ABSTRACT
This paper discusses an interactive SAS macro that creates a standard directory structure for use with SAS programming related projects. The macro provides a generic directory structure framework for any project, and allows users to provide their own project-specific directory names where appropriate within the framework. It then issues the required operating system level commands to actually create the desired directories on the disk subsystem. The benefits of utilizing such a macro, parts of the SAS code for the macro, and a possible directory structure framework with application towards the pharmaceutical industry are discussed. The use of the %WINDOW statement of the SAS Macro language to achieve the user interface is also discussed. SAS processing platforms targeted are Win/NT and UNIX.

INTRODUCTION
As part of an initiative to increase the efficiency of SAS programming tasks for filings to regulatory agencies, a programming development team determined that a standard directory structure would be beneficial for the statistical analysis of clinical trial data. A tool was conceived to automate the creation of the directory structure since the adopted structure contained numerous subdirectories and would be produced many times. This tool, a SAS macro, would generate the standard structure thereby promoting and facilitating its use. It would also address the problems of tedious, repetitive work and human error. A discussion of the programming strategy behind this macro and some important pieces of its code will be presented.

THE STANDARD DIRECTORY STRUCTURE
The programming development team decided that the standard directory structure would be individually implemented for each drug project. Thus, the compound name or drug name would be utilized as the root of a directory hierarchy. This hierarchy would form a unique file system for that drug project. Such an arrangement allows a project team to have a clearly defined repository for all of their work. In addition, it was decided that the SAS programs within this project repository should be drive/directory independent to allow easy transference to the computing systems of regulatory agencies.

Since preparations for an NDA, WMA, etc. could involve numerous files, each file system for a drug project would additionally have individual directories for submissions to regulatory agencies. Directory naming at this level would indicate the filing. Separate storage such as this was deemed beneficial for easy identification and retrieval of related work.

Breaking the framework down even further, there would also be distinct file subsystems within the submissions for each clinical study (ie. protocol) involved. These subsystems would contain directories for SAS programs, SAS data sets, and the different types of outputs (ie. logs, listings, graphics, tables).

Additional common directories below the submission level, but parallel to the protocol level were included for the purpose of sharing SAS formats, graphics devices, and macro programs between clinical studies. Sharing of files between submissions however, was not encouraged since they were considered to be separate entities. Therefore, no common directories were placed under the drug project directory.

By the time the standard structure had become theoretically complete, it included 14 distinct directories for a file system containing just one protocol. Each subsequent protocol would add another 10 directories. To manually create each of these directories would be tedious and time consuming. In addition, all directories except those for the drug project, submission, and protocol had required names. Manual creation of the directories would introduce the possibility of typos and other errors. Considering the fact that the entire process would need to be repeated many times, once for each drug project, an automated procedure for creating the standard directory structure was deemed necessary.

THE STANDARD DIRECTORY STRUCTURE MACRO
One basic requirement for a tool that would automate the creation of the standard directory
structure was a simple and functional user interface. Project team members would need to input the Win/NT drive or top level UNIX directory upon which to create the structure. They would also need to supply the names for the drug project, submission and protocol directories. A SAS program written in macro format that made use of %WINDOW statements seemed to be a suitable approach since it did not involve an additional software tool such as SAS/AF or an outside package such as Visual Basic in the development of the application. Use of %WINDOW allowed input to be easily translated into macro parameter values for production of the directory structure.

A Win/NT example of a SAS input window created with %WINDOW is shown below in Figure 1:

![Figure 1: Sample SAS Input Window Created with %Window](image)

The black rectangles are fields awaiting user input. The inputted values will become the values of macro parameters for use throughout the rest of the program. The code that produces the above window appears below in Figure 2:

```
%window screen1 color=gray
#1 @8 'Standard Directory Structure Set-Up Program' 'for Windows NT'
#2 @6 '---------------------------------------------'
#4 @ 3 'Enter the letter for the drive upon which you would like to create the standard directory structure (example: U):'
#5 @ 3 'standard directory structure (example: U):'
+2 drive 1 attr=rev_video autoskip=no
#8 @ 3 'Enter directory names for the drug project, submission, and protocol.' '
submssn 50 attr=rev_video autoskip=no
#10 @ 3 'Enter only one protocol. You will be prompted for additional protocols later.' '
submssn 50 attr=rev_video autoskip=no
#13 @12 project 50 attr=rev_video autoskip=no
#15 @ 5 '2) Submission (Example: NDA)'
#17 @ 5 '2) Submission (Example: NDA)'
#19 @ 5 '3) Protocol (Example: PROT001)'
#20 @ 8 'Enter only one protocol. You will be prompted for additional protocols later.' '
#22 @ 12 prot 50 attr=rev_video autoskip=no
#25 @ 3 'Enter N then press ENTER to continue to next screen, or C then ENTER to cancel.' '
#26 @ 3 'Enter N then press ENTER to continue to next screen, or C then ENTER to cancel.'
enter1 1 attr=rev_video color=red;
%display screen1;
```

Figure 2: %WINDOW Code to Produce Sample Input Window

Note that due to space constraints within this paper, quoted lines that exceeded the allowable length were wrapped and then indented for clarity. A second set of quotes was applied to the wrapped line to avoid unnecessary spaces. Within the SAS editor, the entire definition for row #1, for example, could be placed on a single line with one set of quotes.

In this example, the %WINDOW statement will be defining a window called SCREEN1 whose overall characteristic color is gray. The subsequent lines of code will determine the appearance and content of the rows presented on the interface. The line beginning "#1 @3 ..." will define the content of the first row of the interface. "#2 @2 ..." will define the second and so on. If a row number is skipped, a blank line will be inserted into the interface. In the example above, a definition for row 3 is omitted, thereby inserting a blank line between row 2 and row 4.

Within the individual row definitions, the @ symbol is used to determine the starting column position for printing characters on the interface. Quotes are used to print text strings. Macro parameters to be printed are left unquoted. Since the PROTECT=YES option has not been specified for the macro parameters above, DRIVE, PROJECT, SUBMSSN, and PROT can all receive input for their values. They will then become the macro parameters for drive letter, drug project, submission and protocol, respectively. In order to make their input fields more visible, the field lengths and attributes have been specified, thus producing the black rectangles that are visible in the sample window above.

The window named SCREEN1 is presented to the user by the last line of code in Figure 2. The %DISPLAY command names a window previously defined by %WINDOW to be displayed.

```
Figure 1: Sample SAS Input Window Created with %Window
```

```
%window screen1 color=gray
#1 @ 8 'Standard Directory Structure Set-Up Program' 'for Windows NT'
#2 @ 6 '---------------------------------------------'
#4 @ 3 'Enter the letter for the drive upon which you would like to create the standard directory structure (example: U):'
#5 @ 3 'standard directory structure (example: U):'
+2 drive 1 attr=rev_video autoskip=no
#8 @ 3 'Enter directory names for the drug project, submission, and protocol.' '
submssn 50 attr=rev_video autoskip=no
#10 @ 3 'Enter only one protocol. You will be prompted for additional protocols later.' '
submssn 50 attr=rev_video autoskip=no
#13 @12 project 50 attr=rev_video autoskip=no
#15 @ 5 '2) Submission (Example: NDA)'
#17 @ 5 '2) Submission (Example: NDA)'
#19 @ 5 '3) Protocol (Example: PROT001)'
#20 @ 8 'Enter only one protocol. You will be prompts for additional protocols later.' '
#22 @ 12 prot 50 attr=rev_video autoskip=no
#25 @ 3 'Enter N then press ENTER to continue to next screen, or C then ENTER to cancel.' '
#26 @ 3 'Enter N then press ENTER to continue to next screen, or C then ENTER to cancel.'
enter1 1 attr=rev_video color=red;
%display screen1;
```
ISSUING OPERATING SYSTEM COMMANDS FROM SAS

In order to physically produce the standard structure, the macro must create the directories on the disk subsystem. Since this is beyond the scope of the Base SAS language, execution of operating system commands is required. Methods for issuing operating system commands within SAS include using an X command, a %SYSEXEC macro statement or a CALL SYSTEM routine. All three methods allow the user to issue any command available on the host system. As an example, the use of the X statement within a DATA _NULL_ is shown in Figure 3 below. In this sample code, the system command is placed within quotes after the X.

```
data _null_;  
x "md &drive\&project\&submssn\macrolib";  
x "md &drive\&project\&submssn\utility";  
x "md &drive\&project\&submssn\&prot";  
x "md &drive\&project\&submssn\&prot\data_analysis";  
x "md &drive\&project\&submssn\&prot\macrolib";  
x "md &drive\&project\&submssn\&prot\out_graphics";  
x "md &drive\&project\&submssn\&prot\out_listsings";  
x "md &drive\&project\&submssn\&prot\out_logs";  
x "md &drive\&project\&submssn\&prot\out_tables";  
x "md &drive\&project\&submssn\&prot\pgm_adhoc";  
x "md &drive\&project\&submssn\&prot\pgm_analysis";  
x "md &drive\&project\&submssn\&prot\utility";  
run;
```

Figure 3: Sample SAS Code Issuing Win/NT Operating System Commands

Through use of the above code, the SAS macro will create a Win/NT file system based on the standard structure. The MS-DOS command MD will create the directories specified. Note the use of the macro parameters &DRIVE, &PROJECT, &SUBMSSN and &PROT. The values for these parameters are those received from the input for drive letter, drug project, submission and protocol, respectively, from SCREEN1 above. Some manipulation within the macro code will be required to ensure that the drive letter will resolve with the colon in its appropriate place.

An example for the UNIX operating system is show in Figure 4:

```
data _null_;  
x "mkdir &root/&project";  
x "mkdir &root/&project/submssn/macrolib";  
x "mkdir &root/&project/submssn/utility";  
x "mkdir &root/&project/submssn/prot";  
x "mkdir &root/&project/submssn/prot/data_analysis";  
x "mkdir &root/&project/submssn/prot/macrolib";  
x "mkdir &root/&project/submssn/prot/out_graphics";  
x "mkdir &root/&project/submssn/prot/out_listsings";  
x "mkdir &root/&project/submssn/prot/out_logs";  
x "mkdir &root/&project/submssn/prot/out_tables";  
x "mkdir &root/&project/submssn/prot/pgm_adhoc";  
x "mkdir &root/&project/submssn/prot/pgm_analysis";  
x "mkdir &root/&project/submssn/prot/utility";  
run;
```

Figure 4: Sample SAS Code Issuing UNIX Operating System Commands

In this case, the operating system command MKDIR is utilized for generating all of the directories within the standard structure. Since the concept of a drive letter is no longer valid in UNIX, the macro parameter &ROOT for root directory is used instead of &DRIVE as in the previous example. Another difference from Win/NT is the use of forward slashes instead of back slashes. This allows consistency with host system requirements.

DRIVE/DIRECTORY INDEPENDENCE

Drive and root directory independence were determined to be important criteria for the standard directory structure. This independence implies that SAS programs contained within the structure would not be tied to any specific drive letter or root directory, thus allowing easy transference to a different drive or computing system (e.g. that of the FDA). Instead, references to physical location within the project programs would be resolved at run-time through use of macro parameters defined in a protocol-specific file that was termed an initialization program. This initialization program would be run just prior to all of a protocol's analysis programs.

The most important aspect of the initialization program is the line of code that defines the macro variable DRIVE (see Figure 5 below). This is the one and only piece of code that will need to be changed for each protocol when the project programming is moved to a different physical location, provided that the directory structure in the new location is the same as that in the original location. The %LET statement for the definition of &DRIVE may contain a drive letter and/or root directory depending on the situation. Macro parameter references for all of the data libraries and directories within the standard structure are then defined using &DRIVE (see Figure 5 below). These macro references will be used within the SAS programs instead of hard-coded directory and library
references. This method allows the whole structure to be moved to a different drive or computing system without requiring numerous and burdensome changes to all of the SAS programs. Only a simple change to the %LET DRIVE= statements of the initialization programs would need to be made.

As an added benefit, it was decided that the standard directory structure macro should create the initialization file for each protocol as it creates the protocol's directories. The name chosen for the initialization program was STARTUP.SAS. It would be automatically placed in the protocol-level UTILITY directory. Shown below in Figure 5 is an example of STARTUP.SAS where the drug project is FC9999 (a fictitious compound number), the submission is an NDA, and the protocol number is 001:

```sas
/*****************************************************/
/* Program:  STARTUP.SAS                         */
/*                                                 */
/* Function:  This program creates macro variables  */
/* that represent standard directories. In addition,*/
/* SAS librefs are created for the data directories.*/
/*                                                 */
/* Drug project:  FC9999                         */
/* Submission:    NDA                            */
/* Protocol:      PROTO001                       */
/*                                                 */
/* Date:  10DEC99                                */
/*****************************************************/
%let drive = u:;
%let subpath  = &drive\FC9999\NDA;
%let protpath = &drive\FC9999\NDA\PROTO01;
%let datadir  = &protpath\data_analysis;
%let macdir   = &protpath\macrolib;
%let graphdir = &protpath\out_graphics;
%let listdir  = &protpath\out_listings;
%let logdir   = &protpath\out_logs;
%let tabledir = &protpath\out_tables;
%let adhocdir = &protpath\pgm_adhoc;
%let pgmdir   = &protpath\pgm_analysis;
%let utildir  = &protpath\utility;
%let smacdir  = &subpath\macrolib;
%let sutildir = &subpath\utility;
libname datadir '&datadir';
```

Figure 5: Sample Initialization Program, STARTUP.SAS

This initialization program would be run in an interactive SAS session before any other programs for protocol 001 are submitted. If batch programming is preferred, a %INCLUDE of this file could occur at the beginning of all protocol programs.

SAS code for producing an initialization program from the standard directory structure macro appears below in Figure 6.

```sas
filename initsas
"&drive\&project\&submssn\&prot\utility\startup.sas";

data _null_;
  file initsas;
  put '="/**************************************************/
   "***********************************";
  put "% Program:  STARTUP.SAS%";
  put "%**%";
  put "% Function:  This program creates macro variables that represent standard directories. In addition, SAS librefs are created for the data directories.";
  put "%**%";
  put "% Drug project:  &project%";
  put "% Submission:    &submssn%";
  put "% Protocol:      &prot%";
  put "%**%";
  put "% Date:  &sysdate%";
  put "="/**************************************************/
   "*/";
  put "%let drive = ' '&drive';%";
  put;
  put "%let subpath = &drive\&project\&submssn%;%";
  put "%let protpath = &drive\&project\&submssn\&prot%;%";
  put;
  put "%let datadir = &protpath\data_analysis%;%";
  put "%let macdir = &protpath\macrolib%;%";
  put "%let graphdir = &protpath\out_graphics%;%";
  put "%let listdir = &protpath\out_listings%;%";
  put "%let logdir = &protpath\out_logs%;%";
  put "%let tabledir = &protpath\out_tables%;%";
  put "%let adhocdir = &protpath\pgm_adhoc%;%";
  put "%let pgmdir = &protpath\pgm_analysis%;%";
  put "%let utildir = &protpath\utility%;%";
  put;
  put "%let smacdir = &subpath\macrolib%;%";
  put "%let sutildir = &subpath\utility%;%";
  put;
  put 'libname datadir ' ' &datadir ' '' ;%";
  put;
run;
```

Figure 6: Sample SAS Code For Producing an Initialization Program

Again, note that due to space constraints within this paper, lines exceeding the allowable space were wrapped and then indented for clarity. For quoted strings that were wrapped, an additional set of
quotes was applied to the second line to avoid unnecessary spaces. Within the SAS editor, an entire PUT statement could be placed on a single line with one set of quotes.

Within the sample SAS code of Figure 6, there is a point worth discussing for clarification purposes. It concerns the following line of code:

```
put '%let drive = ' "&drive:;";
```

The macro parameter &DRIVE on the right-hand side of the equal sign will be resolved when the standard directory structure macro is run. The value of this parameter will be the input for system drive that the user specified in the first screen of the macro (refer to Figure 1). In the sample initialization program of Figure 5, the value for &DRIVE resolved to "u" as demonstrated by the following line:

```
%let drive = u:;
```

The macro parameter &DRIVE on the left-hand side of the equal sign will be used within the project programming to specify the physical location of the standard structure.

**CONCLUSION**

A SAS macro that will create a standard directory structure and an initialization program can be extremely useful for project organization and setup. Use of the X command within a DATA_NULL_step allows execution of operating systems commands for the purpose of building the structure's directories. %WINDOW from the SAS Macro Language provides a convenient method for building a simple and functional interface for such a macro due to its availability within Base SAS software. With basic changes to operating system commands and directory format, this macro could be easily transported between a Win/NT system and a UNIX system.

**TRADEMARKS**

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® Other brand and product names are registered trademarks or trademarks of their respective companies.