

# User friendly management of continuously improving standard macro systems

Katja Glaß, Bayer Schering Pharma, Berlin, Germany

## ABSTRACT

The environment today is affected by continuous improvements and high standardization. With those preconditions it becomes important to be flexible to use different standards at different times in different environments in a user friendly way. The purpose of this paper is to present the possibilities on how standard macros can be managed and how easy different macro systems and versions can be initialized in any environment with a flexible initialization macro.

## INTRODUCTION

Standards are around everywhere. But often they are hard to find. Especially within a company it is important, that all employees have the knowledge about and access to those standards. As there are various kinds of standards, this paper concentrates on standards which are implemented as SAS® macros.

## STARTING WITH STANDARD MACROS

### COLLECTION AND DEVELOPMENT

Typically the users start to write macros. At the beginning they are study specific, but then they become more and more flexible as the user reuses their macro for other studies as well. In the next step those macros are shared with other users. The pool of user macros is typically very large and valuable. It does make sense to collect those small macros, validate them and provide them as standard macros. Then all employees can benefit from the supporting tools and this pool can be seen as knowledge base. Example macros could be for date variable combinations, different calculations, value replacements, repeating checks and many more.

The second idea of standard macros is, that larger macros are developed containing complex structures and performing major tasks like creating a special demography table regardless of treatment quantity or additional stratification variables. Another example could be a complex macro to define treatment emergent events considering various rules or a standard macro to create box plots for laboratory values regardless to the scale of those. These complex macro systems are available to support time consuming tasks, which are more or less homogeneous and can easily be supported by complex macros.

### GROUPING AND DOCUMENTATION

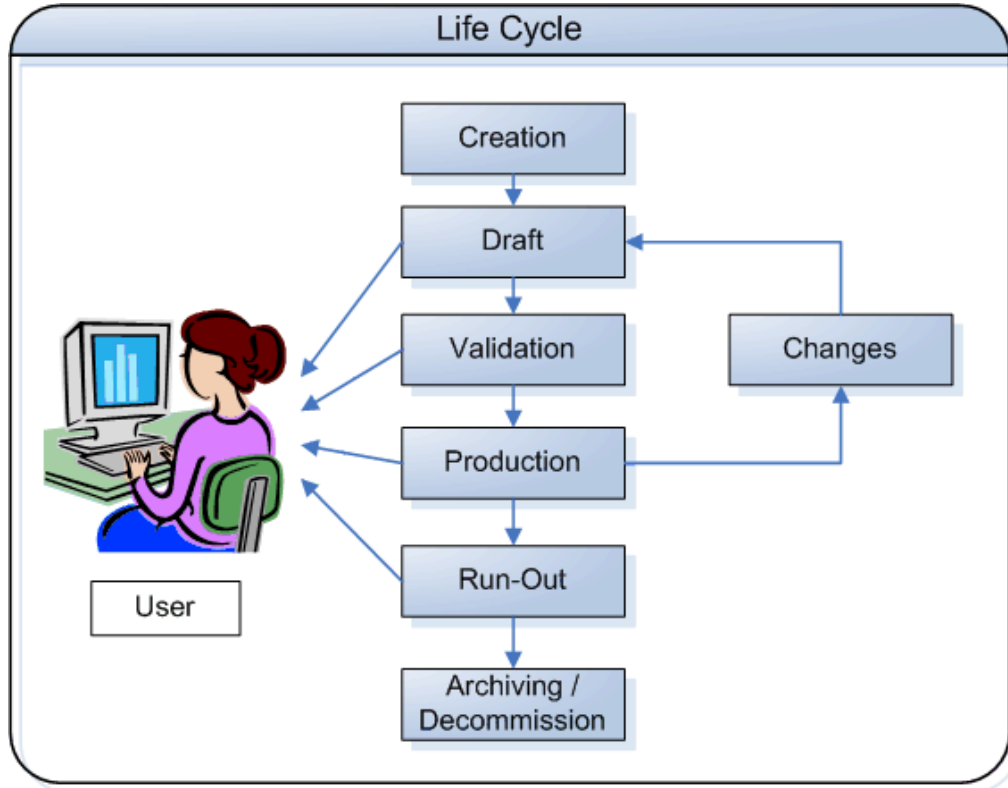
Typically the number of standard macros collected that way is in the beginning very low, but after a while it could grow dramatically. It makes sense to group the standard macros, so that it is easier for the user to look and find for solutions for specific problems. Grouping could be content like "Standard Table Package", "Standard Check Package" or "Support Box", but they can also be by function or site like "Pharmacometric Box", "GermanSite Package".

Each package needs to have some kind of documentation. For small support macros it could be enough, when the macro name is speaking and the details are described in the macro header. More complex macros needs a much better documentation in the form of a manual. Additional examples are essential to give a fast overview on how to use the macro, but also the details of calculation strategies needs to be documented to be tracable.

**MANAGING STANDARDS**

**LIFE CYCLE**

When the standards has been collected, developed and grouped, the next important question is regarding the life cycle management. Typically standards are not fixed, at least over decades. A standard macro to create a computed date variable according to a data standard should probably change within the future. Even when it does not seem, that changes should be performed, there will always be situations where this is required.



**Figure 1: Life Cycle overview of Standard Macro Systems**

The life of a standard starts with the creation. At the beginning a draft is created, for example a draft standard macro creating a standard demography table. As the development is typically done according to an immediate purpose, the draft may already been used by a user. Of course in this development phase, the user has to be aware that the macro could contain errors or malfunctions. For this the study specific validation is required to prove that for the used case, the macro works correctly. In the next step, the standard macros has to be validated. The validation should be very detailed and should also include a user validation to prove that the user requirements are fulfilled. After validation, the standard can be seen as final and can be released to production. Now all users can benefit from the standard without additional high validation effort.

To have a final standard does often not mean the end of implementations. Typically there is always room for enhancements. For example when the number of decimals of the demography table should be flexible, as some very special studies needs to use the standard macro slightly different. Also different data could lead to an update, e.g. it should probably be supported when having a single quote as minute unit within a label, which had not been working before. It is very likely, that the validation has not checked all cases which could appear. For this, more or less regular updates are required. After the update a draft could be provided, which again can be used with restrictions by users and after validation, everyone can benefit.

Some when comes the time, where special standard macros should not be used any more. Typically there is a run out phase, where for example the expiring standard is still used for ongoing studies to avoid the effort. For example for the nearly final study the old standard demography table should be used, whereas the new study should already use the new standard. This time should be used to stay efficient in the process, while new standards are upcoming.

It all ends with decommission. One major aspect is that studies has to be traceable and for old studies it is required to keep the original programs and macros (including standard ones) in case of upcoming questions regarding results. So the standard macros needs to be available in at least one way and best would be if they are still able to run in the current environment. For this decommissioning is typically not really easy and it needs to be investigated,

## PhUSE 2009

whether decommissioning is possible at all or whether some kind of archiving can be done. Probably those standards macros are just kept in its original status of the “run-out” phase for several decades.

With respect to decommissioning or long-term-archiving, an other aspect is the user aspect. When users are used to special standard macros, there needs to be reasons for them to switch as well. Standard macros should be replaced due to improvements in those areas, so that at the end the user profits. This has to be clear to the users, as they probably would use expired standards otherwise.

### VERSION CONTROL

As there will probably be different draft and final standard macros, a version control needs to be considered. Apart from different version for development, there should be visible versions which are to be used by end-users. Versions can be classified by different attributes. Versions could be identified by date or identifiers like letters or numbers. The most famous classification is by version number. Whereas the date classifies a version very detailed (that date the file(s) has been released), the number does not contain additional information. For users it is easier to talk about version numbers instead of version dates, especially as these can be remembered more easily.

Sometimes it makes no sense to create a new version whenever the standard macro is updated, e.g. a bug has been fixed. Nevertheless it is of importance to have an indicator, whether a standard macro has been updated or not. In this case a sub number could be used. To keep the version numbers as low growing as possible for the end users, the suggestion is to use increasing sub numbers whenever bugs has been removed or functionality has been enhanced that way, that the standard is backward compatible. In all other cases the main version number should be updated.

Also when not validated draft standard programs and macros should be provided, an additional indicator should be used to make aware users of the not validated status. This could be done by adding a “draft” mark next to the version number. Maybe a third version number level would be useful as well, when different versions of a draft macro should be available before the final standard macro is released to identify the currently used version.

### ACCESSING STANDARDS

After having the standards collection and decided on a version control, the storage location and access way needs to be defined. In case of having standard macros, there are two principles in general: provide them within a catalog or as SAS file.

### STORAGE FORMAT

The major advantage of a SAS file is, that transparency is provided. For small and easy macros even documentation and examples could be stored within the header of the file. When the users are interested in details which may not be described enough, e.g. which options are used for a proc summary, this is clearly visible. Those files can also be used as knowledge base, so when users needs to create similar processes, the file can be analyzed and the source can be used educational. The risk of providing SAS files is, that users which should not learn from the knowledge, e.g. external support personnel, also have the possibilities to do so. An other risk is, that the standard macro could be copied, modify and then the modified not validated version is used.

When using a catalog, then the source is invisible – remember using the NOPRINT NOMLOGIC etc. option or the SECURE option in SAS 9.2 [1] – and for this no changes and no review of the source is possible. This is to avoid modifications and knowledge transfer.

A suggestion would be to use the SAS file storage in-house and provide a compiled macro catalog when the standard macros should be used by a CRO who only needs to use those.

Typically the most beneficial aspect of an implemented standard macro is the specification and the interface. The implementation which is often seen as the major part, is in most cases just a minor. The specification as well as the interface on how to use the standard macro can not (and should not) be protected. Typically the documentation and validation costs the highest efforts. For this it could be seen as not too critical when providing standard macros as SAS program files to CROs as well.

# PhUSE 2009

## STORAGE LOCATION

The storage location should be accessible by all users and via the SAS® environment used to avoid the necessity of a copy. Typically there would be a standards area, which is write protected. The structure behind the storage location could be quite different, but at least it should follow always the same rules, so the user and an automated process could investigate where to find the standard macros. Often it makes sense to store additional files next to the macros, e.g. manuals and example programs including datasets and outputs.

For the transparency to the users it should be clear, where the standard macros or programs should be stored and should be used from. An easy solution would be to use just one folder containing all necessary files. But in that case there would be no overview and versions could not be managed, so it still does make sense to group the macros via purpose and version.

As mentioned, versioning is a very important process. For this the version needs to be considered for the folder structure as well. Development versions which are not validated should probably be stored at a different place, to point out that this is a not validated macro. Then there are still main and subversions of macros to be stored. As subversions could be released quite often and those are backward compatible, it does make sense to store only the latest subversion for user access. With this, the location is more stable and the user does not need to update the location used frequently. Old main version which are not backward compatible needs to be kept due to the requirement to reproduce old evaluations and to be able to provide a shift time where both version could be used.

## EXAMPLE STORAGE LOCATION

The suggestion is to have a standards area where the macros are stored. They should be grouped and the storage should be by version. All standards should be stored in the same structure for the usability. The following graphic displays one example how it could be done:

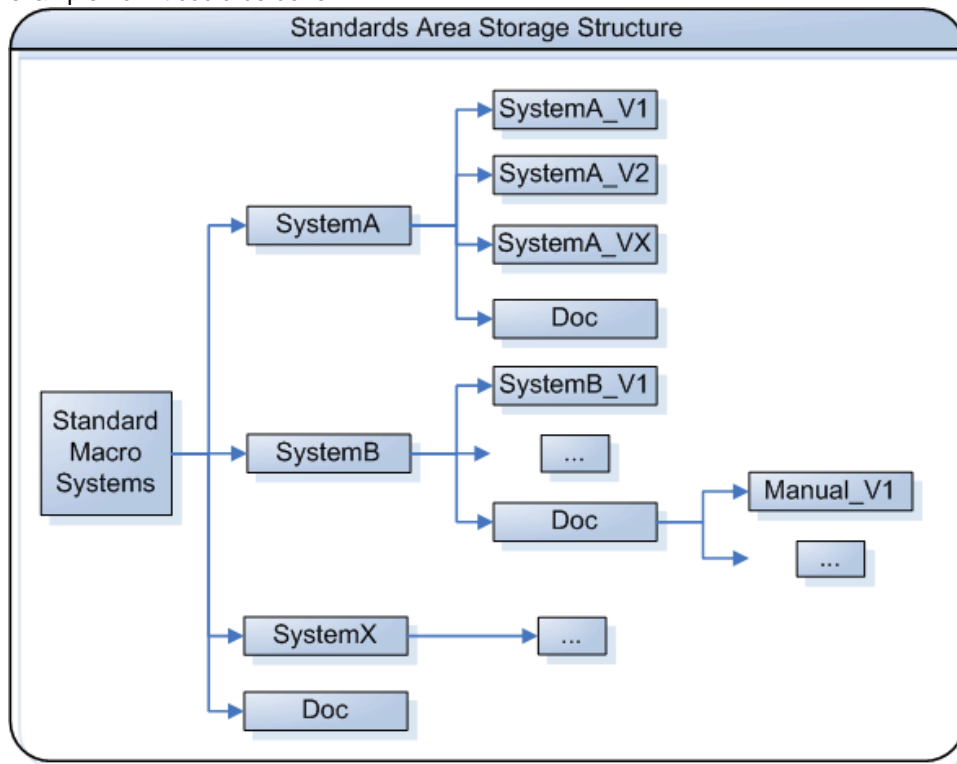


Figure 2: Example storage structure

This example has several advantages. There is a starting folder and all available systems can shortly be overseen by name and additional short system information can be found in the documentation “doc” folder. It happens quite often, that there are more and more macro systems being developed. Even though the topics should be structured it could still appear that due to different specializations the number of systems can become quite large. For this an overview documentation of the systems should be available to provide an overview for the end users.

Below each system, the system name is repeated with a connected main version number. There the macros which belong to the system are stored. No further subversions are available, as these should always be backward compatible and accessible via the main version number. Of course the unique subversions have to be stored somewhere in the development or validation area, so it will still be traceable when an aspect had not been backward

## PhUSE 2009

compatible and a question was raised according differences. Of course the system provider should make sure the backward compatibility is given any time, but in very rare cases side effects can not be excluded. When the system only contains easy to understand macros, a documentation within the header could be enough. More complex documentation should be stored in the system specific doc folder, where also an overview should be given. In most cases there will be several standard macros per system. If the documentation is very complex and likely to change with the version, then the documentation should be grouped also by version.

### ACCESS IN SAS

Apart from the knowledge of the location, the user needs to access the macros via SAS. There are different options available [2]. The most common option is to use include statements. This is very ineffective, as there could be quite a lot of standard macros which should be used and the list of files to include could be very large. As an enhancement an "include" file could be provided for all systems of the latest version, containing all required %include statements. This has the disadvantage, that this file needs to be included as well and using one for the latest versions will change as version change and could result in the problem of not knowing which systems has been used in older studies.

Another option with includes would be to use one include file which bundles all includes per system and version of all belonging standard macros. This has the advantage that the users only need to include one file for several macros of one macro system group. The only disadvantage is, that the path still needs to be addressed and for each system an include is needed.

A second major option would be to modify the SAS startup file. There are different ways, there is a configuration file or a so called AUTOEXEC file, where those file system macros can be included already on SAS startup. The major advantage is, that the users does not need to do anything in their programming and the macros are available. No hard coded paths are required. But there are two major disadvantages. The first problem is that the different macro systems are included fixed in one specified version. For this reason no further changes are possible to be able to reproduce analyses ten years later. The second disadvantage is that the macro systems are hidden for the users, so they can be unaware of the systems and versions used. To work with externals or the authorities having different environments has to be considered as well. So there needs to be at least a documentation what needs to be setup in an alternative environment.

Apart from includes used in user programs or in the startup of the environment, the standard macro systems can be used via a compiled catalog. This option has to consider, that the source itself is hidden, which could be a preferred solution when working with externals, but has the already mentioned disadvantages when used by internal user groups.

The last option is to use SASAUTOS. Those can also be initialized via user programs or the environment startup. A hierarchy of folder paths can be given, which is used to search macros which are not already compiled (an include would mean a compiled macro). For this, standard macros could be overwritten by user via an additional include or via an additional path in the SASAUTOS which has a higher level, then the standard system paths.

### EXAMPLE ACCESS IN SAS

It has been shown, that many ways are available on how to access standard macro system. Those options can also be mixed to provide an optimized way to use the continuously improving standard macro systems. The option in this example presents a way which is adjusted to match the needs of different parties whereas the transparency for the internal users is kept.

The user should decide which standard macro system in which version should be available for their evaluations. This should be clear in the user program as well as in the log file. To provide optimal support to use macro systems of a version, a system initialization macro has been created. This macro will be called *%initsystems*. For this macro the user does not need to know the storage location of standard macros, but just needs to use the system name and version number. This one macro is programmed that way, that it knows the storage location and prints those information to the log, so the user has the possibility to search for the corresponding documentation and macro details in that location. Furthermore the macro system is flexible enough to deal with new systems and new versions without any macro update.

To initialize standard system in the standards environment, the user could call for example:

```
%initsystems ( <systemname> = <version>, ... , <systemname> = <version> );  
%initsystems ( pharmabox = 1, toolbox = 2 );
```

This macro adds all via parameter given macro systems and version in the SASAUTOS, so they are available in the environment. The SASAUTOS has been used in this example to support a hierarchical order of available macros without the need of includes. After system initialization, a specific study working environment could be initialized,

## PhUSE 2009

where other paths are added in front of the SASAUTOS to use project or user specific macros first. This could be the case, when the user want a modified macro instead of a standard macro.

One problem with this strategy is, that this first macro needs to be available within the environment. This is done via the SAS configuration file. The %initsystems macro has been developed that way, that - for a specific environment - it will always be backward compatible and a usage apart the default environment is possible as well. So in the default company environment, the users does not need to make this first macro available. But when a different environment is used, e.g. a local SAS on a laptop, then this macro needs to be included in a user program. The macro is constructed that way, that the macro system storage structure is defined internally and only a start path needs to be further given as parameter when working in an alternative environment.

To initialize standard systems in any environment, the following example could be used:

```
%initsystems (path = <startpath>,  
              <systemname> = <version>, ... ,  
              <systemname> = <version>);
```

For this environment example this option is used, when internal users needs to run evaluations on other environments. An additional advantage of this way is, that when it comes to an environment change including different storage location structures, then just the %initsystems macro needs to be updated, to find and provide the correct macro systems in the correct version. User programs which would have alternatively hard coded paths could be avoided with that principle and for this even do not need an update according to different storage locations.

When evaluations should be performed by CROs, then a compiled macro catalogue is provided, containing all systems in the corresponding version they should used. This catalogue is protected and the initialization needs just to be exchanged in one place, where the %initsystem calls is exchanged with the catalog embedding.

### EXAMPLE SYSTEM INITIALIZATION MACRO

The system initialization macro should perform several tasks. The major task is, that systems are initialized in a specific version. System names and versions should be flexible. The macro should support two environments - the company environment and a flexible local environment. In case of working in an interactive SAS, macro system switches within one session should be possible as well.

As the systems which will come up are unknown to the macro, the *PARMBUFF* instead of macro parameters are used. When *PARMBUFF* is used, then the macro allows any kind of parameters used. SAS provides a macro variable, where the complete parameters including the brackets are stored in the *SYSPBUFF* macro variable. This variable can be used to analyze the system name and version number when a predefined syntax is used.

```
%MACRO initsystems () / PARMBUFF;  
  %PUT &syspbuff;  
%MEND initsystems;  
  
%initsystems(systemA = 1, systemB = 3);
```

Output: (systemA = 1, systemB = 3)

As two different environments should be supported, one parameter is used in the macro defining the environment itself. As the only aspect from the environment is the root macro systems directory, the parameter will be called *PATH*. When the parameter is not used, the root location will be from the company environment, otherwise the *PATH* is used. To support different servers or operating systems, the company root path can also be set conditionally, e.g. to a windows or a unix path.

```
%MACRO initsystems (path = ) / PARMBUFF;  
  %IF %LENGTH(&path) = 0  
  %THEN %DO;  
    %LET path = /server/systemroot;  
  %END;  
%MEND initsystems;
```

SAS stores included and used macros in a SASMACR macro catalog in the work library. When a macro is called, SAS searches at first within this macro catalog and then searches in the SASAUTOS paths. The system initialization macro should reset all macros and for this, the work catalog needs to be deleted. As there is a conflict of having a compiled macro trying to delete all compiled macros, the complete catalog could not be deleted via a macro. To get

## PhUSE 2009

a workaround, all catalog entries but the one executing macro will be deleted. The following source can be used to do so:

```
%MACRO initsystems (path = ) / PARMBUFF;
  %LOCAL l_allmac;

  PROC SQL NOPRINT;
    SELECT objname INTO :l_allmac SEPARATED BY " "
    FROM dictionary.catalogs
    WHERE libname EQ "WORK"
        AND memname EQ "SASMACR"
        AND objname NOT IN ( "INITSYSTEMS");
  QUIT;

  %IF %LENGTH(&l_allmac) > 0
  %THEN %DO;
    PROC CATALOG CAT=work.sasmacr ENTRYTYPE=macro;
      DELETE &l_allmac;
    RUN;QUIT;
  %END;
%MEND initsystems;
```

The most important part is the investigation of the systems and versions used and to use them in the SASAUTOS option. The SYSPBUFF is stored within another macro variable. As working with macro variables containing comma is sometimes complicated, all comma are replaced with a special character '#'. The brackets are removed as well. In the next step the number of macro parameter groups is investigated by counting the '#' and add one.

```
%LET systems=&syspbuf;
%LET systems=%SYSFUNC(TRANWRD(&systems,%STR(,),#));
%LET systems=%SYSFUNC(COMPRESS("&systems",'() '));
%IF %LENGTH(&systems)=0
%THEN %LET number=0;
%ELSE %LET number=%EVAL(%SYSFUNC(COUNT(&systems,#))+1);
```

Now a loop is created. Each parameter pair is used to investigate whether the corresponding macro system folder of the specified version exist and for this is added to the SASAUTOS and a note is printed to the log or an error message is printed. The folder is build up by an initial path followed by the system name (part upfront the equal sign of the parameter group), followed by a subfolder containing the system name and the version number (second part after the equal sign of the parameter group.) As the SASAUTOS should be modified later on again in a study initialization, the paths are stored within a macro variable for further use.

```
%DO l_i = 1 %TO &number;
  %LET system = %SCAN(&systems,&l_i,#);
  %LET l_path =
  "&path./systems/%SCAN(&system,1,'=')/%SCAN(&system,1,'=')%SCAN(&system,2,'=')";
  %IF %sysfunc(fileexist(&l_path)) = 0
  %THEN %DO;
    %IF %UPCASE(%SCAN(&system,1,'=')) NE %STR(PATH)
    %THEN %DO;
      %PUT %STR(ERROR): initsystems - The system "%SCAN(&system,1,'=')"
        in version "%SCAN(&system,2,'=')" does not exist.;
    %END;
  %END;
%ELSE %DO;
  %PUT - initsystems: The system "%SCAN(&system,1,'=')"
    in version "%SCAN(&system,2,'=')" can be used.;
  %IF %LENGTH(&sautosystems) = 0
  %THEN %DO;
    %LET sautosystems = &l_path;
  %END;
%ELSE %DO;
  %LET sautosystems = &sautosystems, &l_path;
%END;
```

# PhUSE 2009

```
%END;  
%END;
```

The last step is to set the SASAUTOS and additionally use the option MRECALL to force SAS to research for macros also when SAS already compiled a macro with that name earlier.

```
OPTIONS SASAUTOS = (&sautosystems , SASAUTOS);  
OPTIONS MRECALL;
```

## EXAMPLE MACRO SUMMARY

With this macro the users can now easily initialize the different standard macro systems. When it comes to updates, they have to be copied to the directory structure and can immediately be used without any initialization macro update, as it only reads the directory structure. When not backward compatible macro system updates are done, the old program runs are not affected, as they address older macro versions. Also a migration from one environment to another one would be supported, as only the folder structure or initial paths needs to be updated.

## CHANGE MANAGEMENT

By using the presented principle, it can be investigated which systems in which version had been used for evaluations. Updates of standard SAS macro systems can be done easily without the need to be always backward compatible while ensuring that old evaluation can be reconstructed. The users can profit any time from new enhancements which could come backward compatible in the same main version or with different changes in a new version.

Especially in the change management it is of importance, that the users can easily investigate the changes of systems. Regular trainings regarding updates are required as well, to allow all users to know the enhancements. While ensuring with Change Request Forms that user feedback is in-built into the macro systems, the acceptance of changes is also supported and helps the standard systems to be comfortable for any user.

## CONCLUSION

As shown in this paper, the creation of an environment supporting fast development and enhancements of standard macro systems should be managed, to benefit from ideas and realizations in a short time by supporting the end users in the best way. Especially as standardization becomes more and more important, but is not as constant as it should be in the pharmaceutical area, those flexible changes needs to be organized.

Another very important factor is the inclusion of the end user. As they have the most beneficial ideas of how the daily work for themselves can be enhanced. Wishes, ideas and critics from end users needs to be considered any time, to provide flexible and user friendly systems with a high acceptance rate. Standards management, change management and environment management needs always to consider the end users, as all systems rise and fall with them.

## REFERENCES

- [1] SAS documentation – “%MACRO Statements –  
<http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/macro-stmt.htm>
- [2] Ronald Fehd [2005], “A SASautos Companion: Reusing Macros”, SUGI 30 Paper 267-30  
<http://www2.sas.com/proceedings/sugi30/267-30.pdf>

## RECOMMENDED READING

TODO

## COMPLETE SOURCE FOR %INITSYSTEMS

```
%MACRO initsystems (path = ) / PARMBUFF;  
  %* initialize used macro variables;  
  %LOCAL l_allmac systems l_i l_path;  
  %GLOBAL sautosystems;  
  
  %* use the standard environment path or the given path;
```

## PhUSE 2009

```
%IF %LENGTH(&path) = 0
%THEN %DO;
    %LET path = /serverA/systemroot;
%END;

%* detele already compiled macro to research macros;
PROC SQL NOPRINT;
    SELECT objname INTO :l_allmac SEPARATED BY " "
    FROM dictionary.catalogs
    WHERE libname EQ "WORK"
        AND memname EQ "SASMACR"
        AND objname NOT IN ( "INITSYSTEMS");
QUIT;
%IF %LENGTH(&l_allmac) > 0
%THEN %DO;
    PROC CATALOG CAT=work.sasmacr ENTRYTYPE=macro;
        DELETE &l_allmac;
    RUN;QUIT;
%END;

%* read system buffer;
%LET systems=&syspbuff;
%LET systems=%SYSFUNC(TRANWRD(&systems,%STR(, ),#));
%LET systems=%SYSFUNC(COMPRESS("&systems",'() '));
%IF %LENGTH(&systems)=0
    %THEN %LET number=0;
    %ELSE %LET number=%EVAL(%SYSFUNC(COUNT(&systems,#))+1);

%* create new sautosystems;
%DO l_i = 1 %TO &number;
    %LET system = %SCAN(&systems,&l_i,#);
    %LET l_path =
"&path./systems/%SCAN(&system,1,'=')/%SCAN(&system,1,'=')%SCAN(&system,2,'=')";
    %PUT &l_path;
    %IF %sysfunc(fileexist(&l_path)) = 0
    %THEN %DO;
        %IF %UPCASE(%SCAN(&system,1,'=')) NE %STR(PATH)
        %THEN %DO;
            %PUT %STR(ERROR): initsystems - The system "%SCAN(&system,1,'=')"
                in version "%SCAN(&system,2,'=')" does not exist.;
        %END;
    %END;
    %ELSE %DO;
        %PUT - initsystems: The system "%SCAN(&system,1,'=')"
            in version "%SCAN(&system,2,'=')" can be used.;
        %IF %LENGTH(&sautosystems) = 0
        %THEN %DO;
            %LET sautosystems = &l_path;
        %END;
        %ELSE %DO;
            %LET sautosystems = &sautosystems, &l_path;
        %END;
    %END;
%END;

%* use sautosystems in the SASAUTOS;
OPTIONS SASAUTOS = (&sautosystems , SASAUTOS);
OPTIONS MRECALL;

%* print user information;
%PUT;
```

## PhUSE 2009

```
%PUT Environment path : "&path";  
%PUT Environment Path(s): &sautosystems;  
%PUT;
```

```
%MEND initsystems;
```

### **CONTACT INFORMATION**

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

Katja Glaß  
Bayer Schering Pharma AG  
Sellerstr. 31  
13342 Berlin  
Katja.Glass@bayerhealthcare.com

Brand and product names are trademarks of their respective companies.