PhUSE 2015

Paper IS02

Introduction to GPP with Hints and Tips to make them Second Nature

Alexandra Marquart, Boehringer Ingelheim Pharma GmbH & Co. KG, Biberach, Germany
Shafi Chowdhury, Shafi Consultancy Limited, London, United Kingdom
Beate Hientzsch, Accovion GmbH, Eschborn, Germany

ABSTRACT

Within the Pharmaceutical Industry, we are always fighting with dinosaurs who refuse to follow good programming practices. "It is the way I have always programmed" we hear, however, following Good Programming Practice (GPP) can save companies a lot of time and money as we spend most of our time maintaining old programs and copying programs from one study to another.

Industry starters is the perfect crowd to learn about GPP, what is GPP and what they should do and avoid. If they are armed with hints and tips like how to use abbreviations to make following GPP much easier, they are more likely to both adopt these practices and to continue following them. This knowledge will set them up to follow the best practices in their long career as a programmer, ensuring their programs are easy to read and maintain over time and allow them to focus on the interesting and complex tasks.

INTRODUCTION

The need to follow Good Programming Practice (GPP) is more important now than ever before. When dealing with reduced timelines, limited resources, increase in remote teams, the practice of re-using and sharing code is more prevalent and necessary than ever. Following GPP right from the beginning will lead to high quality programs that are structured in a predefined, standardized way which increases the readability and reusability of the code. Program code can be maintained easier and shared in an efficient way with minimal resources.

The PhUSE GPP steering board has developed a GPP guideline that highlights main standards that should be followed and what should be avoided. These easy to follow standards offer the industry a simple way to ensure we continue to develop code that can be easily maintained and shared over time, saving time and cost. This paper gives an introduction to GPP providing tips that will enable industry starters to follow the GPP guidance and make them second nature.

WHAT IS GPP?

Good programming practice is a set of rules developed over time that have shown to produce robust programs that not only produce correct results, but are easy to read, understand and maintain over time. The guidance is primarily aimed at SAS programmers, however the principles of GPP also apply to other languages such as R and Stata.
The guidance aims to show how to produce well-structured and well-documented programs so that they are easy to read and maintain over time. It is designed to be applicable to all programs, and hence all programmers regardless of experience. Specific rules may be of more use to novice programmers, but applying the principles should be in mind for experienced programmers and mentors.

The basic principles behind GPP Guidelines are:

- The rules are easy to follow
- Improve clarity and readability of programs
- Improve quality and reliability of programs
- Make programs easy to update

Most pharmaceutical organizations have their own GPP guideline; these aims are the cornerstone behind them all and the PhUSE GPP is intended to define common principles.

GETTING STARTED

Before actually starting to program it is important to familiarize with the system you are working on as well as with the study and work package. Review all relevant documentation, including at a minimum the following:

- Clinical Study Protocol
- Case Report Form / annotated Case Report Form (CRF)
- Statistical Analysis Plan
- Analysis Dataset Specification
- Table, listing and figure shells

The protocol helps to understand the outline of the study, how many subjects or patients will be enrolled, entered and treated. Which phases are scheduled, i.e. screening, washouts, run-in and treatment phases, how many treatments are planned. The statistical sections give an overview on the primary and secondary endpoints. More detailed information can be found in the statistical analysis plan as well as in the tables, listings and figure shells, to see what data is reported and how.

The annotated CRF serves as base to understand where the data comes from, how it is collected and stored including dataset and variable names. Based on this information an Analysis Dataset Specification describes which derived datasets should be created and their content, including a detailed definition of endpoints. All derivations are recommended to be done in analysis datasets rather than in output programs. All this documentation helps to understand the data flow from raw data to tabulated datasets and analysis datasets to outputs.

Another important part of collecting all relevant information is to check whether there are already study or project specific programming and data standards available in combination with company and industry standards (like CDISC - Clinical Data Interchange Standards Consortium - which is a data standard with massive impact on data structure requirements and thus programming). Other similar studies or projects may already cover much of the programming and SAS code requirements, and if they have been written with GPP in mind then reusing them should save time and cost.

PROGRAM STRUCTURE

After this first phase of introduction to the new study/project the program development phase can start. The language used in programming code and within headers and comments should be English.

Programs should follow a clear structure. Start with a header, afterwards define the environment e.g. libnames, inclusion of macros etc., followed by steps to read in external data, do the processing, and then produce any outputs or permanent analysis datasets. Keep in mind that derivations should only be done once in one place, preferably in analysis dataset programs and avoid data driven programming.

As a basic rule the program code should be kept as simple as possible. Adding comments during programming, describing programming steps combined with easy to read code enables future reuse with a minimum effort, especially when somebody else needs to adapt or change the code.

PROGRAM HEADER

A standard header should be used for every program. The purpose of the header is to identify the program and to provide documentation including revision history. It provides the necessary information to a code reviewer to identify and understand the program and its development life cycle. Standardizing the header allows the information contained in the header to be leveraged programmatically for things such as auditing, project documentation, macro and dataset use tracking, consistency checking, and revision history reporting. The elements included in a header will vary from organization to organization.

The following should be included in all program headers:

- Identification of the project of which the program is a part
- Program name
- Author identification which should be human readable and unique
- Short description of program purpose

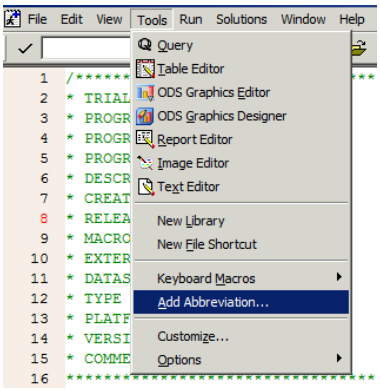
- List of external programs or macros used in the program
- Date program was first put into production, was finalized, or first passed validation
    - This date will be chosen based on the operational procedures used within the company /organization creating the program. The date should indicate the first date when the program was released for final use.
- Revision history
    - The revision history section is critical to document the revisions made to the program once it is put into production. It should include the author of the change, date of release of the change and a short description of the change. It may also include an identifier that can be used as reference in the code.

It is highly recommended but not required to also add the following in a standard program header:

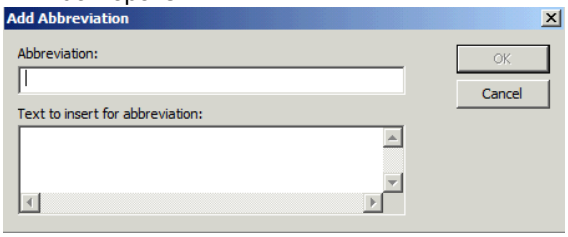
- The date on which program development started
- All outputs generated by the program
- External files used such as datasets or databases that are used as data inputs to the program or macros used
- Platform and operating system which the program was developed to run in
- Software/programming language and version which the program was programmed in

An easy way to only create the header once instead of typing the header each time when creating a new program is to use **SAS abbreviation** functionality. Create your own abbreviation including all points listed above by:

- Creating a header in the Enhanced Editor window; copy the code; place the cursor in the Enhanced Editor window
- Select *"Tools"* → *"Add Abbreviation"* in the menu bar

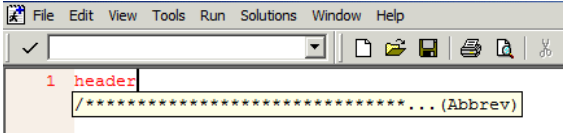


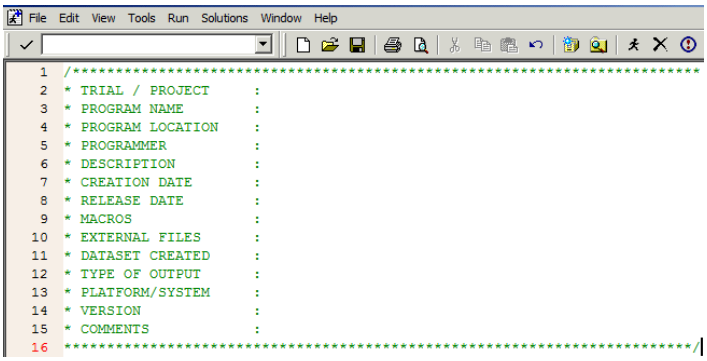
- A window opens:



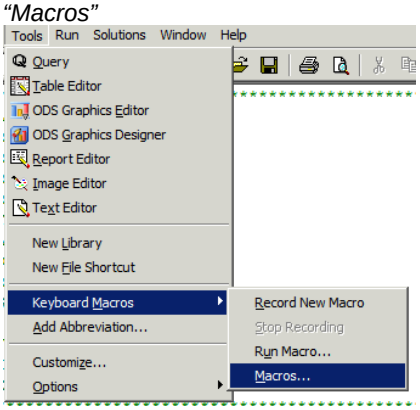
Insert the copied header code into *"Text to insert for abbreviation"*. Specify an easy to remember abbreviation name e.g. *"header".*

- If you want to use your abbreviation, place the cursor into the Enhanced Editor and type the name assigned. A small pop-up will appear showing the first bit of your header code. By using "enter" or "tab key SAS will insert the code and remove the abbreviation's name.





- If you want to manage (change or delete) your abbreviations select *"Tools"* → *"KeyboardMacros"* → *"Macros"*



Abbreviations in general can be used for pieces of code that are often reused.

COMMENTS

Comments are important to help anyone reviewing, modifying or using a program to be able to quickly understand the code. All major data or proc steps should be commented, especially data specific and complex code. Ideally comments should be comprehensive, and should describe the rationale and not simply the action. For example, instead of simply typing "Access demography data", describe which data elements you are accessing and why they are needed, for example, "Bringing in DM to get gender and age and subset to include only the intent to treat population". Comments can also include links to external documentation (requirement specifications, design documents). The programs can also be split up into sections by creating a different type of comments, e.g. many rows with asterisks which could also be easily implemented by using abbreviations (see above). This helps to structure the program and make it easier for others to see an overview of the program. Please keep in mind to comment your steps right during programming, do not plan to add them afterwards.

Long comments inside a DATA step or a procedure should be avoided. Having the comments before the program code being described starts will make the program code easy to read and follow.

HOUSEKEEPING

At the end of the program code (starting with a header, followed by steps to read in external data, processing data, and outputting data) the program should clean up after itself. The SAS environment should be set into its original state, for example delete temporary datasets and reset options you used during programming.

PROGRAM CODING

Programs can be written in a variety of ways and styles to achieve the same objective. Good Programming Practice does not intent to force all programmers to use the same style. Rather, the goals of GPP are to increase program readability, decrease the time for program maintenance and increase the sharing, re-usability of code by establishing coding standards.

NAMING CONVENTIONS

All organizations should have standard naming conventions. Program naming conventions should make it possible to identify groups of related programs such as analysis dataset programs vs. output programs. Within output programs it should be possible to identify related groups as well, for example adverse events tables. Dataset and variable names should describe their content as best as possible. Temporary datasets and variables should be named consistently in a way that makes their purpose and role in the program clear. Where possible, organizations should use industry level standards, such as Clinical Data Interchange Standards Consortium (CDISC) standards for permanent datasets and variables. This aids sharing of programs across projects and facilitates the development of standard code. Special characters should be avoided in variable, dataset and output file names.

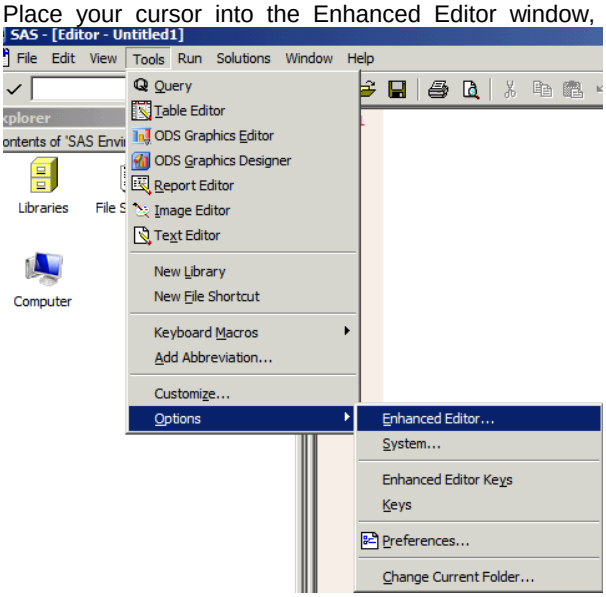
CODING CONVENTIONS

SAS code that follows standard conventions is much easier to follow, read, modify, maintain and correct. These conventions are divided into those which should be considered as required and those which are merely recommendations to be followed as applicable.

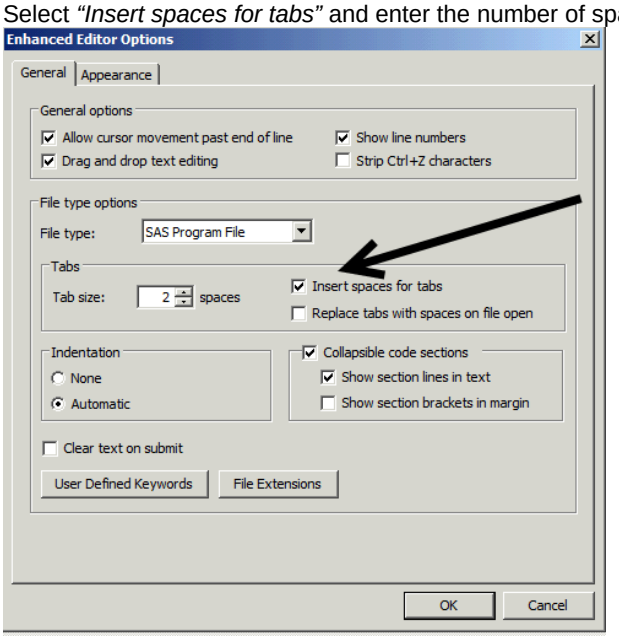
Required conventions are:

- Do not overwrite existing datasets or files
- Use meaningful dataset names
- Use `KEEP/DROP` statements to limit the number of variables and increase the processing speed
- Each organization may have its own standards for using upper/lower case within programming code but use of all uppercase should be avoided. Be consistent in using upper and lower cases throughout your code
- Use `"data=dataset"` option in procedure statements so that the dataset being used is explicitly stated to ensure that the statement will work if it is moved to another location
- Separate data steps and procedures with at least one blank line
- Write one statement per line
- Use consistent indentations, blank rows and comments to arrange the program in a clear structure
- Do not use tabs for indentation because they will be displayed differently depending on the platform and text editor being used, use blanks instead:

SAS editor options can be modified in a way that tabs are directly replaced by blanks. Place your cursor into the Enhanced Editor window, select *"Tools"* → *"Options"* → *"Enhanced Editor"*



Select *"Insert spaces for tabs"* and enter the number of spaces that should be entered instead into *"Tab size"*



- Left justify global statements and data procedure statements and their corresponding `RUN` and `QUIT` statement
- Use `RUN;` or `QUIT;` at the end of each data/proc step
- For do loops place the `END` statement in the same positions as the `DO` statement, so that they can be easily matched
- Insert parentheses in meaningful places in order to clarify the sequence in which mathematical or logical operations are performed
- When converting character variables to numeric or vice versa, use the `PUT` and `INPUT` functions to explicitly convert the variable to ensure that it is done in the way intended

Recommended conventions are:

- Use logical groupings to separate code into blocks
- Double space between sections
- Group similar statements together
- Define new variables with the `ATTRIB` statement in order to ensure that the variable properties such as length, format, and label are correct instead of allowing them to be implicitly determined by the circumstances in which they are initialized in the code
- Use defensive coding
  Defensive coding is an approach to programming intended to anticipate future changes of the data that might influence the coding algorithms. Ideally programs should be written in such a way that they will continue to work correctly in case of new unexpected data values which did not exist at the time the code was developed. (`IF … THEN … ELSE` logic or formats).
- Read and write only necessary observations and variables
- Do not re-read/import data more than once if possible
- SAS keywords should not be used for dataset and variable names
- Hardcoding should be avoided whenever possible
  Hardcoding is the modification of the value of an item of source data within the program code. Changes to source data should be done in data entry or capture systems. Permanent hardcoding to fix incorrect data values in a final database is strongly discouraged, but if it is unavoidable then it must be approved following a standard process and clearly documented.

PROGRAM PROCESSING

As part of development and validation practices, it is often mandated that the log file generated is checked to ensure that the program has executed in the correct intention. This log check should be done right after submitting the program code.

Many companies may have their own automatic log file checking utilities to aid this, and there are many examples of such tools in widely available papers.

"ERROR" and "WARNING" in logs should normally be avoided. There are sometime exceptions to this, such as warnings that are output from statistical models that do not have enough data. Ordinarily, any warnings that are deemed acceptable are to be documented.

There are some specific "NOTE"s / "INFO"s that can indicate a problem in the code. Check your log file carefully for:

- repeats
- more than one
- uninitialized
- referenced
- overwritten
- format was too small
- truncated
- invalid

The use of user-generated errors and warnings labeled "NOTE:", "WARNING:" or "ERROR:" can be useful at times, but these may make it difficult to find genuine problems when searching the log. In these cases, it is suggested to split the text in the program so it does not appear as a whole word within the program, but it does in the LOG if it is sent there. For example: "ERR" "OR: There should not be any duplicates. "

CONCLUSION

Start programming with the end in mind. Do not just simply write program code, use Good Programming Practice guidance right from the beginning. It may appear to be more cumbersome at the beginning to follow standard programming rules, but repeated use will help to adopt them quite easily and make them second nature. It does not take much effort to create abbreviations once and reuse them when required but they need to be created that one time.

Source code review together with an experienced programmer at the beginning can also be a good tool to help implementing programming practices for industry starters. It is also recommended to monitor GPP compliance from time to time –we all do have our bad habits.

REFERENCES

- PhUSE Good Programming Practice Guideline: http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice_Guidance

ACKNOWLEDGMENTS

We would like to thank the PhUSE GPP Steering Board for review and comments.

RECOMMENDED READING

- PhUSE Good Programming Practice Guideline: http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice_Guidance

CONTACT INFORMATION

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

Author Name	Alexandra Marquart
Company	Boehringer Ingelheim Pharma GmbH & Co. KG
Address	Birkendorfer Str. 65
City / Postcode	88400 Biberach an Riss
Work Phone:	+49 (7351) 54-94003
Fax:	+49 (7351) 83-94003
Email:	Alexandra.marquart@boehringer-ingelheim.com
Web:	www.boehringer-ingelheim.de