PharmaSUG 2012 - Paper IB04

Could Have, Would Have, Should Have!

Adopting Good Programming Standards, Not Practices, to Survive An Audit.

Vincent J. Amoruccio, M.A., Alexion Pharmaceuticals, Inc., Cheshire, Connecticut

ABSTRACT

The primary purpose of a Pharmaceutical is to develop and market drugs which treat or prevent disease safely and efficaciously. A major step in the licensing of a drug, in particular in the United States, is the inevitable audit by the FDA. While the FDA suggests that adherence with Good Clinical Practices ("GCP") is a critical requirement, it falls short in providing programming standards for the SAS® deliverables of a submission. SDTM and ADaM are solutions to standardizing the review of data, but not the programs themselves. The lack of regulations leaves programmers unmanaged and exposed to risk when asked to deliver SAS® Programs to the FDA during a submission. While many programmers are addressing this through groups, papers, websites, and blogs there are no formal Good Programming Practices ("GPP"). Until there is, programmers must create, manage, and defend their own programming choices during an audit. It is not enough to establish programming 'practices' since auditors only care about what was done rather than what could have, would have or should have been done. This paper will first discuss the importance for GPP and common practices appearing in current GPP discussions. It will then discuss the difference between practices and standards and suggest ways to select practices to manage as standards. It will suggest ways to check and document compliance with Good Programming Standards ("GPS"), not GPP, and prepare for a successful FDA Audit. Finally, it will end with a call to the FDA for established programming standards.

INTRODUCTION

Pharmaceuticals are in the business to make and sell drugs that treat and prevent diseases all over the world. The Food and Drug Administration protects the public health of the U.S. by assuring the safety, efficacy, and security of human and veterinary drugs and products. Similar agencies exist in other countries such as The Medicines and Healthcare products Regulatory Agency ('MHRA') in the United Kingdom. Each country has different but similar requirements for submitting applications to make and sell drugs. This paper will focus its discussion on the U.S. and submitting data to the FDA.

The FDA resigns itself from promoting any specific vendors such as SAS®. Despite the FDA's neutral stance on software, SAS® is the predominant software being used to collect, create, manipulate, analyze, and submit data to the FDA. Many other software languages have been developed and are being used to analyze data for drug applications. The most common are R, S-Plus, and SPSS. I have specifically written this paper to discuss SAS® Programming however, I have used generalized terms such as 'scripts', 'programs', and 'programmer' in place of 'SAS® code', 'SAS® programs' and 'SAS® Programmers' because I believe that many of the conventions discussed in this paper should be applied to all programming languages.

This paper is intended to address the lack of guidance by the FDA to regulate how programs are written. The FDA provides general guidance and suggestions on how companies should conduct clinical trials. While most of these guidelines and suggestions have been adopted as 'the rules' to conduct clinical trials, these rules fail to provide oversight of programming scripts. Over the last few years, many programmers have begun to address this gap privately. Each of them discussed similar programming conventions however, they all seemed to lack authority. The Pharmaceutical industry needs the FDA to enforce GPS, not GPP.

History and Role of GxP

GxP is a commonly used acronym used to define quality guidelines and regulations in many fields such as Accounting, Agriculture, Business, IT, Laboratory, Manufacturing, Pharmaceutical, Service, and Tourism. The middle letter of the acronym, 'x', usually characterizes the area of regulation. Good Clinical Practices and Good Clinical Data Management Practices are two of the most common GxP in the Pharmaceutical industry.

While the FDA is the main body of human protection in the United States, many GxP standards have been adopted as 'law' by other countries. In the early 1990s, the 'International Conference on Harmonization' ('ICH') brought together the drug regulatory authorities and the pharmaceutical industry of Europe, Japan and the United States. The ICH has become the international government over standards for clinical trials involving human subjects. One of its primary goals is to ensure global safeguards on quality, safety, efficacy, and regulatory obligations to protect public health[1].

Good Clinical Practices ("GCP")

Good Clinical Practices ('GCP') is the most common standard used in the Pharmaceutical industry. This is mostly due to the ICH's adoption of these practices in 1995. By ICH definition, 'GCP' is the standard for the design, conduct, performance, monitoring, auditing, recording, analyses, and reporting of clinical trials that provides assurance that the data and reported results are credible and accurate, and that the rights, integrity, and confidentiality of trial subjects are protected. As a result, GCP serves to assure the protection of human rights in clinical trials and the safety and efficacy of drug compounds. It specifically provides guidance for the Investigator, the Sponsor, the Clinical Trial Protocols, Amendments, and the Investigators Brochure.

Good Clinical Data Management Practices ("GCDM")

Clinical data management ("CDM") involves the entry, validation, quality assurance, and quality control of all data gathered about a subject's involvement in a clinical trial. CDM activities also include Case Report Form ("CRF") design, user acceptance testing, data entry, data validation, logical edit checks, central laboratory data, coding adverse events, and coding medications. The FDA and ICH provide guidance over a limited set of data management practices. Section 2 of the ICH Efficacy Guideline regulates the management of Safety Data which is specific to the collection and reporting of Adverse Drug Reactions and Events.

Most recently, the FDA released draft guidance for Electronic Source Documentation in Clinical Investigations. They provide suggestions for data elements, modifications and correction, repeat appearance, electronic prompts, originators of data elements, and identification of data originators.

Additionally, the Society for Clinical Data Management created a comprehensive document for accepted practices that are not covered by the FDA. Given the broad spectrum of CDM and the lack of regulation over this spectrum, the SCDM's GCDMP addresses the continuing need for guidance about CDM. The Guidelines of the FDA and ICH, along with those of the SCDM are commonly referred to as Good Clinical Data Management Practices ("GCDMP")

Current FDA Requirements for Programming

Programming is a robust field encompassing many well-respected computer languages such as C, HTML, Java, Perl, R, SAS® and other languages. All of these may or may not have a place in the Pharmaceutical industry, but SAS® has a specific role in submitting data to the FDA. In January 1999, the FDA's Center for Drug Evaluation and Research (CDER) and Center for Biologics Evaluation and Research (CBER) issued guidance for electronic submissions, entitled 'Providing Regulatory Submissions in Electronic Format - General Considerations'. This guidance describes the use of the SAS® XPORT format for electronic data sets[2]. This guidance suggests the use of SAS® as the programming language to be used to submit data to the FDA.

In July, 2004, the FDA issued a draft guidance, which was finalized in November 2005 that specified a new submission structure based on the electronic Common Technical Document (eCTD) format originally defined by the International Council for Harmonization (ICH). SDTM was recommended as the "preferred" submission standard. In December, 2006, FDA published a proposal in the Federal Register that would mandate the use of SDTM for all Data Tabulation submissions with a two-year transition period[3].

In December 2011, CDER released an update to its Common Data Standards Issues Document. In it, sponsors were encouraged again to submit data in a standard form. CDER has been accepting SDTM datasets for 8 years. It addresses the mistake many sponsors make by not submitting analysis datasets. CDER suggests the "preferred" model for analysis datasets is the Analysis of Datasets Model (ADaM).

Shortcomings of FDA Requirements

SAS® is an integrated system of software products. Programmers in the Pharmaceutical industry use SAS® predominantly for data retrieval, management, mining, analysis, reporting, and graphing. SAS® allows programmers to access data across multiple platforms and transform data into usable output. There are hundreds, if not thousands of conventions to manipulate and transform data using SAS® however, there is no FDA or ICH guidance on programming.

ICH GCP specifically defines itself as 'A standard for the design, conduct, performance, monitoring, auditing, recording, analyses, and reporting of clinical trials that provides assurance that the data and reported results are credible and accurate, and that the rights, integrity, and confidentiality of trial subjects are protected'. So where does SAS® fit in? Specifically, what part of ICH governs how programs are written to report and analyze the data?

Programmers have to comply with GCP like everyone else and depending on what area of the Pharmaceutical a programmer works in, there may or may not be further guidance on how to proceed. Currently, there is virtually no guidance over specific programming conventions used to create output for the FDA. This is quite detrimental to programmers and pharmaceutical companies.

GOOD PROGRAMMING PRACTICES

What does GPP mean to you?

As previously described, GxP is commonly used to define quality guidelines and regulations. The two most common GxP are GCP and GCDMP. These two GxPs have been defined by the FDA and ICH as 'the rules' to follow when submitting to the FDA and most other International Agencies. GPP is not covered by the ICH nor is it covered by the FDA.

Over the past few years, the need for GPP has been identified by programmers in the Pharmaceutical industry. The contributions from "sasCommunity.org" are the most comprehensive and notably useful resources for GPP I have seen. This organization of SAS® users released draft recommendations for good programming practice for analysis, reporting and data manipulation in clinical trials and the healthcare industries[4]. In 2010 they formed a GPP Steering Board ("The Steering Board") "to lead the creation of documented Good Programming Practices, to make this publicly available free of charge and to promote and publicize its use within the healthcare industries". The Steering Board is the first organized effort I have seen to create an enforceable list of programming conventions for all programmers.

Other efforts to bring about GPP have come from independent contributors such as myself. The following sections summarize my ideas, and the ideas of other GPP contributors, into common themes regarding programming practices. After discussing each practice, I will provide my suggestions to either implement it as a standard or maintain it as a practice.

Common GPP Themes

The promotion of GPP is not exhaustive, particularly when restricting the information to SAS® programming. Distinct themes can be identified through all of the GPP contributions. All of the contributors implemented their own style and personality but the same practices appeared repeatedly. All authors specifically discussed topics about regulatory requirements, readability, portability, data integrity and efficiency.

Regulatory Requirements

The Steering Board has an unfinished section for Regulatory Requirements and lists two subsections: Validation and 21 Part 11 Compliance. While I completely agree that these two concepts are vitally important and perhaps critical, I believe they fail to stay within the scope of GPP. SAS® technology is not robust enough to provide conventions which would make programs and output fully compliant with 21 CFR Part 11. There are some great contributions available about this topic and think they should remain to be discussed separately.

This section will aim to discuss programming standards to make programs 'submission ready'. In other words, we will discuss key requirements for making programs and their output ready to be submitted to the FDA at any time.

Readability

Readability is the quality of the program which makes it easy to read and understand. Before a script is written the author must consider the different levels of understanding by all potential users. Programs must be written in such a way that any programmer or non-programmer can read and understand the program. Readability also directly affects a program's maintainability. The more readable a program, the better it will be understood. The more comprehensible it is, the easier it will be to update and maintain.

Portability

Software portability is the ability to use software across multiple platforms. The portability of a program has a slightly different meaning. At Alexion, the directory structure reflects a standard hierarchy. Programs are written at global, drug and protocol levels and are compatible across each level. A program written for a protocol or project within Drug A can easily be crossed over to a protocol in Drug B. Likewise, a program written for Protocol X within Drug A can be easily used for Protocol Y within Drug A. Using these two examples, we can simply define portability as the ability to use programs across multiple trials and drugs.

Data Integrity

All programs must be trustworthy enough to produce output that is accurate and consistent regardless when, where, and how they are run. There are specific tasks programs can be written to perform that will ensure they execute successfully.

Efficiency

There are many definitions of efficiency. The one which best describes efficiency with regards to programming is "the ratio of the work done to the energy or effort supplied to it". All of the contributors had suggestions for efficient programming and as expected there were distinct commonalities amongst them.

Problems with GPP

The major problem with GPP is the lack of authority from any governing agency such as the ICH or FDA. While all of the contributors have discussed common interests, they are only suggestions or guidelines for what could be done or should be done.

The FDA requires data to be submitted through Version 5 compliant transport files. It strongly suggests the use of SDTM and prefers the use of ADaM. These current requirements focus on the structure and transmission of data, but neglect the design of programs.

Programmers have their own style of programming and this can lead to issues with regulation, readability, portability, and efficiency. This will become problematic if the Pharmaceutical industry tries to adopt standard scripts. In fact, initiatives have already begun to start sharing script within our industry. After nearly 10 years of standardization through SDTM, the growing acceptance of ADaM and the increase need to facilitate the FDA's review of drug applications, there has been a recognized need for standardized scripts to be shared across the Pharmaceutical industry. Sharing is an extraordinarily great initiative, not to mention a much needed one. However, this increases the need for Programming standards if programmers from around the world are going to share scripts.

Moreover, I think it is not only in the FDAs best interest, but the Pharmaceuticals as well to globally adopt programming standards. Doing so ensures consistency and assurance that key programming issues are being addressed by all companies.

ADOPTING GOOD PROGRAMMING STANDARDS

The difference between a practice and a standard

The words 'practice' and 'standard' are two words which have been incorrectly used in the Pharmaceutical industry. They have evolved into synonymous definitions and seem to be used interchangeably. I disagree with this convention and believe they should be used distinctly. In the scope of Clinical Trials, I interpret these two words as something you 'should do' versus something you 'must do'. Practices are suggestions or recommendations. They use words such as 'should', 'could', 'would', and 'might'. They lack authoritativeness and are less enforceable then standards. Standards are directive. They explicitly tell you what you can or can't do and compel obedience.

Accomplishing GPS

There are three steps to select Good Programming Practices and change them into Good Programming Standards ("GPS"). The first is to identify realistic conventions which are common amongst all programmers. The second is to determine which conventions are manageable. The last step is to determine which of these conventions are auditable. In other words, an enforceable convention must be something we can reasonably force ourselves and be accountable to follow.

There are thousands of programming conventions we can follow, but not all are realistic to follow or manageable to enforce. It's plausible to enforce all of them but it becomes unrealistic because of the undue hardship it would have on us to document our compliance to regulatory agencies such as the FDA. Later in this paper we will discuss the importance of documentation.

At Alexion, we selected a core set of programming conventions we felt were realistic, manageable, and auditable. We developed documentation which was easy to use and demonstrated compliance.

In the following sections I have accumulated conventions identified throughout all GPP contributions. I agree with the majority of the programming conventions discussed in GPP articles however, I specifically identify which conventions should be practices or standards with explanations.

REGULATORY REQUIREMENTS

SAS® Version 5 Compliant Datasets

In the General Considerations for Providing Regulatory Submission in Electronic Format, the FDA specifically states it wants SAS® transport files (XPORT) created by PROC XCOPY. The more granular detail of this guidance specifies the datasets must be in SAS® Version 5 ("V5") format. As a result, it is critical to understand what the V5 format is in order to ensure compliance.

The V5 format requires the following:

1. Dataset annotations less than or equal to 8 characters.

2. Dataset labels less than or equal to 40 characters.
3. Variable names less than or equal to 8 characters.
4. Variable labels less than or equal to 40 characters.
5. Variable length less than or equal to 200 characters.
6. Format names, including the dollar sign symbol '$', less than 8 characters.

Practice or Standard? Standard. Be prepared! Program everything as if you are submitting it to the FDA, even if you are not. Doing so will avoid the inconvenience of re-writing your programs.

Use Analysis Datasets (aka: ADS)

CDER's Common Data Standards Issues Document publicly addresses the FDA's position on analysis datasets. This guidance specifically suggests submitting datasets following ADaM. Through personal experience, I have witnessed instances where the FDA explicitly requested "the analysis datasets used as the source for the safety and efficacy tables, listings, and figures".

Practice or Standard? Standard. CDER states "Analysis files are critical for the FDA to understand on a per patient basis how the specific analyses contained in the study report have been created." Creating analysis datasets, whether sponsor defined or ADaM must be a standard. I also believe creating analysis datasets helps avoid unexplainable differences in output. At Alexion, all variables passed into a statistical procedure must come from an Alexion ADS or ADaM.

All programs and output must be ICH eCTD Compliant

The ICH eCTD is an interface used for industry to agency transfer of regulatory information. It takes into consideration the facilitation of the creation, review, life cycle management and archiving of the electronic submission. The ICH Electronic Common Technical Document Specification describes the requirements for submitting electronic data to the FDA. Programmers should be most concerned with Appendix 2 (The eCTD Submission). Appendix 2 states the following guidance must be followed in order to successfully submit data electronically:

1. The maximum length of the name of a file name may not exceed 64 characters including the extension.
2. Only lower case letters may be used in the file name.
3. No spaces may be used in the file name.
4. Only letters ("a" to "z"), digits ("0" to "9"), and hyphen ("-") may be used in the file name.

Practice or Standard? Standard. On the agenda for 2012, the FDA will release an update to its general considerations about Providing Regulatory Submission in Electronic Format. It is anticipated that this update will retire paper submissions and require electronic submissions. It is prudent to employ this standard across all programming to reduce the chance of modifying programs that weren't intended to be submitted.

READABILITY

Language

The Steering Board suggests the use of the English language as the primary language for scripts, comments, and output. In most countries, English is considered the premier language even though it may not be the official language. This ubiquitous language is more often referred to as the 'Universal' language. As a result it should be used uniformly in scripts, comments, and outputs. Although programmers might be fluent in other languages or even work for companies where English is not the primary language, it must be used in programming to ensure all scripts and output can be understood throughout the world.

Practice or Standard? Standard. Writing scripts in a universal language not only increases readability, but also ensures portability because the program will be understood by all reviewers.

Programmers should inspect all programs and output for non-English language in dataset names and variable attributes. Imagine a regulator reviewing the contents from the sample dataset below. The output from this CONTENTS procedure provides a great example of non-English language in output. Any reviewer would have a difficult time understanding the dataset name and variable attributes unless they had an understanding of German language.

```
                                The CONTENTS Procedure

Data Set Name  SOURCE.MODUL_UNTERSCHRIFT                Observations         3385
Member Type    DATA                                    Variables              12
Engine         V9                                      Indexes                 0
Created        Tuesday, February 07, 2012 09:01:02 AM  Observation Length    232
Last Modified  Tuesday, February 07, 2012 09:01:02 AM  Deleted Observations   0
Protection                                             Compressed             NO
Data Set Type                                          Sorted                 NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)

                             Variables in Creation Order

#    Variable                Type    Len    Format         Label

1    modul_unterschrift_id   Num       8    11.            modul_unterschrift_id
2    form_id                 Char     45    $45.           form_id
3    form                    Char    100    $100.          form
4    arzt                    Char      1    $1.            arzt
5    arzt_datum              Num       8    DATE9.         arzt_datum
6    monitor                 Char      1    $1.            monitor
7    monitor_visuell         Char      1    $1.            monitor_visuell
8    monitor_datum           Num       8    DATE9.         monitor_datum
```

Program Header

Every program must contain a header! It is perhaps the most useful piece of metadata a programmer can provide. Headers serve as descriptive information about the creation, purpose, and usage of the program. Although the style and format of a program header may vary across programs and companies, I believe there are six critical elements a header must contain: program name, author, date created, purpose, inputs, and output.

Practice or Standard? Standard. To be helpful internally to your organization and externally to a reviewer such as the FDA, the header must be implemented.

Here is a sample program header:

```
*************************************************************************;
** Protocol\ Project:                                                   **;
** Program Name:                                                        **;
** Purpose:                                                             **;
** Creation Date: DDMMMYYYY                                             **;
** Author:                                                              **;
** SAS Version:                                                         **;
** Input:                                                               **;
** Output:                                                              **;
*************************************************************************;
** Modification History                                                 **;
** Date      Initials Description                                       **;
** ------------------------------------------------------------------- **;
** DDMMMYYY                                                            **;
*************************************************************************;
** Program Notes                                                        **;
*************************************************************************;
```

Revision History and Archiving

Programs will inevitably be modified, either due to corrections or enhancements. There are technologies available that allow program modifications to be archived and tracked. Most of the solutions I have encountered operate independently from SAS® and are neither financially feasible nor technologically manageable for most programmers. Without systems to audit revisions and archive programs, the management of modifications becomes a manual process the programmer is responsible for. Although most manual processes are prone to human error and are

therefore not ideal, manually archiving and tracking modifications can be a practical solution. At Alexion we encourage all programmers to systematically back-up their programs before a revision is made. We use a 'revision history' section in our program header which includes the name of the modifier, the date, and a description of the modification.

Practice or Standard? Practice. Although this convention is realistic, the absence of technology to force the documentation of revisions makes this convention unmanageable and not auditable. I highly encourage all programmers to employ a trail of all modifications.

Comments

Most programmers will find well commented programs easy to understand and manage. The golden rule of programming is to write scripts that can be easily read and understood by anyone the first time they open your program.

Practice or Standard? Standard. Every program must contain comments even though the logistics of commenting may not be easily defined. It is difficult to regulate how and when a programmer comments their scripts but, it must be done somehow. All of the GPP contributors agree on the commenting convention however, some of my fellow contributors propose 'placing comments before all major data steps'. While I agree with them, none of them define the difference between 'major' and 'minor' data steps and procedures. I do not think there needs to be such a distinction. Commenting must be done in any way deemed realistic and manageable to provide a clear level of understanding.

Additionally, I think there should be a uniform style of commenting. My suggested commenting style is to use the forward slash and asterisk to begin and end each comment as seen below.

```
/* Comment 1 */
/* Comment 2 */
```

This style makes it easier to utilize programs and logs as metadata.

I also caution programmers to be extremely careful with the contents of their comments. Since we are uncertain of the path our programs will travel, we must anticipate them being read by the FDA. Before you write a comment, ask yourself "Is this something I want the FDA to read?" Also remember, that too much of anything isn't a good thing. Not only is less better but, it also has the potential of cluttering a program and leading to confusion.

Meaningful Naming Conventions

Another way to enhance the legibility of scripts is to describe data and variable attributes by using meaningful, non-technical conventions. Naming conventions which provide purpose increase comprehension. Use naming conventions which describe what the scripts are doing. Below is a very simple example. The script is deriving an age variable as the difference between a subject's date of first dose and their date of birth. In the first example, the derived variable 'x' is less descriptive as compared to the derived variable 'age' in the second example.

```
x = rfstdt - dob;

age = rfstdt - dob;
```

Practice or Standard? Standard. Choosing names which provide important information help readers understand their meaning and increase the readability of the program. The same philosophy should apply to dataset names, formats, informats, macro variables, arrays, and anywhere else it makes sense to apply within the program. Additionally, naming conventions should be used for program names, log files and list ('lst') files.

One SAS® Statement per line

A SAS® Statement is a line of code which ends with a semi-colon. There are rare exceptions to this such as SQL scripts in which a 'statement' is not defined by a semi-colon, but rather by a specific combination of code (e.g. 'select', 'create table', 'order by', etc.). A SAS® Statement can be as short as two characters and as long as your SAS® environment allows. To enhance the readability of your program, keep all scripts to a minimum of one line per statement.

Practice or Standard? Standard. This convention has been suggested by almost all GPP contributors as a convention to enhance readability. I agree! Although this might be considered a relatively difficult convention to enforce there is more than one reason to make this a standard. When submitting your programs to the FDA as part of an eCTD, '.SAS' is not an acceptable format. Programs need to be converted to ASCII files (aka: "txt' files). After experience with submitting programs to the FDA, I would also take this one step further and specifically suggest ending each statement with a 'hard-return'. This ensures a consistent flow of code in the program even after it is converted to an ASCII file.

The scripts below demonstrate proper and improper usage of this convention. In the first example, the dataset 'badcode' is written continuously. Meanwhile, dataset 'goodcode' is properly written in the second example which is much easier to read and understand.

```
data badcode; set names; do i = 1 to 4; newname = name; order = i;
output;end;run;
```

```
data goodcode;
     set names;
        do i = 1 to 4;
             newname = name;
                order = i;
                output;
        end;
run;
```

The only exception to this would be for long statements. The Steering Board suggests splitting long statements among as many lines necessary to make it readable. Other contributors suggested using an exact line size (e.g. 72) as a guide for splitting long statements.

Blank lines between DATA Step and PROC

Inserting a blank line in between each DATA Step and PROC has only two purposes – better flow and understanding.

Practice or Standard? Standard. This practice has been suggested by most GPP contributors including the Steering Board. Although it might not be problematic for relatively short programs, it will be for longer and more complex programs. These will be difficult to read and understand if blank lines between data steps and procedures are absent.

Left Justify all Statements and Indent

There is no other reason to left justify your statements other than to increase readability. Previously I mentioned the standard convention involving 'English' as the universal language. The English language reads left to right. As a result, most documents have been widely accepted as left justified.

Ten years ago when I opened up SAS® for the first time at the SAS® Learning Center in New York City, I was taught to 'sandwich' my scripts. This provided an image of how scripts should look when they are properly indented. The top piece of bread is the data step or procedure. The bottom piece of bread is the 'run' or 'quit' statement. The 'meat' of the sandwich is everything in between.

Most importantly, 'do loops'. 'logical', or 'conditional' coding need proper justification and indentation. The Steering Board and most GPP contributors unanimously agree the 'do loops' and 'logical' code should also be properly indented.

The first 'do-loop' below is right justified, not indented and difficult to read. Imagine an entire program written in code similar to the first 'do-loop'. It would be sloppy and difficult to read.

```
                  DO something;
                       DO more;
                IF X=1 THEN DO;
                    statements;
                            END;
      ELSE IF X NE 1 THEN DO;
                  statements;
                            END;
                            END;
                            END;
```

This second 'do-loop' is left justified, indented ('sandwiched') and much easier to read.

```
DO something;
     DO more;
          IF X=1 THEN DO;
               statements;
          END;
          ELSE IF X NE 1 THEN DO;
               statements;
          END;
     END;
```

```
END;
```

Practice or Standard? Standard. Left justified, indented programs are more widely accepted. Left justification might not be preferred by programmers whose native language is Arabic, Hebrew, Chinese or Japanese. They might prefer fully justified or right justified script, however, you are creating your programs with the possibility it will reviewed by the FDA.

Remove all unused scripts

It is acceptable to tests scripts during the development phase of programming. When we test scripts we usually 'comment out' scripts and run various combinations of scripts until we find the most suitable one. Before passing the script off to validation and running the program for production, all unused (aka 'dead') scripts must be removed. Not doing so makes the program unmanageable, confusing, and suggests the program might not be finalized.

Practice or Standard? Standard. Leaving dead scripts in a program is confusing, sloppy, and an indicator for careless programming.

Case Standards

Fonts in upper case ('up case') exhibit negative or angry tone. For example, 'DON'T DO THAT' is interpreted much differently than 'don't do that'. Many businesses are prohibiting the use of 'up case' fonts in communication. Uniformly using up case font deteriorates readability.

Practice or Standard? Standard. To increase readability, all programs must be written using fonts in lower case. In the two examples below, dataset 'one' is easier to read than dataset 'two'.

```
data one;
     set raw.rd_disp  (rename=(dthdt = dthdte notcpr = notcpr_edc));

        studyid  = protocol;
        siteno   = cats(sitemnemonic);
        patid    = cats(subjectnumberstr);
        patinit  = subjectinitials;
        visitnme = cats(visitmnemonic);
        dpcompny = input(dpcompny_c,8.);
        howcp    = howcp_c;
        hownotcp = hownotcp_c;
        notcpr   = notcpr_c;

run;


DATA TWO;
     SET RAW.RD_DISP  (RENAME=(DTHDT = DTHDTE NOTCPR = NOTCPR_EDC));

        STUDYID   = PROTOCOL;
        SITENO    = CATS(SITEMNEMONIC);
        PATID     = CATS(SUBJECTNUMBERSTR);
        PATINIT   = SUBJECTINITIALS;
        VISITNME  = CATS(VISITMNEMONIC);
        DPCOMPNY  = INPUT(DPCOMPNY_C,8.);
        HOWCP     = HOWCP_C;
        HOWNOTCP  = HOWNOTCP_C;
        NOTCPR    = NOTCPR_C;

RUN;
```

Use keywords instead of positions

Positional scripts pass values conditionally with reference to a place in the code. Keywords scripts use words to describe how or where the values are being utilized. Positional coding appears mostly within two areas of programming: Macros and SQL. In both cases, keywords enhance the readability and maintenance of the script.

When passing values into a macro, use keywords to identify the value instead of positions.

```
%frequency(demog, agecat, sex)

%frequency(data=demog, descrip=agecat, by=sex)
```

In the first 'macro call' there is no way of knowing which macro parameters the values are being passed into. Not only is this confusing for a reviewer, but it could also lead to error if we aren't sure what the correct order of the macro variables.

Likewise, positional SQL scripts are more difficult to read, understand, and update. In the first example below, a positional SQL script is used. The values from the 'select' statement are passed into the dataset 'radio' based on the position of the variables prescribed in the 'insert' statement. Not only does this make it difficult to read and understand, but challenging to update.

```
proc sql;
       insert into radio (STUDYID, SITENO, PATID, PATINIT, VISITNME, EDCSEQNO,
                          RDSEQNO, RDDD, RDMM, RDYY,)
       select PROTOCOL, SITEMNEMONIC, SUBJECTNUMBERSTR, SUBJECTINITIALS,
              VISITMNEMONIC, compress(put(RDSEQNO,8.)), SEQNO,., ., .,
       from radio_all a;
quit;
```

In the second example, keyword SQL scripts are used. You can clearly see what values are being passed into the variables.

```
proc sql;
     create table radio0 as
     select protocol as studyid,
            sitemnemoni as siteno,
            subjectnumberstr as patid;

quit;
```

Practice or Standard? Standard. It is more beneficial utilizing positional scripts. Not only does it enhance readability but maintainability.

Specify the name of the macro on the %MEND statement

Providing the name of the macro on the 'mend' statement enhances the readability of a program.

Practice or Standard? Standard. This is particularly important when nesting macros inside of each other. In the example below two macros are nested together.

Always use DATA= in Procedures

Many GPP contributors suggest this method, which I support. First and foremost, it ensures that the correct dataset is being used. Second, it allows for enhanced readability and increased understanding.

Practice or Standard? Standard. It provides better readability and traceability.

PORTABILITY

Setup Programs

Standard library references, formats, format catalogs, and options must be adopted by an organization to ensure consistency and must be initialized or 'set-up' through a separate program. Many GPP contributors agree with this convention, as it ensures correct data is being used by all programmers.

Practice or Standard? Standard. Although there are many ways to ensure consistency without using a separate initialization program, this method is one of the easiest and highly manageable conventions. At Alexion, we use a standard file structure and hierarchy. This allows us to use an initialization program to establish standard library references, format catalogs, macro libraries and options.

The following example is from the log of an Alexion program. The libnames and format catalogs are established automatically. Macros are stored in a separate directory and are automatically called. The primary advantage of this convention is the ease of change and consistency between all programs. You can clearly see how portable the programs are. Programmers do not have to specify any library references or format catalogs.

```
NOTE: Libref CRT was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\<drug>\<PROTOCOL>\Data\CRT

NOTE: Libref SDTM was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\<drug>\<PROTOCOL>\Data\SDTM

NOTE: Libref ADAM was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\<drug>\<PROTOCOL>\Data\ADaM

NOTE: Libref SOURCE was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\<drug>\<PROTOCOL>\Data\Source

NOTE: Libref GFORMATS was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\Formats

NOTE: Libref DFORMATS was successfully assigned as follows:
 Engine:        V9
Physical Name: D:\SAS\<drug>\Formats

NOTE: Libref PFORMATS was successfully assigned as follows:
Engine:        V9
Physical Name: D:\SAS\<drug>\<PROTOCOL>\Formats
```

DATA INTEGRITY

All programs must have a log

A SAS® log ('log') is a record of the execution of the program. It includes information about the execution of each statement in the program.

Practice or Standard? Standard. The log is the only proof a programmer can produce to show that the program executed successfully. I suggest that there be one log per Program (e.g. demo.sas, demo.log) and the log is saved automatically. At Alexion, the following macro is placed at the end of every program to ensure it is saved with each run of the program.

```
%macro savelog;
    dm log 'print file ="&_logpath\&pgmname..log" replace';
%mend savelog;
```

Show all executed scripts in the log

A log showing fully executed script benefits both the programmer and the reviewer. It provides full disclosure of what the program is reading, manipulating, and writing. Most importantly, the log indicates how successful the program is performing through messages regarding errors, warnings, and notes.

Practice or Standard? Practice. There is no requirement to show all executed scripts in the log. This convention is one I highly recommend since more information is beneficial. A log with a fully executed script provides valuable information for a programmer to defend themselves and their work in the event of an audit.

In the log message below demonstrates the use of the MPRINT options. By doing so, you can see the resolutions of macro variables in addition to the notes describing the data.

```
MPRINT(REPORTIT):  ;
MPRINT(REPORTIT):   proc sort data=final out = out.t14230380001;
MPRINT(REPORTIT):   by breakpg ord1 ord1 tmp;
MPRINT(REPORTIT):   run;

NOTE: There were 28 observations read from the data set WORK.FINAL.
NOTE: Duplicate BY variable(s) specified. Duplicates will be ignored.
NOTE: The data set OUT.T14230380001 has 28 observations and 8 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.09 seconds
```

Errors in Log

Errors are prohibited! There is no acceptable reason for errors when a program executes. Although I do encourage programmers to write script to generate internal edit or logic checks on the data, I caution them to use the word 'error' in the 'put statement' writing to the log.

Practice or Standard? Standard. There is no reason to allow an error in the log.

Avoidable warnings or notes in the log

Programmers eventually become lazy. We start to ignore warnings and notes and leave them if we think they are harmless. Many of the GPP contributors agree that avoidable warnings or notes in the log are not acceptable. These messages are confusing and usually lead to errors. If any message in the log is avoidable programmers should program around it.

Practice or Standard? Standard. I fervently stand behind my fellow GPP contributors however, this standard must be very specific. There are hundreds of different warnings and notes the SAS® System will generate. I believe standards should be adopted to prohibit avoidable warning and notes. At Alexion, the following notes are not permissible because they are easily avoidable and affect the integrity of the output:

Notes about uninitialized variables, formats or macro parameters. For example:

```
NOTE: Variable <variable name> is uninitialized.
```

Notes about variable conversions. For example:

```
NOTE: Character values have been converted to numeric values at the places
given by: (line):(column).

NOTE: Numeric values have been converted to character values at the places
given by: (line):(column).
```

Notes about invalid arguments to SAS® functions. (e.g., input, put, mdy, substr, etc.). For example:

```
NOTE: Invalid argument to function <function name> at line xxx column yyy.
```

Notes about mathematical operations not being performed or performing operations on missing data. For example:

```
NOTE: Mathematical operations could not be performed at the following places.

NOTE: Missing values were generated as a result of performing an operation on
missing values.
```

Notes about merge statement. For example:

```
NOTE: Merge statement has more than one data set with repeats of by values.
```

Notes about the SAS® system stopping notes. For example:

```
NOTE: the sas system stopped processing this step because of errors.
```

For example:

Automatically check your logs

The log is proof that your program executed successfully. The FDA does not have a specific requirement to submit your logs, however, everything is open to inspection during an audit. Personally, I believe it is a matter of time before the FDA and other regulatory agencies start requesting logs. The log is the most important documentation to describe how successful your program executed.

To ensure the log is free from error, avoidable warnings and notes, the log must be checked by the programmer. I spent the majority of my career working for CROs being audited and having to answer to my program and the program log. I have spent a good portion my pharmaceutical career auditing other companies and programmers. I have been surprised by the amount of manual checking that has occurred with logs.

Practice or Standard? Practice. Companies can check their logs manually however, they should do so automatically. This assures you and the FDA that the log is free from error. At Alexion, we have a standard macro called into all of our programs to check the log for all errors, warnings, and the notes marked as prohibited in our Programming SOP.

The script below is a snap shot of some code from Alexion's standard check log macro. This macro is placed in every programming folder. It reads in the logs and checks keeps lines from the log which contain keywords which violate our SOP.

```
filename logs "&_logpath\*.log";

data all;
     length filename $256 name $32;
     infile logs filename=filename eov=eov length=L;
     input grepline $varying256. l;

     if index(grepline,'ERROR')          or
        index(grepline,'WARNING')        or
        index(grepline,'uninitialized')  or
        index(grepline,'converted to')   or
        index(grepline,'to function')    or
        index(grepline,'not performed')  or
        index(grepline,'missing values') or
        index(grepline,'repeats')        or
        index(grepline,'stopped ');
run;
```

Hard-coding and Imputing

Probably the most taboo term in the Pharmaceutical Industry is 'hard-coding'. I believe it is misused and highly confusing. What is hard-coding? There are many definitions, not to mention many thoughts and opinions about what hard-coding is. In object oriented programming such as the 'C' languages (e.g. C+, C#, etc) hard-coding is defined as "an explicit rather than a symbolic name for something that is likely to change at a later time"[5]. This definition can be applied to programming. Hard-coding is when the script being placed into the program is specifically written in such a way that it cannot be logically modified. Programmers hard-code scripts all the time.

The concept is neither bad nor ideal. We want to write programs that are robust and transportable across multiple platforms. Specifically, in the clinical trial and pharmaceutical world, we want to create programs that are standard enough to be used across multiple protocols. Hard-coding reduces portability of our programs and makes them difficult to manage. The concept of hard-coding has evolved into a concept which can cause potentially adverse effects.

A few simple examples of hard-code are listed below.

```
options ls = 250 ps = 80 missing = 0;

libname libref = 'C:\users\tables';

if patid – '00X-00X' then gender = 'M';
```

Each of the three examples above is considered to be hard-coding because we are placing scripts into the program in such a way that it cannot be logically modified. In the first example we are hard-coding options into the program. In the second example, we are hard-coding the library reference. In the third example, we are hard-coding a patient's gender. These are three simple examples of hard-coding.

The third example is the most problematic one, rather, its explicitly changing the value of the source data. What if the subject's gender is marked on the CRF as female and the code is changing it to male? This is problematic to a reviewer because they will want to know why it was done.

The perception of hard-coding that the Pharmaceutical industry should be concerned with is when it explicitly changes the source value of a variable.

Imputing data is similar to hard-coding except the value being assigned is replacing null data. Imputation is a very common practice in statistical analysis and is widely accepted.

Practice or Standard? Practice. In the perfect world there should be no need to hard-code values or impute data. However, it happens and is sometimes necessary at various levels. We all wish we can collect perfect data and be able to effortlessly describe all data points. Nonetheless, we do not live in a perfect world and hard-coding and imputation may be required. The key is to fully disclose the reasons for changing the values or imputing the data and to put every effort forward to not do it.

EFFICIENCY

Use Dataset Name once in a program and only write to a permanent library when needed

Using the same name of a dataset over and over in a program adulterates traceability and makes it difficult to update and maintain. Additionally, you should only write a dataset to a permanent library when the dataset is final.

Practice or Standard? Standard. Using the same name only once not only increases traceability but facilitates maintenance and validation. Repeatedly writing a dataset out to a permanent library is dangerous, a waste of processing time, and unmanageable. It is dangerous because you increase the risk of overwriting a dataset. Writing a permanent dataset uses more processing time than writing a temporary dataset. It also deteriorates traceability because you are using the same name more than once.

Conditional Logic

When coding IF-THEN-ELSE statements use a final ELSE statement to code any observations that do not meet the logical conditions.

```
If  0 < age <= 18 then agecat = 1;
      Else if 18 < age <= 65 then agecat = 2;
      Else if age > 65 then agecat =3;
      Else agecat = 4 ;
```

Practice or Standard. Practice. Although I highly suggest this method as it is more robust than other non-traditional methods, it is not the only way to identify discrepancies.

'Other' as format keyword

When coding a user-defined FORMAT, include the keyword 'other' on the left side of the equals sign so that all possible values have an entry in the format.

A missing entry in a user-defined FORMAT can be difficult to detect. One way to identify this potential problem is to ensure that all values are assigned a format.

Practice or Standard? Practice. There are multiple ways to catch this discrepancy.

IF IT ISN'T DOCUMENTED, IT DIDN'T HAPPEN

What SOPs do you have? How did you follow your SOPs? Do you have proof that you followed your SOPs? These three questions will undoubtedly be presented to you by the FDA auditor at your company. Every company big or small has Standard Operating Procedures ("SOPs"). SOPs are characteristics, practices, steps, or procedures a company commits to follow to ensure levels of safety and effectiveness. GCP is quite clear on the responsibility of the sponsor. The sponsor is responsible for implementing and maintaining quality assurance and quality control systems with written SOPs to ensure that trials are conducted and data are generated, documented recorded, and reported in compliance with the protocol, GCP, and the applicable regulatory requirements. In other words, f*ollow a standard and provide you did!*

There is only one way to demonstrate compliance with a standard and that is through documentation. When I first started my career in the Pharmaceutical industry ten years ago a good friend of mine told me "if it isn't documented, it never happened". I learned the understanding of this during an audit I performed. I requested the logs from the final run of safety and efficacy tables from a vendor. They couldn't provide the logs, nor could they provide any hard-copy evidence assuring me they had checked their logs. They told me they did it but couldn't prove it. My audit report simply stated 'the company did not check their log". It wasn't documented, so it never happened.

How can we document Programs?

The preceding section on readability discussed ways to document your program to make it legible, comprehensible and maintainable. There is another level of documentation each program must have to prove its compliance with SOPs about programming.

If GPS is realistic and manageable, then it should be easily documented and easily auditable. There must be a document for every program minimally describing the following:

1. Project
2. Development program and output location
3. Validation program and output location
4. Documentation location
5. Programmer name

6. Validator name
7. Validation date
8. Result

Additionally, GPS documentation should explicitly identify each convention being examined. The example in Figure 1 is a snap shot of programming documentation used by Alexion. During the development phase of a program, each program goes through 'cycles' until it is validated completely. The form contains sections for header information, a cycle summary, test descriptions, results, and signatures. Attached to each form should be substantiating documents such as COMPARE procedure output or results of programmatic validation.

Every company must adopt their own documentation procedure and document every program. Some companies implement this level of documentation on a random selection of a percentage of programs. For example, 5 out of 100 programs (5%) might be selected for this level of documentation. In my experience, this increases the potential for non-compliance with GSP. Remember, it is our responsibility to implement and maintain quality assurance and quality control systems for all programs.

Risks of Poor Documentation

The primary risk of poor documentation is the inability or failure of a company to ensure compliance with GPS. If it isn't documented it didn't happen. Some GPS can be assured through output and logs, however, documentation assures that each GPS was checked and complied with. It neither guarantees nor does it mean compliance. It simply means compliance was evaluated.

Protocol/ Project:	
Development Program:	
Development Program Output:	
Validation Program:	
Validation Output:	
Validation Form:	
Validation Level: *(select one)*	

Cycle Summary

Cycle #	Programmer	Validator	Validation Date	Result	Comments
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Test #	Check Item	Check Description	Cycle # (P = Pass, F=Fail, N=Not Applicable) 1	2	3	4	5	6	7	8	9	10
Output Review												
1.01	Output	Output has been reviewed.										
Log Review												
2.01	Logs	A log using the same naming convention as program.										
2.02	Logs	Errors were not found.										
2.03	Logs	Avoidable warnings were not found.										
2.04	Logs	Only permissible notes were found.										
Code Review												
3.01	Program Header	Program header is being used.										
3.06	Program Comments	Comments are used to explain logic flow.										
3.07	Naming Conventions	Explicit and meaningful names for variables and datasets.										
3.08	Naming Conventions	Same name for a dataset is not used more than once.										
3.09	Naming Conventions	SAS Version 5 Compatibility Requirements met.										
3.10	Program Design	Program meets regulatory submission criteria.										
3.12	Program Design	Program code is left justified.										
3.13	Program Design	Proper indentation is used.										
3.14	Program Design	All 'DATA' and 'PROC' steps are closed.										
3.15	Program Design	"DATA=" is used appropriatedly.										
3.16	Program Design	All SORT Procedures use 'OUT='.										
3.17	Program Design	RETAIN statement is not used to control variable order.										
3.20	SQL	Positional 'SELECT' statements are not used.										
3.21	SQL	"FEEDBACK" option used when creating a dataset.										
3.24	SQL	Every SQL statement ends with a QUIT;										
3.26	Macros	Macros are written and invoked using keyword parameters.										
3.28	Macros	No macros are created using "SYS" prefix.										
3.29	Macros	All mend statements include the name of the macro.										
Independent Programming												
4.01	Output	Match on all dataset and variable values and attributes.										

Results:

Problem Logged:			Problem Resolved:
Cycle #	Test #	Validator Comments	Programmer Response

Validator	Signature	Date

Figure 1. Sample Program Documentation

PREPARING FOR THE FDA AUDIT

When the FDA first arrives at your building they will ask for several key documents, most importantly your SOPs. After meeting with several executives and senior management they will delve into the SOPs and start to explore processes. They will probably meet with department leaders, ask questions, and look for answers. Once an answer is provided they will ask for documentation to support your answer.

There are four key steps to take to prepare for an FDA audit: 1.) review your SOPs, 2.) audit your documentation, 3.) respond to any gaps, and 4.) organize yourself.

Review your SOPs

SOPs are rules we have defined and agreed to follow. GCP requires us to create standards and comply with them. Often, there is so much detail embedded in the SOPs we can't recall all of them. The FDA will critically review each line of your SOPs, ask questions, and request documentation. Reviewing your SOPs beforehand will refresh the details you may have forgotten and help clarify any gaps you might need to address.

Audit Yourself

Be prepared! After you have reviewed your SOPs go to your documentation and make sure nothing is missing and there are no errors. Create a checklist to audit yourself against and use it routinely before the FDA arrives.

Respond to any gaps

Documentation is the single most important key when dealing with an FDA auditor. They understand the world isn't perfect and neither are clinical trials. Documentation is synonymous with awareness. It's acceptable for something to deviate from the planned protocol as long as you were aware of it, addressed it, and documented it.

Organize your documentation

When the FDA is on site, everyone will be excitable and full of energy. The FDA will need punctual answers and organization will expedite responses.

CONCLUSION

The words 'programming', 'scripts', and 'code' are scarce in most of the guidance about GCP. There is minimal guidance on how to write programs, what the programs should do, and what they should look like. The requirements that exist suggest ways to present and transmit the data, but fail to standardize the organization of the programming conventions. GCP specifically suggests that quality control should be applied to each stage of data handling to ensure that the data is reliable and has been processed correctly. This must apply to the programs that are written to produce the data that the FDA reviews.

We can write scripts in a variety of ways and have the ability to manipulate data intentionally and unintentionally. Without regulation, this has the potential to be dangerous. I believe the majority of concepts in this paper are realistic and manageable. They are key programming principles which can and should be adopted by all programmers to ensure reliable results.

After twenty years of GxP in the Pharmaceutical Industry, its evolution has reached a lull. The FDA is aware of inefficiencies and plans to hold a two-day public hearing in April 2012. While the purpose of this meeting is to discuss improving practices on the conduct of clinical trials, programming is not on the agenda. There is an identifiable lack of regulation for the scripts used to create, analyze, and submit data. The consistencies among all of the GPP contributors suggest the need for and call for regulated and authoritative guidance from the FDA. Until that time, I strongly recommend companies in the Pharmaceutical industry to adopt standard programming procedures that are realistic, manageable, and auditable.

REFERENCES

1. "Vision." *: ICH.org*. Penceo. Web. 31 Mar. 2012.
2. "FDA Standards for Electronic Submissions." *SAS.com*. SAS.com. Web.
3. Chen, William, Coughlin, Margaret. 2008. "An Approach to CDISC SDTM Implementation for Clinical Trials Data". NESUG.
4. "Good Programming Practices For Clinical Trials." SAScommunity.org/wiki. 13 June 2010. Web. 31 Mar. 2012.
5. "Hard-code." *What Is ? Definition from WhatIs.com*. Web. 31 Mar. 2012.
6. "FDA Plans Two-day Hearing on Improving GCP." *Thompson.com*. 8 Mar. 2012. Web. 31 Mar. 2012.

7. Levin, Lois. 2006. "SAS® Programming Guidelines". SUGI 31. Paper 123-31.
8. Winn, Thomas. 2004. "Guidelines for Coding SAS® Programs". SUGI 29. Paper 258-29.
9. Ford, Jason. 2009. "Good Programming Practices in SAS®". NESUG 2009.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Vincent J. Amoruccio, M.A.
Associate Director, Clinical and Statistical Programming
Alexion Pharmaceutical, Inc.
352 Knotter Drive, Cheshire, CT 06410
(203) 439-9623
AmoruccioV@alxn.com