location, and under Windows operating systems, this is accomplished through the use of the -AUTOEXEC option that is invoked at system initialization. This option is demonstrated in the example dealing with creating shortcuts below. For other operating systems you should consult your SAS Companion for more details.

Portions of a sample AUTOEXEC.SAS program are shown below. This AUTOEXEC is used to set up the environment for an application, which is then also started from within the AUTOEXEC.

In this AUTOEXEC three macro variables that will be used throughout the application are globalized ❶. These are &PATH, &TST, and &PROJECT. Each will be explained in more detail in the section on Path Control. The macro variable &PATH is used to declare the top of the directory structure ❷. Since all aspects of the application are beneath this portion of the path, this makes the application portable from machine to machine or from one place on the network to another. The macro variable &TST is used to setup a parallel test/production environment. During production it is set to null ❸. A project specific name or code is stored in &PROJECT ❹. Together these three macro variables are used to define the full directory structure down to the project level.

```
* Establish GLOBAL Macro variables for this
application;
%global tst path project; ❶

* Set up the general path for this application;
%let path = e:\clinical\bigprojs; ❷

* tst - test (tst) or production ( );
 %let tst = ;       * Production; ❸
*%let tst = tst;    * test environment;

* Project title;
%let project = XYZ; ❹
```

The three macro variables &PATH, &TST, and &PROJECT are used in the next portion of the AUTOEXEC to name the location for the macro autocall libraries ❺, which are used to store macro definitions. Autocall libraries and libraries of compiled stored macros are discussed later in this paper. The %LIBNAMES macro ❻ is used to define ALL of the application's *librefs* and *filerefs* and its definition is stored in the autocall library. The application is then started by using a DM statement ❼.

```
* Define the location for the macros;
options
noxwait xsync mautosource
sasautos=("&path\&project&tst\pgms\sasmacro" ❺
          "&path\allproj\pgms\sasmacro");

* Define Librefs and Filerefs for this project;
%libnames ❻

* Execute the application;
dm "af cat=appls.&project..userid.program" af; ❼
```

Usually you will want to use a different AUTOEXEC.SAS program for each project or task that you are working on. Under Windows this is easily accomplished by creating a shortcut for each task and by making sure that each shortcut points to its own specific AUTOEXEC program.

In the properties for the shortcut, look under the SHORTCUT tab for the TARGET: dialog box. It is here that you will find the command to execute SAS. Following the ....\SAS.EXE add the -AUTOEXEC execution option with the appropriate path to the AUTOEXEC program:

```
.....\sas.exe -autoexec
e:\clin\bigproj\xyz\pgms\autoexec.sas
```

FOLDER OR DIRECTORY STRUCTURE

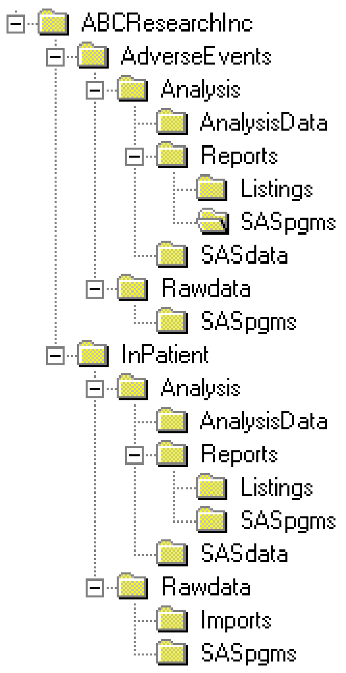
There is more than one philosophy regarding the appropriate use of folder and directory structure when setting up a project or study. Aside from the all too often encountered, 'put it anywhere-location does not really matter' philosophy known as anarchy, there are four structures that are commonly attempted. These structure types include:

- data
- flat (no discernable sub-folder structure)
- task
- project

The determining factor as to which type of structure will be chosen will be based on where in the structure hierarchy the primary or common element resides.

DATA STRUCTURE

When a large common data base is established to be used by a number of projects or tasks, the data sets themselves may drive the structure making it higher in the file hierarchy. A structure built around a data base for 'InPatient' and 'Adverse Event' information might be something like:



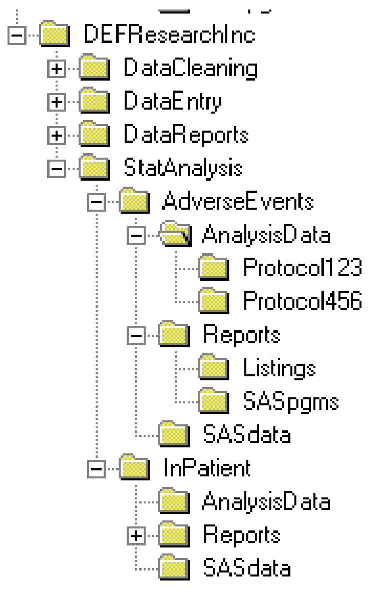
In this scheme the programs for all projects and tasks reside in directories under the type of data with which they are associated.

FLAT STRUCTURES

Flat structures minimize subdirectories by placing most data and program folders on the same level. While this structure is the least complex (nothing is hidden in sub-folders), it is not really suitable for anything other than simple projects and tasks. Because all folders for all projects are on the same level, individual projects may be harder to organize. Simple projects that make use of this structure probably do not need to take advantage of the concepts discussed in this paper.

TASK STRUCTURES

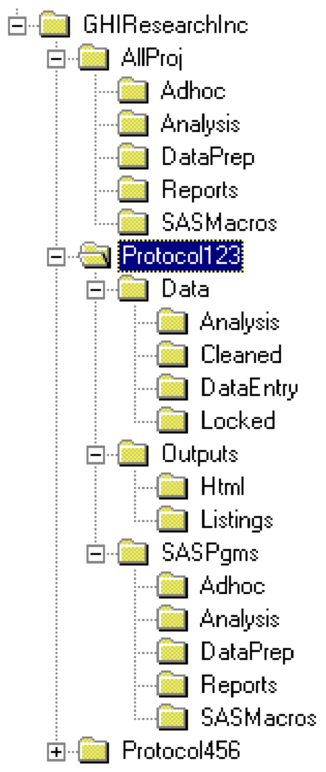
Task oriented structures are used when the task or report is foremost. This structure might be employed when a common task is applied across projects and for a variety of data sets. The task itself is the constant. A task structure might include:



```
DEFResearchInc
  DataCleaning
  DataEntry
  DataReports
  StatAnalysis
    AdverseEvents
      AnalysisData
        Protocol123
        Protocol456
      Reports
        Listings
        SASpgms
      SASdata
    InPatient
      AnalysisData
      Reports
      SASdata
```

In this type of structure the data from different studies may or may not share a common sub-directory. Usually this structure is used when the data sets are very similar and the analysis and data prep programs will work for any study under consideration.

PROJECT STRUCTURE
Since many of the applications that we have installed are project based, each with their own independent data sets and analysis programs, we most commonly employ a project oriented structure. We have found that this gives us the most flexibility when establishing applications that need to be portable and dynamic. The structure follows a hierarchy with the project highest in the directory tree. A typical structure is shown below.



```
GHIResearchInc
  AllProj
    Adhoc
    Analysis
    DataPrep
    Reports
    SASMacros
  Protocol123
    Data
      Analysis
      Cleaned
      DataEntry
      Locked
    Outputs
      Html
      Listings
    SASPgms
      Adhoc
      Analysis
      DataPrep
      Reports
      SASMacros
  Protocol456
```

We have found that some programs, especially macro tools, are used by all projects. We locate these programs and macros in the \AllProj folder, which is made available to all the projects by including it in the AUTOCALL macro libraries.

Because all information specific to a project is stored under one folder, it is very easy to move, archive, and locate information on that project. When the same structure is used consistently for each project, it becomes fairly easy to locate specific information across all projects *e.g.* the AE data in the \Analysis subdirectory for all projects. The examples shown below assume a PROJECT structure, but would apply equally to either the DATA or TASK structures.

PATH CONTROL
By creating global macro variables that can be used to identify the locations of the directory structure of interest it is possible, to construct ALL references to location (typically *libref*s and *fileref*s) to be independent of file structure except in terms of

these variables.

&PATH
This macro variable contains all of the structure above the project name. Usually this starts at the drive letter and extends as many levels as is necessary to get down to the level above the project.

&PROJECT
This is a project name or code. Since this variable may also be used in the titles of project reports or even as a constant in some of the data sets, it should be descriptive but not overly long. When the length of this variable is a problem because of competing uses, having two variables, one to hold the project code and another to hold the project label, may prove to be useful.

&TST
Often it is useful to set up two parallel structures one for testing and another for production. Since we want to use the same software for both, but only want to point to different locations, we can set up a macro variable that takes on a null value for production and some code for testing (we use TST below).

Combining these there macro variables allows us to specify the all items that change from project to project outside of the application.

```
filename critvar
"&path\&project&tst\list\gnrlrpts\critvar.lst";
libname  audit
 "&path\&project&tst\data\live\audit";
```

When in the test environment (`%let tst=tst;`) for the definitions used above, these two statements become:

```
filename critvar
"e:\clinical\bigprojs\xyztst\list\gnrlrpts\critvar.
lst";
libname  audit
"e:\clinical\bigprojs\xyztst\data\live\audit";
```

When starting a new project or when moving a project from one network drive to another, changing ALL *libref*s in all the programs is as simple as changing one macro variable definition in one place in the AUTOEXEC.SAS program. A further discussion of the definition of *libref*s and *fileref*s is included below in the section "Unified *Libref* and *Fileref* Definitions".

USING MACRO LIBRARIES

The use of macros is essential to an automated and flexible system. This implies that the control of the macro code is very important to the maintenance of the application. All to often macro definitions become buried within the programs that use them. The result is often a proliferation of multiple versions of similar macros. Parallel code, two programs or macros that do essentially the same thing, is an especially difficult problem in large applications that are maintained by multiple programmers. Even when there is only one programmer, there is a tendency to "clone with slight modifications" a program.

Macro libraries are used to avoid this problem by providing a single location for all macro definitions. Rather than cloning the macro, it is adapted (generalized) to fit each of its calling programs and then stored in one of the libraries. Obviously this requires documentation as well as diligence. Part of the solution is to place ALL macro definitions in a common library, which is accessible to all programs in the application. At least this way no macro definition will be 'buried' within a program.

There are three types of macro libraries. The least sophisticated is the use of %INCLUDE files to store macro definitions. While this solution can avoid the problems associated with duplicate code, it is usually inefficient because macro definitions that are not needed are often loaded along with the ones that are needed. The AUTOCALL and Compiled Stored Macros Libraries provide more efficient solutions.

Macros are compiled before they are executed and it is possible to store the compiled code for future use. Compiled macros are stored in a catalog called SASMACR. By default this catalog is stored in the WORK library and when a macro is called, SAS automatically searches this catalog for the compiled macro. Permanently compiled stored macros are written to a catalog with the same name, but which is stored in a different library. The library is specified using the SASMSTORE= option. Normally the options to make use of this library are turned off (NOMSTORED).

The following OPTIONS statement turns on the use of compiled stored macros and designates the two *libref*s PROJSTOR and ALLMSTOR as the locations to search for the compiled stored macro catalog. If the compiled macro is in both catalogs, the definition first found will be used (read left to right).

```
options mstored sasmstore=(projstor allmstor);
```

The compiled version of the macro is stored by including the /STORE option on the %MACRO statement. For the SASMSTORE definition above the macro AERPT below will be stored in the PROJSTOR.SASMACR catalog.

```
%macro aerpt(dsn, stdate, aelist) / store;
```

When a macro is not found in the compiled stored macro library the AUTOCALL library is searched. This library consists of macro definitions (the code) that have not yet been compiled. By default the ability to make use of the AUTOCALL facility is turned on (MAUTOSOURCE). When a macro is called, SAS searches for a file in the AUTOCALL location with the SAME name as the name of the macro. The code in the corresponding file is then submitted for processing. Since this file contains the macro definition, the macro is then compiled and made available for execution.

The following code makes sure that the autocall facility is available (MAUTOSOURCE) and specifies the *FILEREF*s of the locations (SASAUTOS=) that contain the SAS programs with the macro definitions.

```
options mautosource sasautos=(projauto allauto
sasautos);
```

Be sure that you use ***fileref*s** NOT *libref*s, and include the automatic *fileref* SASAUTOS so that the autocall macros provided with SAS will also be available.

UNIFIED *LIBREF* AND *FILEREF* DEFINITIONS

Once the library structure has been formalized it is equally important that the assignment of *librefs* (and *filerefs*) be controlled in a unified way. The first step is to never use path information instead of *libref*s and *fileref*s within the program. This means that LIBNAME and FILENAME statements will be used to define (all if possible) locations. This approach makes the programs much more portable when path information changes.

Additional control can be maintained if as many, as is possible, of the *libref*s and *fileref*s are defined at a single location within the application. We use a macro (%LIBNAMES) that contains the LIBNAME and FILENAME statements. This is an AUTOCALL macro, and it is called from within the AUTOEXEC.

A portion of the %LIBNAME macro might contain:

```
%macro libnames;
* Libnames used in applications;
libname appls ("&path\allproj\pgms\appls"
              "&path\&project&tst\pgms\appls")
               access=readonly;

* Filenames;
filename dbdirsas
  "&path\allproj\pgms\primary\dedsn.sas";
filename vrdirsas
  "&path\allproj\pgms\primary\devar.sas";
filename cmprlog
"&path\&project&tst\list\compare\compare.log";
```

```
filename cmprlst
"&path\&project&tst\list\compare\compare.lst";

* Primary Project libnames;
libname coded
  "&path\&project&tst\dictnary\live\coded";
libname notcoded
  "&path\&project&tst\dictnary\live\notcoded";
libname editlog
  "&path\&project&tst\data\live\editlog";
%mend libnames;
```

Notice that each of the paths utilizes the global &PATH macro variable and each of the project specific paths also includes the &PROJECT and &TST macro variables which were described above.

SUMMARY

Dynamic applications require a very structured approach to the organization of the data and programs. The use of macros and macro libraries provides a essential tool in the tracking of the various components, however unless you are careful when you set up the very structure of the folders and sub-folders, it becomes very difficult to coordinate the activities of the application.

How you organize your data and programs should depend on the kinds of tasks that you perform. Your application can then be written to take advantage of this structure. You will need to think about how your work is performed, and to what uses your application will be put. When the same programs are run on similar data sets across projects, a DATA driven structure may be appropriate. When the task itself drives what you will be doing e.g. various programs on the AE data, you may want to take a TASK approach to the organization. PROJECT structures apply when each project or protocol is unique or distinct and tasks within the project vary.

The use of the AUTOEXEC and of macro libraries (Compiled/Stored or AUTOCALL) will greatly assist in the automation of your application.

Dynamic applications are automated applications. They require planning and forethought, but the rewards are tremendous gains in efficiency and organization.

ABOUT THE AUTHORS

Richard Smith and Art Carpenter are senior partners at Data Explorations. Data Explorations, a SAS Quality Partner™, provides data management, analyses, and SAS programming services nationwide.

Arthur L. Carpenter

Art Carpenter is a SAS Certified Professional™. His publications list includes three books on SAS topics (*Annotate: Simply the Basics*, *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language*), two chapters in *Reporting from the Field*, and numerous papers and posters presented at various user group conferences. Art has been using SAS since 1976 and has served in a variety of positions in user groups at the local, regional, and national level.

Richard O. Smith

Richard Smith has a masters in Biology/Ecology and has provided complete data management and analysis services for numerous environmental research projects as a senior biologist, SAS programmer, and project manager. He has also provided programming and management services for the health related industries. He has been using SAS extensively since 1981.

AUTHOR CONTACT

Data Explorations
2270 Camino Vida Roble, Suite L
Carlsbad, CA 92009

Arthur L. Carpenter
(760) 945-0613
art@caloxy.com

Richard O. Smith
(760) 438-1336
ROSmith@SciX.com

REFERENCES

Burlew, Michele M., *SAS® Macro Programming Made Easy*, Cary, NC: SAS Institute, Inc., 1998, 280 pp.

Carpenter, Arthur L., *Carpenter's Complete Guide to the SAS® Macro Language*, Cary, NC: SAS Institute Inc., 1998, 242 pp.

Carpenter, Arthur L. and Richard O. Smith, "Clinical Data Management: Building a Dynamic Application", *Proceedings of the PharmaSUG Annual Conference*, 2000. Also published in the *Proceedings of the 7th Annual Conference of the Western Users of SAS Software*, 2000, pp3-8.

SAS Institute Inc., *SAS® Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc., 1997, 304 pp.

TRADEMARK INFORMATION