PhUSE 2015

Paper TS02

Concept and Implementation of an Efficient Version Control System for SAS® Programs in Clinical Development

Matthias Post, Accovion, Eschborn, Germany
Dirk Spruck, Accovion, Marburg, Germany

ABSTRACT

Version control of SAS® programs is a necessity in the preparation and analysis of clinical trial data. The life cycle of all programs in production should be completely documented to meet regulatory requirements.

Furthermore productive output should only be generated using final and validated programs. After productive output is generated the respective program version must be available for later reference and reproducibility of results. For the statistical programming and biostatistics group at Accovion a version control system was built using the following elements:

- A pre-specified folder structure in the SAS® programming environment.
- A collection of command-line tools for the management of validation status and access rights as well as for the actual filing and versioning of programs.
- A database for tracking all changes of program status and version.

This presentation will explain the concept of the tools and structures for an efficient version control system. It will be demonstrated how such a system can be implemented using only typically available software components.

INTRODUCTION

As a contract research organization (CRO) Accovion is responsible for the planning, conduct, and analysis of clinical studies. SAS® programs are used, e.g. for the generation of datasets, tables, listings and graphs.

Programs generating output, which are then passed on to the sponsor and may be submitted to a regulatory agency must undergo a formal validation process. Typical validation steps include a test run of the programs, a review of the code, the log file and the output. In addition independent double programming with comparison of the generated output may be required.

To answer questions from the sponsor or the regulatory agencies it is required to have the complete life cycle of programs in production documented and available. In addition the final analysis must be re-producible at any time. Therefore it is required to not only keep the results of an analysis and the underlying data but also the programs.

The program development calls for additional requirements, e.g. programmers need to have access to earlier program versions, comparison between program versions needs to be easy and it should be possible to save certain milestones during the development of the code.

Based on these internal and external requirements Accovion decided to build their own custom-tailored version control system for SAS programs.

THE SAS ARCHITECTURE AT ACCOVION

In 2010 Accovion's programming environment was moved to a LINUX-server with a validated installation of SAS software[1]. This LINUX instance serves as the central file repository storing SAS datasets, outputs, programs and libraries.

A second SAS installation is available in a Windows-Citrix Server environment. SAS/Connect is used to link up this instance with the LINUX SAS installation. The Windows environment can be used for convenient development and editing of SAS programs. However final programs are always executed in batch mode under LINUX.

Via SAMBA-Share services the LINUX file system is mapped and available in the Windows server environment. Although programmers have access to all programs and data in the Windows environment it is only used for development and not for production.

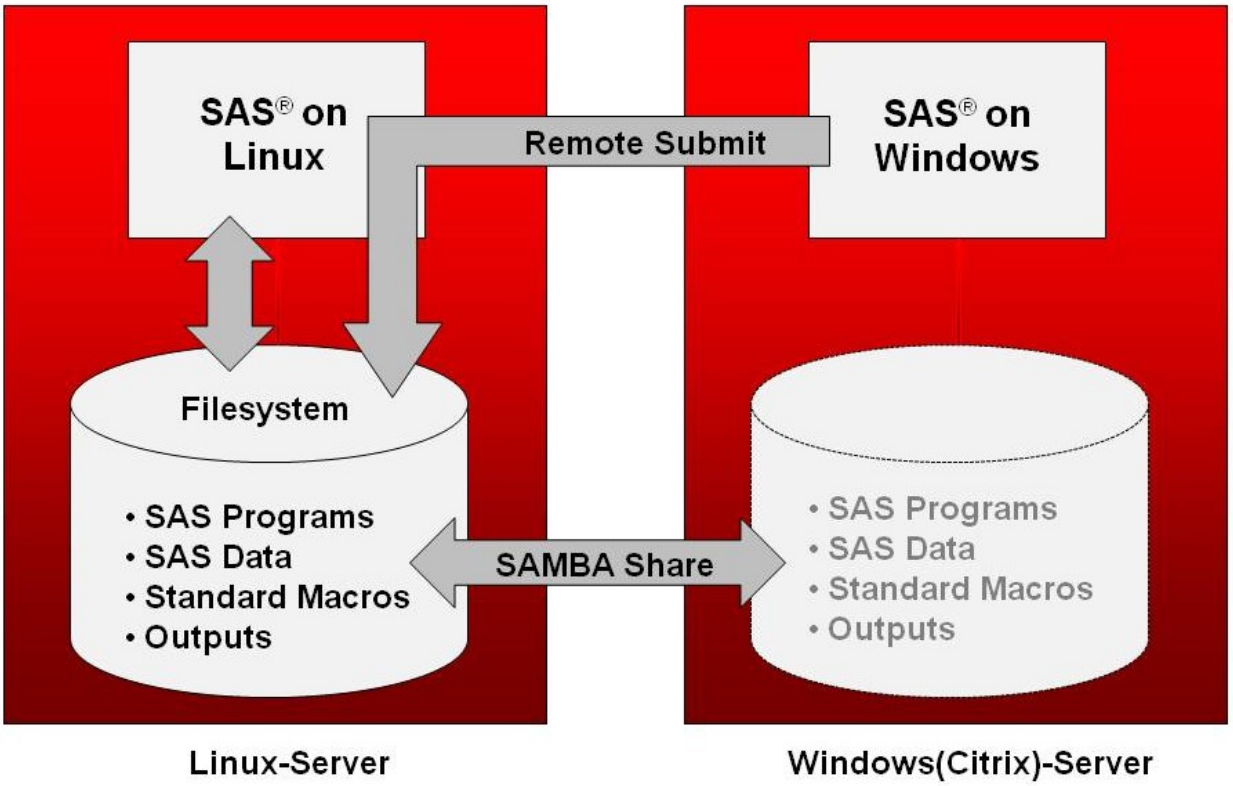

Figure 1: SAS®-Architecture at Accovion

VALIDATION LEVELS FOR SAS® PROGRAMS

To support the development and validation of SAS programs a folder structure with three levels is used. Programs and macros are initially written in the development level (DEV-level). After completion of the initial programming the program is move to the validation level (VAL-level). While the program is in the VAL-level program and output will be validated by another programmer or statistician according to Accovion's standard operating procedure (SOP). After successful validation the program will be moved to the production level (PROD-level), from where it can be used to produce validated output.

Because programs in the VAL- and PROD-levels are read-only no changes can be made there. In case the validation result requires a change of the program, it must be moved to the DEV-level to be corrected. The function for moving programs to VAL or PROD is called LOCK. The reverse function is called UNLOCK and moves a program back to the DEV-level. UNLOCK re-assigns write-access to the program for the program owner.

Within the file system the three validation levels are implemented as folders under the respective project/study folder. Within each level there are sub-folders in which the programs are organized by type and purpose or other relevant criteria. These sub-folder trees are identical underneath each validation level folder. For example programs that are stored here:

```
/project/study/DEV/reports/safety
```

will be move to to

```
/project/study/VAL/reports/safety
```

using the LOCK function.

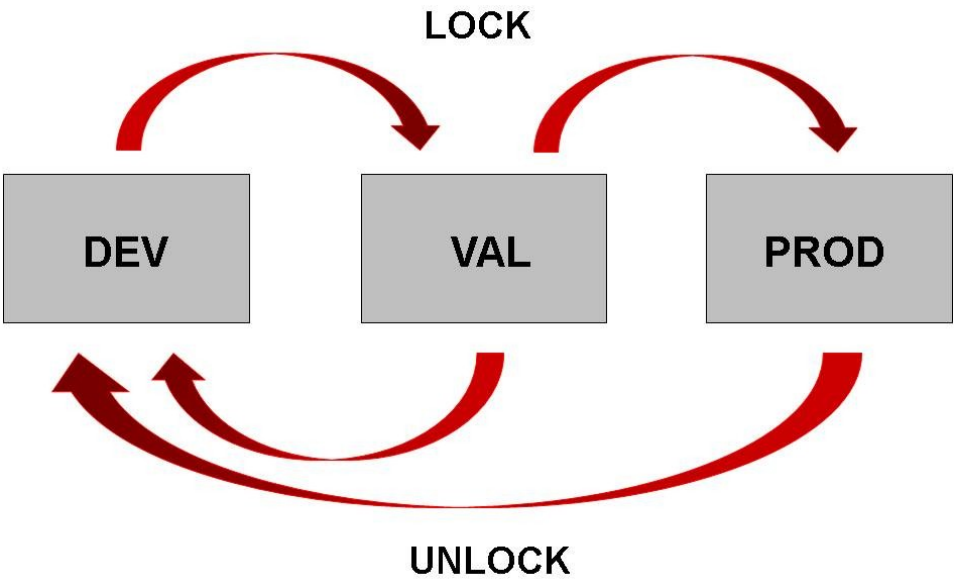

Figure 2: The three validation levels

THE VERSION CONTROL SYSTEM

For the development of a version control system based on the described validation level folder structure the following requirements were most important:

- Shifting of programs between the different validation levels.
- Assigning appropriate read and write access rights.
- Automatic generation of backup of the code as soon as status of a program changes, i.e. by using the LOCK function).
- Generation of unique version numbers for the backup copies.
- Recording of any movement of programs between the validation levels in a database.

The mentioned LOCK and UNLOCK function as well as the additional functions described later in this paper were developed as LINUX command line tools. Statistical Programmers at Accovion are used to work in the command line environment where additional tools are available within the programming environment. The commands can also be called from within the SAS-session using x-commands. So far there has not been a real need to develop a graphical user interface.

For the development of the functions only software that was already available within our IT setting was used. No additional software had to be purchased or installed.

Since most of the functionality is based on operations on the LINUX file system the command line tools were implemented as BASH shell scripts. For the writing of records into the history-database an insert-statement is generated by the shell scripts. This insert statement is then sent to the ORACLE® database by an SQLPLUS client. The database was implemented using an ORACLE® instance already available and consists of a single table. However any other relational database could have been used as well.

BACKUP-COPIES AND VERSION NUMBERS

Except for the UNLOCK function all functions of the versioning system create a backup-copy of the respective program. These backup-copies are stored in a dedicated history folder structure within a project. The sub-folders structure of the program's origin is mirrored in the history folder. In addition within this folder structure one folder per program is generated, which contains all backup-copies of the respective program. The name of this folder is the program name without the .sas extension.

Example:

All backups of a program

/project/study/**DEV**/reports/safety/lab_listing.sas

are stored in the history directory

/project/study/**HISTORY**/reports/safety/**lab_listing**

The history directory will be created for the respective program in case it does not already exist, e.g. during the first backup of the program.

A three-level version number, which is attached to the original file name, is assigned to the backup copy of the program:

<filename>.sas → *<filename>*_v***PP.VV.DD***.sas

- The lowest level DD of the version number is increased by 1 in case a new backup copy of the program is created and the program remains in the development level DEV, e.g. by using SAVE, MODIFY; or RENAME, which will be discussed later in this paper.
- The middle level VV of the version number is increased when the LOCK function is used to move the program from DEV to the validation level VAL. At the same time the lowest level DD of the version number is re-set to 00.
- The highest level PP of the version number is increased when the LOCK function is used to move the program from VAL to the production level PROD. At the same time both lower levels parts of the version number VV and DD are re-set to 00.

All backup copies are set to read-only.

DESCRIPTION OF THE FUNCTIONS

LOCK

The LOCK function moves a program to the next validation level (DEV -> VAL or VAL -> PROD)

Syntax

```
lock [-l] [-c "comment"] filename.ext [filename2.ext …]
```

Parameters

-l In case that a .log or .lst file exists with the same name as the SAS program a backup copy of the .log and/or .lst file is created. The files will be stored in the same history folder as the SAS program and the same version number will also be used.

-c "comment"
A short comment, which is stored in the database can be added to the LOCK function call.

Functionality

- Moves one or multiple programs to the next validation level and sets access rights to read-only.
- Creates a backup copy in the history folder for each program.
- Creates an entry in the history database.

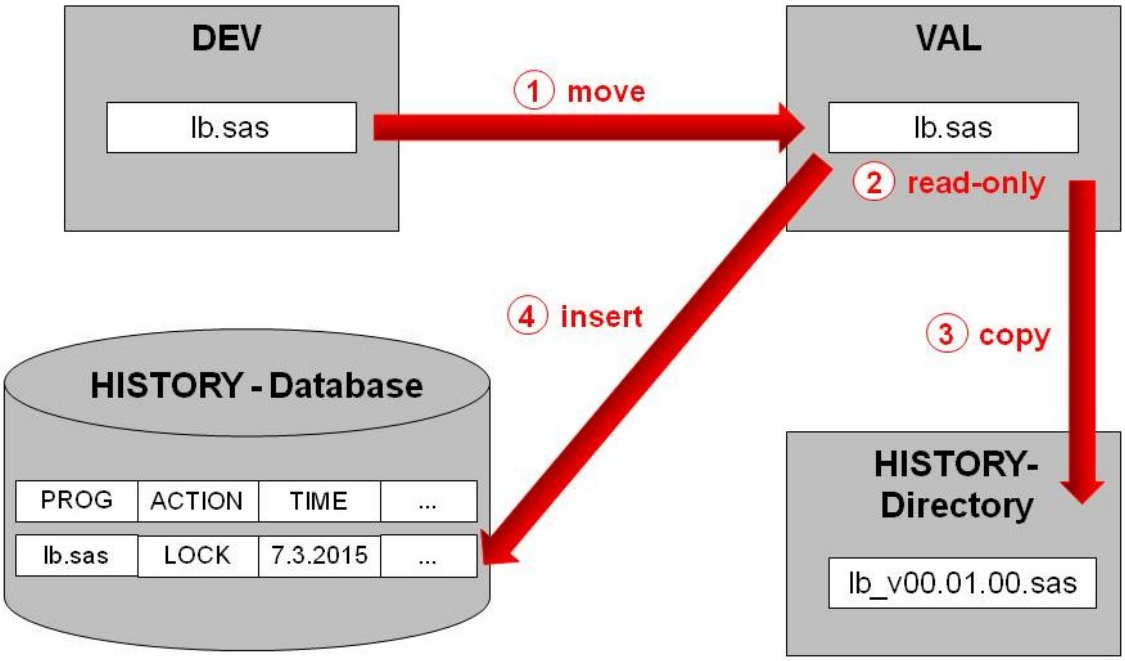

Figure 3: Sample of the LOCK functionality

UNLOCK

The UNLOCK function moves a program back into the development level DEV.

Syntax

```
unlock [-c "comment"] filename.ext [filename2.ext …]
```

Functionality

- Moves one or multiple programs from the VAL or PROD level back into the development level DEV.
- Assigns ownership with write access for the program to the user of the function.
- Creates an entry in the history database.
- No backup copy is created because the program was already read-only in the VAL and PROD levels.

RENAME

The RENAME function is used to rename a program while keeping a link to the original program name. Programs in the history folder and in the history database are identified by a program name and the respective pathname within the project directory. During program development it may become necessary to rename a program or to move it into a new subdirectory within the project. In this case it is necessary to keep a link to the original backup files and database entries under the old name. Therefore the RENAME function was created to have a controlled process available for renaming and moving programs within a project.

Syntax and functionality of the RENAME function are similar to the UNIX-command “mv”.

Syntax

```
rename [-c "comment"] oldfilename.ext [newdir/]newfilename.ext
```

or

```
rename [-c "comment"] filename.ext  [filename2.ext …] newdir
```

Functionality

- Renames and/or moves one or multiple programs into another directory within the project.
- Creates a backup copy of each renamed/moved file in the history folder.
- Moves existing backup copies to the new history folder. In this case a symbolic link (shortcut) pointing to the new folder is created in the old history folder.
- Creates an entry in the history database. The old program name and location are documented. Existing database entries belonging to the old program name will not be modified.

MODIFY

In the development level DEV the following access rights apply to SAS programs:

```
permission   owner group     filename
rw- r-- ---  user1 projectX  addm.sas
rw- r-- ---  user2 projectX  adlb.sas
```

Only the owner of the program has permission to modify the file. Members of the project team have only read access, which is managed through group membership.

Before a programmer can modify a program he/she needs to become the owner of the program. The MODIFY function was developed in order to have a controlled process available for change of ownership and setting the correct access rights for programs. In case the ownership changes MODIFY automatically creates a backup copy of the file.

Syntax

```
modify [-c "comment"] filename.ext [filename2.ext …]
```

Functionality

- The user of the MODIFY function becomes the owner of the program. The user needs to have read access to the file through his group membership.
- Creates a backup copy of each “modified” file in the history folder.
- Creates an entry in the history database. The name of the previous owner is documented.
- The previous owner of the program is informed of the change of ownership by email. The email contains information on version number and location of his/her last program version.

If the UNLOCK and RENAME functions are executed on a program that is owned by another user, they will also perform a change of ownership. In this case the previous owner is informed via email.

SAVE

The SAVE function can be used to create an ad-hoc backup copy of a program without moving the program to a different validation level. This may be desirable in case an intermediate state of the program should be saved for later reference during the development phase of a program.

Syntax

```
save [-l] [-c "comment"] filename.ext [filename2.ext …]
```

Parameter

-l In case that a .log or .lst file exists with the same name as the SAS program a backup copy of the .log and/or .lst file is created. The files will be stored in the same history folder as the SAS program and the same version number will also be used.

Functionality

- Creates a backup copy of one or multiple programs in the history folder
- Creates an entry in the history database.

ADDITIONAL FUNCTIONALITY

- For all functions of the version control system LINUX typical wildcards can be used in the program name to address multiple files at the same time, e.g. *, ? and [].
- The version control system was developed for SAS programs. However it can be used with any type of files, e.g. CSV- or EXCEL files can be versioned as well.
- Records which cannot be added to the history database are saved in a separate directory and will be uploaded later. This can happen in case the history database is not available, e.g. because of a system backup.

THE HISTORY DATABASE

All activities of the version control system are documented in the history database. For each action and file one record is generated.

The database consists of a single table, which is indexed to increase performance. Write access to the database is performed by a generic database account, which has INSERT rights to the database only. Individual authentication by the user is not necessary. Existing entries in the database cannot be changed or deleted.

The following information is documented in the history database:

Table 1: Contents of the history database

Variable name	Typ	Description
PROJECT	Text	Project name.
STUDY	Text	Study name.
FILENAME	Text	Program name.
PATH	Text	Pathname relative to study directory.
OLDPATH	Text	Absolute pathname before the action.
NEWPATH	Text	Absolute pathname after the action.
SOURCE	Text	Validation level before the action (DEV, VAL, or PROD).
TARGET	Text	Validation level after the action (DEV, VAL, or PROD).
ARCFNAME	Text	Name of backup copy (program name + version number).
ARCPATH	Text	Pathname of backup copy.
VERSION_PROD	Integer	Highest (production) level of version number (PP).
VERSION_VAL	Integer	Middle (validation) level of version number (VV).
VERSION_DEV	Integer	Lowest (development) level of version number (DD).
ACTION	Text	Action: "LOCK", "UNLOCK", “RENAME”, "MODIFY" or “SAVE”.
USERID	Text	ID of the user who executed the action.
PREVUSERID	Text	Previous owner of program in case of an owner change.
PREVNAME	Text	Previous path and program name in case of a rename.
TIMESTAMP	Date	Time of action in format DD.MM.YYYY HH:MM:SS
TIMEFILE	Date	Last modification date of program.
COMMENT	Text	Comment, which was provided by the –c option.
SAVELOGS	Text	Flag is „L“ in case the log- and lst-files were saved by using the –l option.
LOGFILENAME	Text	If applicable, name of saved log-file.
LSTFILENAME	Text	If applicable, name of saved lst-file.

SAS ACCESS TO ORACLE is used to read the contents of the history database. A piece of code is integrated in the autoexec.sas to make database entries for the specific project/study available in SAS. Using standard SAS search and filter functionality the lifecycle of a program can easily be analyzed.

Reports can be generated within SAS and frequently asked questions regarding the validation status of a program can easily be answered.

CONCLUSION

The version control system was built within a short development timeframe with software components already available in-house. The system is intuitive, easy to use, and meets the requirements of the programming group as well as regulatory requirements.

With only 1500 lines of productive code the development effort was not very high. Since the development was completely done in-house the system can easily be modified to meet new requirements. Because of the low complexity the system is mostly independent from releases of the underlying software components (LINUX, ORACLE®, SAS®).

One of the drawbacks of the system is that theoretically the users can avoid the system and use traditional ways to copy programs between the program validation levels without using the version control scripts. In this case the contents of the history database would not reflect the actual content of the file system.

Therefore it is essential that the version control system is easy to use and that the users are adequately trained. However, programmers at Accovion were already used to work in the script based environment because the preceding system was already script based with similar functionality.

The newly developed version control system is lean and transparent. It has been in use for more than 18 months and is appreciated very much by the users. Especially the comparison of program versions proved to become quick and easy and is very frequently used in the validation process.

REFERENCES

1. Spruck, D., Kanz, J. “Migrating a SAS Programming Environment to a Virtual Linux Server” PhUSE Annual Conference 2010. Paper TS01.

ACKNOWLEDGMENTS

The authors would like to thank all Accovion colleagues working on the development of the versioning system and contributing to the paper. Special thanks go to Eddie Arnold, Judith Kanz, Michael Ludwig, Nicola Tambascia and Sven Wichmann (Statistical Programming), Paul Gensler and Stefano Fantini (IT) for their fantastic support.

CONTACT INFORMATION

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

Matthias Post
Accovion, A Clinipace Worldwide Company
Helfmann-Park 10
65760 Eschborn
Germany
Email: Matthias.post@accovion.com

Dirk Spruck
Accovion, A Clinipace Worldwide Company
Softwarecenter 3
35037 Marburg
Germany
Email: dirk.spruck@accovion.com